UNIVERSITÉ DE NANTES
FACULTÉ DES SCIENCES ET DES TECHNIQUES

———

ÉCOLE DOCTORALE SCIENCES & TECHNOLOGIES
DE L'INFORMATION ET MATHÉMATIQUES – STIM

Année 2013

Ordre et désordre dans l'algorithmique du génome

———

Algorithmic Aspects of Genome Rearrangements

THÈSE DE DOCTORAT
Discipline : Informatique et applications
Spécialité : Informatique

Présentée
et soutenue publiquement par

Laurent BULTEAU

Le 11 juillet 2013, devant le jury ci-dessous

Présidente	Cristina BAZGAN, Professeur des Universités, Université Paris Dauphine
Rapporteurs	Rolf NIEDERMEIER, Professeur, Technische Universität, Berlin
	Dieter KRATSCH, Professeur des Universités, Université de Lorraine, Metz
Examinateur	Michel HABIB, Professeur des Universités, Université Paris Diderot – Paris 7

Directeurs de thèse :

Guillaume FERTIN, Professeur des Universités, Université de Nantes
Irena RUSU, Professeur des Universités, Université de Nantes

Contents

List of Figures

List of Algorithms

Synthèse

Abstract (French Section)

Remerciements

Je tiens à remercier en premier lieu mes directeurs de thèse Guillaume Fertin et Irena Rusu, pour leur présence et leur implication depuis mon arrivée à Nantes. La confiance qu'ils ont su me montrer, leur clairvoyance dans les choix des sujets et dans les pistes à explorer, ainsi que leur rigueur et leur patience dans la relecture de mes papiers ont été les moteurs de la réussite de cette thèse.

Je remercie également toutes les personnes avec qui j'ai eu la chance de travailler : Minghui Jiang, Pedro Tejada aux États-Unis ; Riccardo Dondi et Anna Paola Carrieri à Milan ; à Marne-la-Vallée : Stéphane Vialette, Guillaume Blin, et maintenant Anthony Labarre ; Antoine Thomas à Lille ; Christian Komusiewicz à Berlin puis à Nantes ; et plus récemment Binhai Zhu en Chine.

Je remercie Dieter Kratch, Rolf Niedermeier, Cristina Bazgan et Michel Habib pour avoir accepté d'évaluer mes travaux. Plus particulièrement, merci à Michel Habib pour m'avoir fait connaître le LINA. Enfin, à Rolf Niedermeier, pour me permettre de continuer en post-doc dans son équipe à Berlin : Danke Schön !

Je tiens à remercier trois excellents professeurs que j'ai eus bien avant d'entrer en thèse : Jean-Paul Courant, Bernard Jouet et Jean-Louis Liters. Bien au-delà des quelques connaissances qui ont pu m'être utiles, j'ai reçu d'eux la curiosité, l'efficacité et la rigueur mathématique qui m'ont permis d'aborder cette thèse en toute sérénité.

Pour la structure de recherche dont j'ai bénéficié, je remercie le LINA en général, et en particulier Anne-Françoise Quin, Annie Boilot et Annie Lardenois.

Un grand merci à tous les membres de l'équipe Combi. De même, merci à chacun des doctorants du LINA que j'ai pu côtoyer. Ils ont donné une atmosphère unique au labo, et sans eux les après-midis auraient été bien trop longues.

Merci à tous les grimpeurs, férus ou occasionnels : Damien, Jean-Yves, Ophélie, Audrey, Matthieu, Alexandre, Marie, Odeline, Bertrand, et surtout Lionel. L'escalade a beaucoup de points communs avec la recherche en informatique : on cherche à résoudre des problèmes, on réfléchit avant de se lancer dans une voie mais il reste une bonne part d'improvisation, et on doit penser avec ses pieds (bon, ça, c'est peut-être une différence...).

Et parce que les années de thèse ne se vivent pas que dans un labo de recherche (ou sur un mur d'escalade) je tiens à remercier tout particulièrement Audrey, Kenny, Cécile, Pascale, Camille, Amaury, Pauline, Pierre, Gaëlle, Laura, Delphine, Auriane et Marine pour leur amitié.

Enfin, mes remerciements les plus profonds vont à Tigrane, à ma sœur Valérie avec Nicolas et Lucas, et à mes parents. Pour être là, vous-même, tout simplement.

Introduction

La bioinformatique, domaine à l'intersection de la biologie et de l'informatique, s'intéresse à l'extraction et au traitement de l'information dans les données biologiques afin de mieux comprendre le vivant. D'une part, la biologie apporte des données obtenues via des expériences et pose les questions pertinentes permettant de mieux comprendre le sujet traité. D'autre part, l'informatique fournit des méthodes formelles pour analyser les problèmes et pour produire des solutions aussi précisément et efficacement que possible. Les bénéfices sont directs pour les biologistes qui peuvent ainsi vérifier leurs hypothèses et faire des prédictions de façon plus fiable et rapide. Les bénéfices sont également importants pour les informaticiens, à qui l'on présente des défis inédits. La variété des problèmes contribue à l'exploration de structures de données originales et à la découverte de nouvelles méthodes algorithmiques, enrichissant ainsi le domaine des sciences informatiques avec une palette agrandie d'approches possibles pour d'autres problèmes.

L'utilisation de la bioinformatique devient encore plus incontournable lorsque les données à analyser deviennent de plus en plus grandes. L'exemple le plus révélateur est le séquençage ADN : il est maintenant possible de lire des séquences d'ADN de chromosomes à très haut débit (par exemple, le Beijing Genome Institute produit 10 téraoctets de séquences de nucléotides brutes quotidiennement [109]). Les séquences ADN sont au cœur de la majorité des processus biochimiques. En revanche, reconstruire parfaitement une séquence génomique relève encore de l'utopie, tout comme, a fortiori, comprendre toutes les fonctions biologiques intriquées qui découlent de ces séquences. Un autre domaine important est l'analyse de réseaux où l'on cherche à identifier les interactions parfois complexes entre les différentes molécules de la cellule (protéines, ARN, métabolites, etc). Les données d'entrée prennent la forme d'un ensemble de réactions atomiques, telles que "la protéine A peut transformer un métabolite B en un métabolite C" ou "les protéines A et B peuvent se combiner pour former une protéine C". Le but est alors de comprendre les processus à grande échelle, tels que les cycles cellulaires ou la synthèse de protéines, avec l'objectif in fine de produire des médicaments agissant avec précision dans ces processus.

Dans cette thèse, nous cherchons à apporter des réponses combinatoires à des problèmes provenant de la génomique comparative. Le point commun aux problèmes étudiés est la présence en entrée d'information génétique, généralement sous la forme de séquences de gènes ou plus généralement de marqueurs, provenant de deux espèces différentes. Les deux espèces ont évolué naturellement depuis un ancêtre commun, et présentent maintenant un certain nombre de similarités : les objectifs des problèmes de génomique comparative sont de repérer ces similarités, de les mesurer, et de les utiliser afin de recueillir de l'information biologique pertinente.

En fonction du modèle utilisé pour décrire les génomes et des questions biologiques posées, nous obtenons des problèmes complexes et divers qui peuvent être étudiés à la lumière des méthodes habituelles de complexité algorithmique. Pour chaque problème, nous cherchons à le résoudre par un algorithme de complexité polynomiale ou, à défaut, à prouver qu'il est NP-difficile. Dans ce dernier cas, des algorithmes efficaces peuvent être recherchés dans deux directions : soit parmi les algorithmes d'approximation (i.e. des algorithmes polynomiaux donnant une solution sous-optimale, mais avec un ratio d'erreur borné), soit parmi les algorithmes paramétrés (i.e. des algorithmes exacts dont la complexité, exponentielle, reste néanmoins suffisamment faible pour de petites valeurs d'un paramètre correctement choisi).

Chaque chapitre du manuscrit est consacré à un problème spécifique de génomique comparative, ces chapitres sont organisés en trois parties en fonction des modèles utilisés pour décrire les génomes d'entrée. La partie I se focalise sur des distances de réarrangement utilisant le modèle le plus favorable, où chaque gène est unique dans le génome. Dans la partie II nous explorons deux problèmes où les séquences d'entrée peuvent avoir des duplications. Enfin, dans la partie III nous ouvrons notre étude à des problèmes où les données sont soit incomplètes, soit erronées. La motivation biologique et les résultats principaux de la génomique comparative sont présentés dans cette introduction, et un état de l'art centré sur chaque problème est présenté dans le chapitre correspondant.

Connaissances utiles en biologie.

Alors que l'information génétique portée par l'ADN est copiée de cellule en cellule et d'ancêtre en descendant par des processus biochimiques répétitifs, un grand nombre d'erreurs peuvent se produire : ce sont les étapes de l'évolution. Ces étapes peuvent être absolument sans conséquence, mais peuvent également mener à la "découverte" de nouvelles possibilités biologiques... ou à la mort de la cellule. De plus, elles peuvent se produire aussi bien à petite échelle, affectant seulement quelques nucléotides dans l'ADN, ou à plus grande échelle et provoquant la coupure, la réinsertion ou la duplication de grands segments d'ADN.

Une notion clé pour l'étude de l'évolution génétique est celle de blocs de synténie, c'est-à-dire, de bandes d'ADN ayant des contenus similaires chez des espèces différentes : nous considérons que de tels blocs étaient présents chez l'ancêtre commun à ces espèces, et ont été conservés tout au long de l'évolution. Deux espèces ayant des blocs de synténie petits et dispersés ont probablement évolué en tant qu'espèces différentes pendant plus longtemps, ce qui implique qu'elles ont un ancêtre commun relativement vieux. De plus, si un bloc de synténie inhabituellement long est présent chez un grand nombre d'espèces, on peut supposer que les gènes qu'il contient ont, d'une façon ou d'une autre, des fonctions liées (les protéines codées interagissent, un gène en régule un autre, etc.). La notion complémentaire au bloc de synténie est celle de point de cassure. Formellement, en comparant deux génomes, il y a un point de cassure entre deux gènes consécutifs d'un génome si les même gènes sont séparés dans l'autre génome, ce qui implique que de tels gènes n'appartiennent pas à un même bloc de synténie. Le nombre de points de cassure est facilement calculable – à condition que chaque gène soit présent exactement une fois dans chaque génome et que les données soient sans erreur – et est considéré comme une première mesure de dissimilarité entre deux génomes (la *distance de cassure*).

Connaissances utiles en complexité algorithmique.

Nous considérons que le lecteur est déjà familiarisé avec les notions de complexité algorithmique telles que les algorithmes polynomiaux exacts et d'approximation, la NP-difficulté et l'APX-difficulté. En quelques mots, on considère qu'un algorithme est efficace si, pour résoudre un problème dont les données ont une taille n, il a besoin d'un temps borné par une fonction polynomiale en n. P est la classe des problèmes pouvant être résolus de façon exacte par un tel algorithme efficace. Par opposition, il est conjecturé qu'il n'existe pas de méthode de résolution efficace pour les problèmes NP-difficiles.

Une stratégie habituelle pour les problèmes d'optimisation NP-difficiles est de créer des algorithmes d'approximation, i.e. des algorithmes donnant une solution *faisable* en temps polynomial, mais dont le score peut être à un facteur r d'écart du score optimal (r devant être aussi proche de 1 que possible). L'APX-difficulté est la preuve d'une borne inférieure sur la valeur de r ; sans une telle borne, il est possible d'envisager un schéma d'approximation en temps polynomial, i.e. un algorithme d'approximation pouvant atteindre n'importe quel ratio d'approximation $1+\epsilon$ pour $\epsilon > 0$.

Une autre approche possible est de créer des algorithmes paramétrés (FPT, pour *fixed parameter tractable*), dont l'objectif est de résoudre de façon exacte mais néanmoins efficace certains problèmes NP-difficiles. La première étape consiste à identifier un paramètre (généralement noté k) qui résume le niveau de complexité du problème pour chaque instance spécifique. Ensuite, un algorithme FPT est un algorithme qui nécessite un temps majoré par $O(f(k)n^d)$, où d est un entier fixé et f peut être n'importe quelle fonction (en général une fonction exponentielle) n'ayant pas de dépendance en n. De tels algorithmes requièrent un temps exponentiel dans les cas "au pire" où k peut être arbitrairement grand, mais ils sont efficaces lorsque k est borné à de faibles valeurs, ce qui devrait être le cas pour la majorité des données réelles.

Pour des informations plus détaillées sur les bases théoriques, les techniques et les résultats classiques, nous renvoyons le lecteur à Garey & Johnson [75] pour la théorie de la NP-difficulté, à Papadimitriou & Yannakakis [108] pour l'approximabilité, et à Niedermeier [107] pour la complexité paramétrée.

Partie I – Distances entre permutations

Dans la première partie, nous considérons que les génomes peuvent être modélisés par des permutations, c'est-à-dire qu'il existe une correspondance parfaite entre les gènes de l'un et de l'autre génome, et seul l'ordre des gènes diffère. Plus précisément, nous considérons les permutations signées et non signées : les permutations non signées sont les permutations habituelles sur l'intervalle $[\![1\,;\,n]\!]$, et, pour les permutations signées, une information supplémentaire représentant l'orientation sur le chromosome est associée à chaque gène, et est marquée avec un symbole $+$ ou $-$. Ce modèle peut manquer de généralité, mais il permet le calcul de nombreuses mesures de distance, à commencer par la distance de cassure. D'autres exemples habituels sont les intervalles communs et conservés [121, 86, 18], ou les distances de Hamming et de Levenshtein [82, 101]. Dans les années 1990, avec notamment [97, 13], furent proposées les *distances de réarrangement* : on considère une opération d'évolution naturelle et on recherche la plus petite séquence de telles opérations permettant de transformer un génome en l'autre. Le scénario obtenu fournit des informations précieuses sur l'évolution des deux espèces depuis leur ancêtre commun.

Plusieurs opérations pertinentes d'un point de vue biologique peuvent être considérées pour les distances de réarrangement. Les plus étudiées sont probablement les inversions (une sous-séquence du chromosome est coupée et réinsérée au même endroit dans la direction opposée), et les transpositions (une sous-séquence est coupée et réinsérée à un autre endroit, avec la même direction). Plus récemment, les opérations de DCJ (pour Double-Cut-and-Join) ont été introduites [124, 17] : les chromosomes sont coupés à deux endroits, et les "bouts" générés peuvent être recollés de n'importe quelle manière (cette opération requiert un modèle de génome plus général, où les permutations circulaires sont considérées). Enfin, d'autres opérations possibles incluent les transinversions (transpositions avec en plus une inversion de la séquence coupée), les échanges de blocs (deux sous-séquences échangent leurs positions), et les variantes préfixes de chacune de ces opérations (la séquence coupée doit contenir une extrémité du chromosome).

Pour la distance d'inversions signée, c'est-à-dire la distance de réarrangement où les opérations considérées sont les inversions et où les génomes sont modélisés par des permutations signées, Bafna & Pevzner [14] ont produit un algorithme polynomial qui a depuis été amélioré en un algorithme linéaire [10]. La distance de DCJ signée et la distance d'échange de blocs bénéficient également d'algorithmes exacts efficaces [124, 103]. Par contre, le calcul des distances d'inversions et de DCJ non signées est NP-difficile [42, 43]. La complexité est donc ouverte pour deux opérations importantes : les transpositions, dont l'apparente simplicité peut être trompeuse, et les inversions préfixes, pour lesquelles le problème combinatoire avait été introduit dès 1975 sous le nom de *retournement de crêpes* [64]. Nous déterminons dans les Chapitres 1 et 2 la complexité de ces deux problèmes (Tri par transpositions et Tri par inversions préfixes) en prouvant qu'ils sont chacun NP-difficiles.

Partie II – Distances entre chaînes

Dans la seconde partie, nous levons la contrainte d'unicité et considérons des modèles où les gènes peuvent avoir plusieurs copies dans chaque génome (un génome n'est plus représenté par une permutation, mais par une séquence arbitraire, ou *chaîne*). En effet, des événements de duplication peuvent se produire et engendrer des gènes indistinguables. Si un gène a plusieurs copies chez deux espèces, deux scénarios principaux sont possibles. Dans le premier, les duplications ont eu lieu avant la spéciation, c'est-à-dire que l'ancêtre commun possédait déjà toutes les copies du gène. Dans ce cas, il est intéressant de retrouver la correspondance des différentes copies dans les deux espèces. Par exemple, si les deux génomes considérés sont $G_1 = axbcxd$ et $G_2 = cxdaxb$, il est probable que l'ancêtre commun contenait à la fois les sous-séquences axb et cxd, ainsi la première copie de x dans G_1 devrait être mise en correspondance avec la seconde dans G_2, et la seconde copie dans G_1 avec la première dans G_2. Ce scénario est appelé le *modèle complet* [25]. Le scénario opposé, le *modèle exemplaire* [115], correspond au cas où toutes les duplications ont eu lieu après la spéciation, avec la conséquence que l'ancêtre commun n'a contenu qu'un seul exemplaire de chaque gène. Dans ce cas, on cherche à identifier parmi les multiples copies d'un gène dans un génome donné, laquelle provient directement de l'ancêtre commun et lesquelles ont été obtenues, par la suite, par duplications. Évidemment, tous les scénarios intermédiaires sont possibles entre ces deux scénarios extrêmes, où certains gènes apparaissaient en plusieurs copies chez l'ancêtre commun et d'autres non. Cette approche correspond au *modèle intermédiaire* [8].

Pour chacun de ces modèles, les problèmes récurrents consistent à déterminer

une mise en correspondance des copies optimisant une distance donnée. Malheureusement, quelque soit le modèle, les résultats de difficulté algorithmique sont omniprésents et très peu d'approximations à ratio constant existent (voir par exemple [24, 7]). Nous présentons, dans le Chapitre 3, plusieurs résultats d'inapproximabilité pour des problèmes sous le modèle exemplaire, afin de construire un panorama exhaustif des résultats de difficulté pour ce modèle. Plus précisément, nous prouvons que même lorsque l'entrée est contrainte à 1) un premier génome sans duplications et 2) un second génome où les gènes ont au plus deux occurrences, il est NP-difficile d'approcher les distances de Hamming, de Levenshtein, d'inversions et de DCJ en-dessous de ratios donnés.

Dans le Chapitre 4, nous considérons le problème Plus petite partition commune (MCSP, pour Minimum Common String Partition). Vu comme un problème sur les chaînes de caractères, l'objectif est de découper deux chaînes en un ensemble minimal de facteurs communs. Il revient à calculer la distance de cassure sous le modèle complet, avec la contrainte que chaque gène doit avoir le même nombre de copies dans les deux génomes. Malgré son apparente simplicité (il s'agit "seulement" de retrouver de longs facteurs communs entre deux séquences), ce problème est APX-difficile et aucun algorithme d'approximation à ratio constant n'est connu. Nous considérons ce problème à la lumière de la théorie de la complexité paramétrée, où le paramètre est la distance de cassure entre les deux chaînes après la mise en correspondance. Par rapport aux algorithmes FPT précédents [61, 91], qui nécessitaient à la fois la distance de cassure et le nombre de duplications comme paramètres, nous prouvons que la complexité algorithmique peut être confinée au nombre de facteurs dans le découpage, même pour des chaînes particulièrement répétitives.

Partie III – Traiter des données génomiques imprécises

Dans la troisième partie, nous considérons des modèles tenant compte d'une information imparfaite sur les séquences. En effet, chaque étape de construction des séquences d'entrée, telle que le séquencement ADN, l'annotation ou la reconstruction, peut produire des incertitudes ou même générer des erreurs dans les séquences. De façon générale, toute information génomique devrait ainsi être considérée avec la possibilité d'erreurs et d'incomplétude. Cependant, bien que l'on souhaiterait être capable de traiter de telles données en résolvant n'importe quelle question de génomique comparative, très peu de problèmes autorisent des modèles suffisamment généraux. Heureusement, il est parfois possible d'inclure des phases de "nettoyage" en tant que prétraitement, par exemple en supprimant des marqueurs probablement erronés, avant d'exécuter d'autres algorithmes requérant des données correctes et complètes.

Retirer des erreurs probables correspond exactement à l'objectif du problème d'Extraction de bandes maximales (MSR, pour Maximal Strip Recovery). Là, nous considérons les génomes de deux espèces aussi semblables que possibles. Ainsi, il est possible de supposer que la quasi-totalité des gènes appartiennent à des blocs de synténie plus grands et donc, il devrait être possible, dans un génome "nettoyé", d'identifier des blocs de synténie couvrant tout le génome (où chaque bloc contient au moins deux gènes). L'objectif du problème MSR est de marquer un nombre minimum de gènes comme étant des erreurs jusqu'à ce que les génomes restants aient cette propriété d'être "nettoyés". Il existe plusieurs variantes de ce problème : on peut tout d'abord considérer plus de deux génomes en entrée

afin de croiser plus d'informations (MSR-d, avec $d \geq 3$) ; on peut également autoriser les génomes avec des duplications (MSR-DU). Nous proposons aussi la variante de gap δ (δ-gap-MSR, avec $\delta \geq 3$) : deux éléments consécutifs d'un bloc synténique de sortie ne devraient pas être séparés (dans les génomes d'entrée) par plus de δ gènes marqués comme étant des erreurs. Enfin, dans le but de créer des approximations ou des algorithmes paramétrés, le problème peut être vu comme celui consistant à garder un nombre maximum ℓ de gènes (MSR), ou bien comme celui consistant à retirer un nombre minimum k de gènes (CMSR). Dans le Chapitre 5, nous étudions en profondeur les variantes de MSR et de CMSR, en proposant des résultats de NP et d'APX-difficulté, des algorithmes d'approximation et des algorithmes paramétrés. Les principaux résultats de ce chapitre sont la NP-difficulté de 1-gap-MSR et, pour CMSR-d et δ-gap-CMSR-d, un algorithme de $(d + 1.5)$-approximation et un algorithmes FPT de complexité $O^*(2.36^k)$.

Dans le Chapitre 6, nous étudions le problème de Linéarisation avec distance de cassure minimale (MBL, pour Minimum Breakpoint Linearization) qui a pour but de déterminer une séquence de gènes dans un chromosome dont l'ordre des gènes n'est que partiellement connu, en utilisant en référence le génome d'une espèce proche. Formellement, étant donnés un ordre partiel et un ordre total sur le même ensemble d'éléments (i.e., de gènes), il s'agit de trouver une linéarisation de l'ordre partiel minimisant la distance de cassure avec l'ordre total, où une linéarisation est un ordre total (ou permutation) compatible avec l'ordre partiel. Dans notre approche, nous proposons la création d'une structure de graphe (le graphe ordre-adjacences) qui permet de réduire MBL à un problème plus général de couverture de cycles déjà étudié en théorie des graphes (Subset Feedback Vertex Set). Les résultats connus de la littérature pour ce dernier nous permettent dans un premier temps d'obtenir deux algorithmes d'approximation et un algorithme paramétré pour MBL (les approximations ayant des ratios non constants et incomparables). Enfin, en connaissant les méthodes de construction des ordres partiels donnés en entrée, nous créons un troisième algorithme d'approximation spécifique à notre problème.

Définitions préliminaires

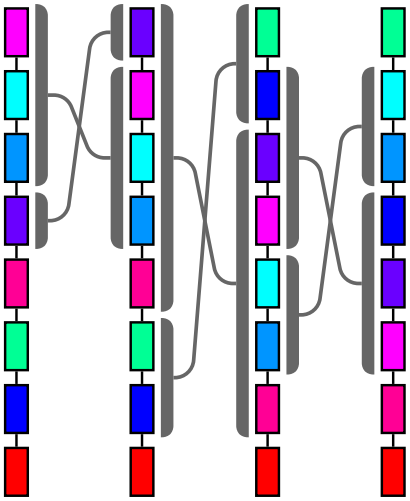
Nous présentons ici quelques définitions générales relatives aux séquences. Elles seront utilisées tout au long du manuscrit.

Pour la majeure partie des problèmes, un génome d'entrée est représenté par une *séquence* d'entiers, ou de *marqueurs*, notée $S = \langle s_1, s_2, \ldots, s_n \rangle$. Le $i$ème marqueur de S est noté $S[i]$ (ici, $S[i] = s_i$). Étant donnés deux entiers a et b, l'*intervalle* $[\![a\,;\,b]\!]$ est l'ensemble $\{a, a+1, \ldots, b\}$ si $a \leq b$ et l'ensemble vide si $b < a$. Une *permutation* de taille n est une séquence de n éléments distincts sur $[\![1\,;\,n]\!]$, et l'*Identité* de taille n, notée $\mathcal{I}_n$, est la permutation $\langle 1, 2, \ldots, n \rangle$. Par opposition aux permutations, une *chaîne* est une séquence qui peut contenir des éléments dupliqués.

Une *sous-séquence* S' de S est une séquence $\langle S[i_1], \ldots, S[i_h] \rangle$ de marqueurs de S provenant des positions $i_1 < i_2 < \ldots < i_h$. Un *facteur* de S est une sous-séquence d'éléments consécutifs ($i_{k+1} = i_k + 1$ pour $1 \leq k < h$). Un facteur de longueur 2 est appelé *adjacence*. Dans la comparaison de deux séquences génomiques, une adjacence d'une séquence qui n'est pas une adjacence de l'autre est appelée *point de cassure*, et la *distance de cassure* est le nombre de tels points de cassure.

Partie I – Chapitre 1

Tri par transpositions



L'objectif du problème de tri par transpositions est de calculer le nombre minimum de transpositions requises pour transformer un génome (représenté par une permutation) en un autre, où une transposition est une opération qui échange deux séquences consécutives dans la permutation.

Dans ce chapitre, nous prouvons que ce problème, ainsi qu'un problème de vérification de borne inférieure, est NP-difficile.

1.1 Présentation du problème

Avec les inversions, les transpositions sont une des opérations élémentaires les plus courantes qui peuvent affecter un génome à grande échelle. Une transposition consiste à échanger deux séquences consécutives de gènes ou, de façon équivalente, à déplacer une séquence de gènes d'un point à un autre dans le génome. Ainsi, la distance de transposition entre deux génomes est définie comme étant le nombre de transpositions nécessaires pour transformer l'un en l'autre. Calculer cette distance est un défi important de la génomique comparative, voir [110], puisqu'elle fournit un scénario d'évolution de parcimonie maximale entre les deux génomes étudiés.

Le problème de Tri par Transpositions est le problème qui consiste à calculer cette distance entre deux génomes représentés par des permutations.

Les résultats de ce chapitre ont été présentés au colloque Seqbio 2011 (Algorithmique, combinatoire du texte et applications en bio-informatique, Rennes), à ICALP 2011 (38th International Colloquium on Automata, Languages and Programming, Zürich [33]), et ont été publiés dans SIDMA en 2012 (SIAM Journal of Discrete Mathematics [35]).

1.2 État de l'art

Depuis sa présentation par Bafna et Pevzner [12, 14], la complexité algorithmique de ce problème n'a jamais été déterminée. Ainsi plusieurs études [14, 55, 79, 85, 65, 16, 68] visent à créer des algorithmes d'approximation ou des heuristiques, l'algorithme le plus précis étant, à ce jour, une 1.375-approximation [65]. D'autres travaux [80, 55, 66, 98, 65, 16] visent à calculer des bornes sur la distance de transposition d'une permutation. Des études ont également été menées sur des variantes de ce problème en considérant, par exemple, le tri par transpositions préfixes [62, 99, 51] (dans lequel chaque transposition doit faire intervenir un préfixe de la séquence) ou le tri par transpositions dans des chaînes [56, 60, 118, 113]. Il est également possible de considérer des transpositions pondérées ou préfixe dans les chaînes [111, 27, 6, 51, 5]. Enfin, il est important de remarquer que le tri par échange de blocs (i.e., par échanges de facteurs qui ne sont pas nécessairement consécutifs) peut être résolu en temps polynomial [54]. Nous référons le lecteur à [69, 3.1 - 3.2] pour un état de l'art détaillé sur ce problème et ses variantes.

1.3 Notre contribution

Dans ce chapitre, nous apportons une réponse à la question de la complexité algorithmique du Tri par transpositions en décrivant une réduction polynomiale depuis le problème SAT, et ainsi nous prouvons que ce problème est NP-difficile. Notre réduction est fondée sur l'étude des transpositions qui retirent trois points de cassure. Un corollaire de notre réduction est la NP-difficulté du problème suivant, proposé dans [55] : étant donné une permutation π, est-il possible de trier π en utilisant exactement $d_b(\pi)/3$ transpositions, où $d_b(\pi)$ est la distance de cassure entre π et l'identité ?

1.4 Méthode

Nous présentons ici les grandes lignes de la réduction dont découle la NP-difficulté du problème de Tri par Transpositions. Dans le but de rester syntétique, elle est exposée très informellement.

La réduction se fait depuis le problème de Satisfiabilité (SAT) en plusieurs étapes, en utilisant deux structures intermédiaires : les *3DT-instances* et les *assemblages de blocs de base*.

3DT-instances

Comme il a été suggéré précédemment, nous nous focalisons sur le problème consistant à décider si oui ou non une permutation peut être triée en $d_b(\pi)/3$ transpositions, où $d_b(\pi)$ est la distance de cassure entre π et l'identité. Il existe une condition nécessaire pour qu'une transposition ait une distance de transposition égale à $d_b(\pi)/3$: ses points de cassure doivent pouvoir être regroupés en triplets $(\langle a, a'\rangle, \langle b, b'\rangle, \langle c, c'\rangle)$ avec $b' = a + 1$, $c' = b + 1$ et $a' = c + 1$. On dit alors que la permutation est une 2-permutation. Ainsi, si ces trois cassures sont placées dans cet ordre (ou toute permutation circulaire de cet ordre), il existe une transposition

retirant les trois cassures en une seule étape (en coupant la séquence précisément à ces trois points). Dans le cas contraire, par exemple si elles apparaissent dans l'ordre $\langle a, a' \rangle, \langle c, c' \rangle, \langle b, b' \rangle$, il n'existe pas de telle transposition.

Cette notion de triplets est formalisée par la notion de 3DT-instance : une *3DT-instance* est un ensemble de triplets d'*éléments* où à chaque élément est associée une position unique. Un triplet (a, b, c) est bien ordonné si les positions de a, b et c correspondent à un triplet de points de cassures pouvant être retirés en une transposition (i.e., a est avant b qui est avant c, ou b est avant c qui est avant a, ou encore c est avant a qui est avant b). Parallèlement à une transposition retirant trois cassures, une *3DT-étape* est l'opération consistant à retirer un triplet (a, b, c) bien ordonné et, pour chaque position d'un élément restant, lui faire subir la transposition correspondant aux positions de a, b et c. Une 3DT-instance est *3DT-réductible* s'il existe une série de 3DT-étapes la réduisant à la 3DT-instance triviale (i.e. l'ensemble vide).

Le parallèle entre transpositions et 3DT-étapes permet de créer une relation d'équivalence entre les 2-permutations et (certaines) 3DT-instances. Nous prouvons le résultat suivant (cf. Théorème 1.9 page 20) : une permutation π peut être triée en $d_b(\pi)/3$ transpositions si et seulement la 3DT-instance équivalente est 3DT-réductible.

Dans la suite de la réduction, nous construisons une structure permettant de prouver qu'il est NP-difficile de déterminer si une 3DT-instance est 3DT-réductible. Nous prouverons dans un second temps que les structures ainsi construites sont toutes équivalentes à des permutations.

Blocs de base

Afin de simuler le comportement d'une formule booléenne (puisque la réduction se fait depuis SAT), nous construisons une 3DT-instance comme étant une concaténation (un *assemblage*) de *blocs de base*, chaque bloc ayant pour fonction la simulation d'un élément logique (variable booléenne, conjonction, etc.). Afin de transférer l'information entre les différents blocs, nous définissons des *variables*.

Variables. Une variable est une paire de triplets de la 3DT-instance notés (a, b, c) et (x, y, z). Ils sont répartis entre deux blocs : x, y et b dans le premier (le bloc *source*), a, z et c dans cet ordre dans le second (le bloc *cible*). De plus, ils ne sont a priori pas bien ordonnés. La variable est *activée* si $\langle x, b, y \rangle$ devient un facteur du bloc source : dans ce cas (x, y, z) devient bien ordonné, la transposition correspondante déplace b du bloc source vers le bloc cible (à la place de z). Le triplet (a, b, c) devient alors bien ordonné, il peut alors générer une 3DT-étape qui aura des conséquences sur le bloc cible. Le but d'une variable est de simuler une expression booléenne : elle ne doit pouvoir être activée que si cette expression est vraie.

Blocs de base. Nous définissons quatre types de blocs de base. Dans cette synthèse, nous ne donnons pas la définition de chaque bloc sous forme de triplets, mais sa caractérisation par le *graphe de comportement*, qui décrit sous quelles conditions les variables dont ce bloc est la cible (les *variables d'entrée*) ou la source (les *variables de sortie*) peuvent être activées.

Le bloc copy a pour simple but de dédoubler une variable (il est nécessaire puisque chaque variable ne peut avoir qu'une seule cible). Il dispose d'une variable d'entrée

qui sert de déclencheur pour deux variables de sortie : si la variable d'entrée est activée, alors chacune des deux variables de sortie peut être activée et ce, dans n'importe quel ordre. Voir Figure 1a.

Le bloc and permet de simuler une conjonction. Il a deux variables d'entrée et une de sortie : il est nécessaire que les deux variables d'entrée aient été activées pour pouvoir activer la variable de sortie. Voir Figure 1b.

De la même façon, le bloc or permet de simuler une disjonction. Il a deux variables d'entrée et une de sortie. Il faut et il suffit qu'au moins une des deux variables d'entrée soit activée pour pouvoir activer la variable de sortie. Voir Figure 1c.

Enfin, le bloc var permet de simuler une variable booléenne x : il dispose de deux variables de sortie représentant $+x$ et $-x$. Sans condition sur l'entrée activée, il est possible d'activer au plus une des deux variables, ce qui correspond au fait que x est considérée soit comme étant vraie, soit comme étant fausse. Ce bloc dispose également d'une variable d'entrée dont le rôle est décrit ci-dessous.

Double passage. La construction telle qu'elle a été décrite jusqu'à présent permet de réaliser un assemblage de blocs de façon à ce qu'une variable "objectif" (notée A_ϕ) corresponde à l'expression booléenne ϕ considérée : A_ϕ peut être activée si et seulement si ϕ est satisfiable. En revanche, il faut que l'assemblage soit totalement 3DT-réductible dans le cas où ϕ est satisfiable. Il est donc nécessaire que l'activation de A_ϕ permette de déclencher des 3DT-étapes pour tous les triplets restants. Cela revient à activer, dans un second passage, toutes les variables qui n'avaient pas été activées lors du premier passage. Pour ce faire, il suffit d'activer toutes les variables correspondant aux littéraux $+x$ ou $-x$ qui avaient été considérés comme faux dans un premier temps, les autres variables pourront être activées en cascade.

La variable A_ϕ, ou plutôt, des copies de la variable A_ϕ, sont donc données en entrée au bloc var : une fois cette entrée, il est possible d'activer la deuxième variable de sortie (qui n'avait pas été activée lors du premier passage). Le graphe de comportement du bloc var final est décrit Figure 1d.

Assemblage des blocs de base et réduction.

Étant donnée une formule ϕ, on crée une 3DT-instance I_ϕ comme un assemblage utilisant un bloc var pour chaque variable booléenne, un bloc and ou or pour chaque conjonction ou disjonction, des blocs copy pour les littéraux ayant plusieurs occurrences et pour la variable objectif A_ϕ. La définition détaillée de I_ϕ n'est pas donnée ici, mais un exemple illustratif est donné ci-dessous.

Exemple 1. *Soit la formule booléenne ϕ suivante.*

$$\begin{aligned}\phi =&(x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee \neg x_4) \wedge \\ &(\neg x_1 \vee x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \wedge (\neg x_2 \vee \neg x_3 \vee x_4)\end{aligned}$$

Note : la formule ϕ est satisfiable avec $x_1 = x_3 =$ vrai et $x_2 = x_3 =$ faux.

La Figure 2 décrit les blocs utilisés dans l'assemblage de I_ϕ et les interactions entre eux (quelles variables de sortie correspondent à quelles variables d'entrée). L'ordre des blocs dans I_ϕ n'a pas d'importance.

Grâce à la construction précédente, on peut créer, à partir de n'importe quelle formule ϕ, une 3DT-instance qui est 3DT-réductible si et seulement si ϕ est satisfiable

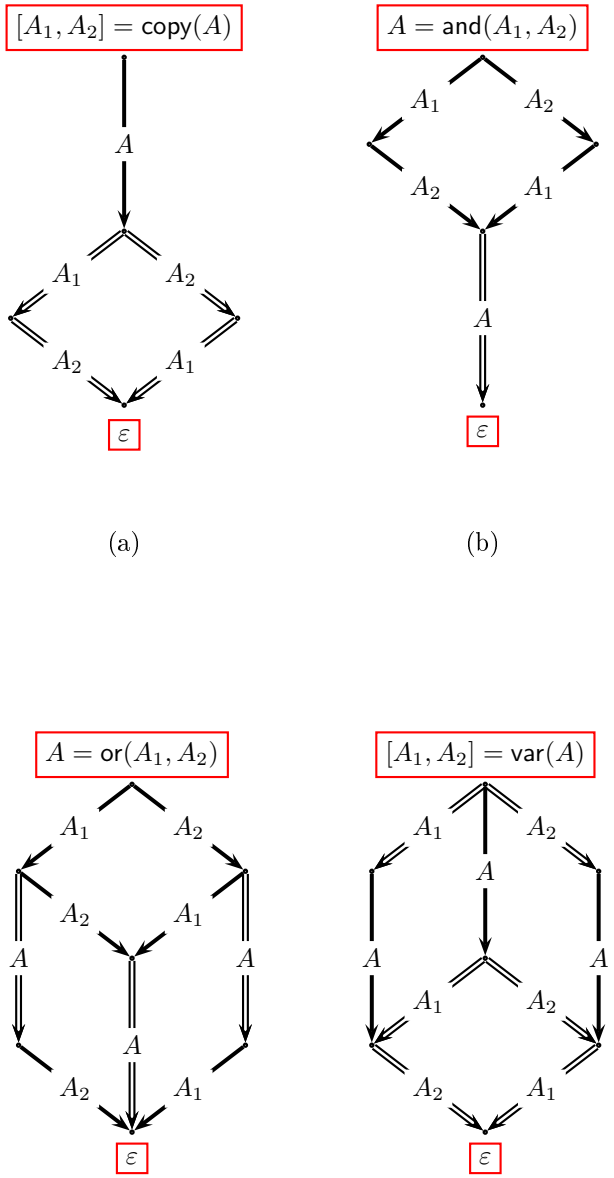


FIGURE 1 – Représentation simplifiée des graphes de comportement des blocs copy, and, or, et var. Un arc épais correspond à l'activation d'une variable d'entrée, un arc double à une variable de sortie. Le bloc évolue à chaque activation de variable, en commençant par sa définition originale et terminant au bloc vide (noté ϵ) dans lequel il ne reste aucun triplet.

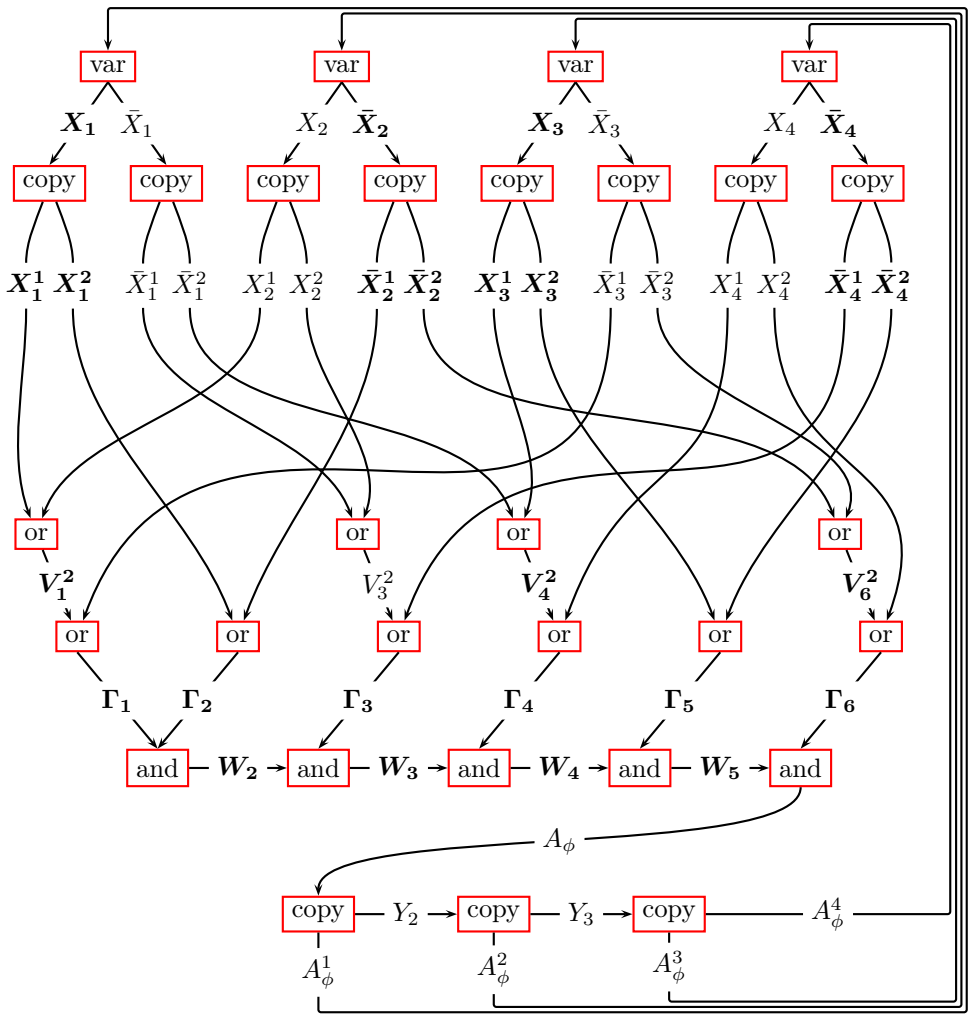


FIGURE 2 – Diagramme illustrant la construction de I_ϕ pour la formule ϕ définie dans l'Exemple 1. Un arc entre deux blocs représente une variable ayant pour source et cible les blocs respectifs. Les variables pouvant être activées lors du premier passage (celles correspondant à des termes vrais dans l'affectation) sont représentées en gras.

(Théorème 1.19 page 38). Cela nous permet de conclure que le problème consistant à déterminer si une 3DT-instance est 3DT-réductible est donc NP-difficile. Malheureusement, il n'existe pas de garantie a priori que les 3DT-instances ainsi construites possèdent toutes une permutation équivalente.

Retour aux permutations Afin de compléter la démonstration, nous prouvons dans la dernière partie (Théorème 1.20 page 41) que toute 3DT-instance qui est construite comme un assemblage de blocs de base a effectivement une permutation équivalente.

Ce résultat permet donc de conclure la réduction, puisque pour toute formule booléenne ϕ nous pouvons construire une permutation π_ϕ dont la distance de transposition est $d_b(\pi_\phi)/3$ si et seulement si ϕ est satisfiable : le problème de TRI PAR TRANSPOSITIONS est NP-difficile (Théorème 1.21 page 45).

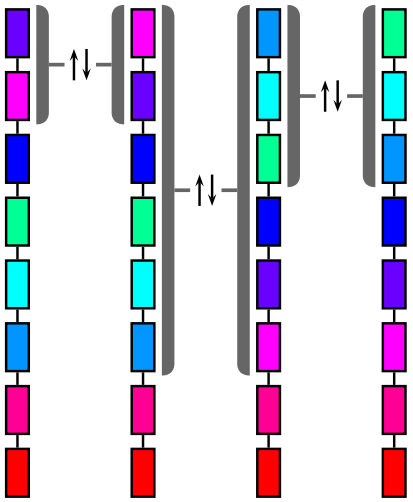
1.5 Conclusion

Même en connaissant la classe de complexité du problème de TRI PAR TRANSPOSITIONS, plusieurs questions restent ouvertes. Par exemple, ce problème admet-il un schéma d'approximation en temps polynomial ? Notre réduction ne permet pas de répondre à cette question puisqu'elle n'est pas linéaire : si la formule ϕ n'est pas satisfiable, il peut être remarqué que $d_t(\pi_{I_\phi}) = d_b(\pi_{I_\phi})/3 + 1$.

D'autre part, existe-t-il des paramètres pertinents pour lesquels le problème est FPT ? Un paramètre éventuel serait la taille des facteurs échangés dans chaque transposition. Le problème devient-il polynomial si ce paramètre est borné ? En fait la réponse est non si le paramètre est la taille du facteur le plus petit de chaque transposition, puisque dans notre réduction, cette quantité est majorée par 6 pour toutes les transpositions nécessaires au tri de π_{I_ϕ}, indépendamment de la formule ϕ.

Partie I – Chapitre 2

Tri par inversions préfixes



Le Tri par inversions préfixes (SBPR, pour Sorting by Prefix Reversals) est un problème plus connu sous le nom de *retournement de crêpes* : il s'agit de réarranger une pile de crêpes de tailles différentes (i.e., une permutation) pour en faire une pile pyramidale, où la seule action possible est d'insérer une spatule dans la pile et de retourner les crêpes se trouvant au-dessus (i.e., d'effectuer une inversion préfixe). Calculer un scénario optimal est un problème combinatoire vieux de plus de trente ans.

Dans ce chapitre, nous prouvons que le problème de retournement de crêpes est NP-difficile.

2.1 Présentation du problème

Le problème de retournement de crêpes a été tout d'abord présenté dans [64] sous la forme de l'énigme suivante.

> Imaginez un serveur devant apporter une pile de crêpes à des clients. Malheureusement, le chef n'est pas particulièrement minutieux, et les crêpes ont toutes des tailles différentes. En chemin vers la table, le serveur réarrange la pile (de façon à mettre la plus petite en haut, etc., jusqu'à la plus grande tout en bas) en saisissant, autant de fois que nécessaire, plusieurs crêpes du haut de la pile et en les retournant. S'il y a n crêpes, combien, au maximum, le serveur devra-t-il effectuer de retournements (en fonction de n) pour réarranger la pile ?

Les résultats de ce chapitre ont fait l'objet d'une présentation invitée au colloque A&P 2012 (Algorithms and Permutations, Paris), et ont également été présentés à MFCS 2012 (37th International Symposium on Mathematical Foundations of Computer Science, Bratislava [34]). Ils sont actuellement soumis pour publication à JCSS (Journal of Computer and System Sciences, [37]).

Puisque toutes les crêpes ont des tailles différentes, une pile peut être représentée par une permutation (1 est la plus petite, 2 la suivante, etc., jusqu'à n la plus grande), et un retournement consiste à effectuer une inversion d'un préfixe de taille arbitraire. L'énigme présentée ci-dessus engendre deux problèmes liés :

- Réaliser un algorithme qui trie n'importe quelle permutation avec un nombre de retournements minimal (ce problème d'optimisation est abrévié en SBPR, pour Sorting By Prefix Reversals). Voir Figure 3.
- Calculer $f(n)$, le nombre maximum de retournements requis pour trier une permutation de taille n (ce nombre correspond au diamètre d'un graphe nommé *pancake network*).

Gates et Papadimitriou [76] ont proposé la variante du problème où les crêpes sont *brûlées* : chaque crêpe a deux faces (l'une brûlée, l'autre nom), et une contrainte est ajoutée, imposant que toutes les crêpes aient la face non brûlée au-dessus à la fin du tri. La borne sur le nombre de retournements nécessaires dans le cas brûlé est notée $g(n)$.

2.2 État de l'art

De nombreuses études [50, 57, 59, 76, 88, 87, 100] ont pour but d'encadrer le plus précisément possible les valeurs de $f(n)$ et $g(n)$, avec les résultats suivants :

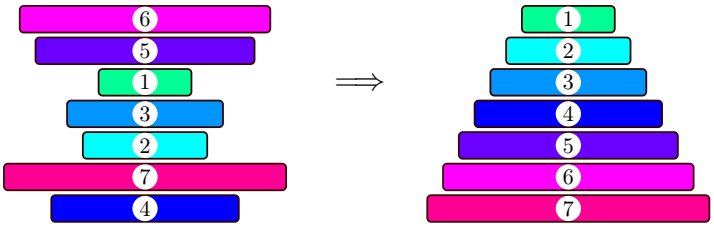
- $f(n)$ et $g(n)$ sont connus exactement pour $n \leq 19$ et $n \leq 17$, respectivement [57].
- $15n/14 \leq f(n) \leq 18n/11 + O(1)$ [87, 50].
- $\lfloor (3n+3)/2 \rfloor \leq g(n) \leq 2n-6$ [57] (la borne supérieure est valide pour $n \geq 16$).

En ce qui concerne le problème SBPR, des algorithmes de 2-approximation ont été créés à la fois pour les variantes brûlées et non brûlées [59, 70]. De plus, Labarre et Cibulka [100] ont caractérisé une sous-classe de permutations (les *permutations simples*) pour lesquelles un tri optimal peut être calculé en temps polynomial.

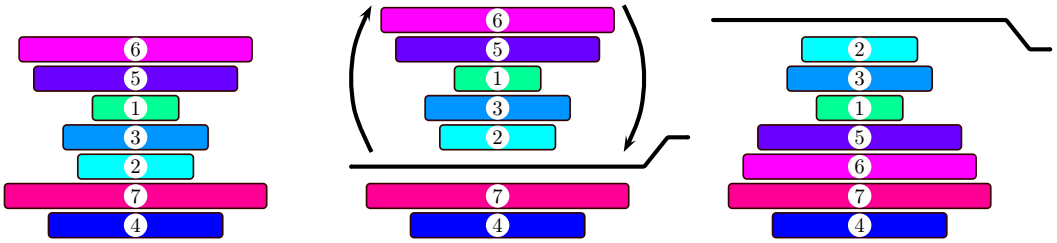
Le problème de retournement de crêpes a plusieurs applications. Par exemple, le graphe *pancake network* a à la fois un faible degré et un faible diamètre par rapport au nombre de ses sommets. Il est ainsi intéressant pour des applications en calcul parallèle [1, 112]. L'aspect algorithmique, i.e. le problème de tri, correspond exactement à la distance de réarrangement de génomes où les opérations autorisées sont les inversions préfixes (les variantes brûlées/non brûlées correspondent respectivement aux modélisations des génomes en permutations signées/non signées). Par conséquent, ce problème a particulièrement éveillé l'intérêt de la communauté de génomique comparative, notamment à cause de ses similarités avec le problème de tri par retournements (ou Sorting By Reversals) [11], qui admet un algorithme exact polynomial [83] pour la variante signée, et une 1.375-approximation [19] pour la variante non signée (qui est APX-difficile [20]).

2.3 Notre contribution

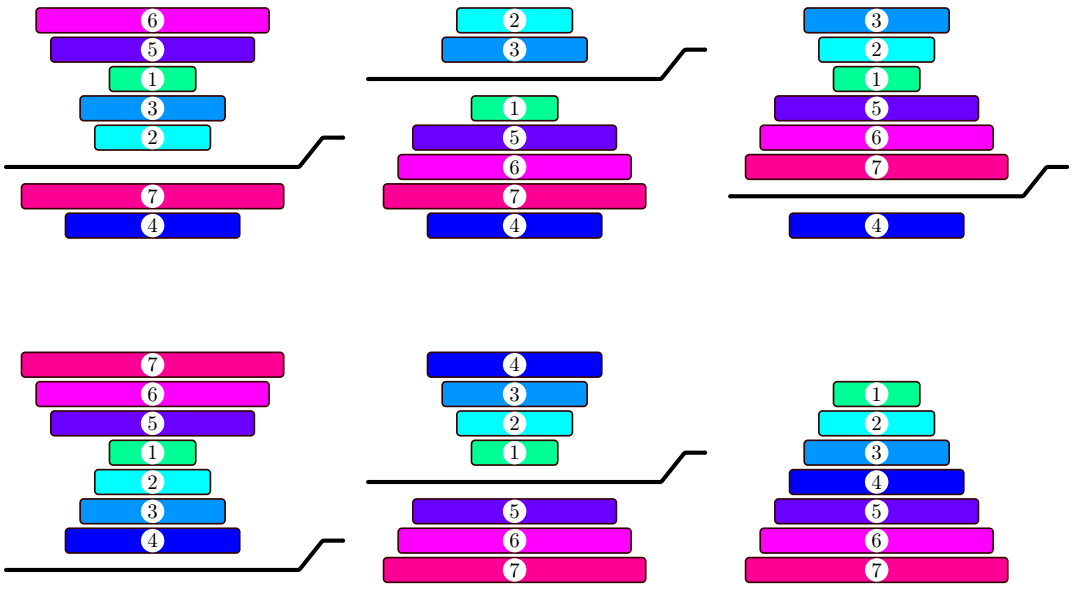
Dans ce chapitre, nous prouvons que le problème SBPR est NP-difficile (dans sa variante non brûlée, ou non signée), ce qui permet de clore une question restée ouverte pendant plusieurs décennies. Nous prouvons en fait un résultat plus fort : le nombre de cassures est connu pour être une borne inférieure sur la distance par inversions préfixes. Nous prouvons, comme pour le problème de tri par transpo-



(a) Objectif : transformer une permutation (e.g., $\langle 6, 5, 1, 3, 2, 7, 4 \rangle$) en l'identité $\mathcal{I}_7^1$.



(b) Un *retournement* de 5 éléments : $\langle 6, 5, 1, 3, 2, 7, 4 \rangle$ devient $\langle 2, 3, 1, 5, 6, 7, 4 \rangle$. La limite du retournement est marquée avec un symbole de spatule.



(c) La séquence de 5 retournements nécessaires pour trier $\langle 6, 5, 1, 3, 2, 7, 4 \rangle$. Elle est optimale car le nombre de cassures de la permutation est 5. Elle est aussi, de fait, la seule séquence de retournements optimale.

FIGURE 3 – Illustration du problème de TRI PAR INVERSIONS PRÉFIXES.

SITIONS, que le problème consistant à déterminer si cette borne est atteinte pour une permutation donnée est lui-même NP-difficile.

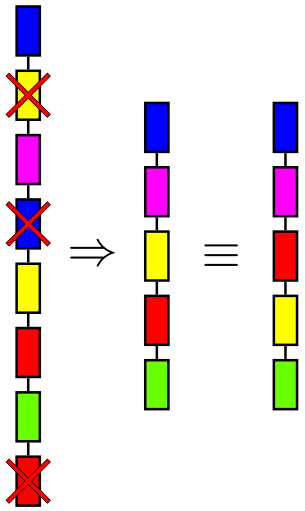
Les méthodes employées pour les deux preuves ont de nombreux points communs, avec une réduction à un problème de satisfiabilité. Nous construisons des séquences par concaténation de gadgets, telles qu'il existe un chemin "efficace" (i.e. qui retire un maximum de points de cassure à chaque opération) si et seulement si une formule est satisfiable. La contrainte de "double passage" est également présente. En revanche, la spécificité des inversions préfixes, notamment le fait qu'il n'existe à tout moment que deux points de cassure au maximum qui peuvent être retirés par une opération efficace, font que les gadgets définis sont très différents, et que l'ordre de passage dans les gadgets est complètement contraint.

2.4 Conclusion

Comme pour le problème de TRI PAR TRANSPOSITIONS, la NP-difficulté du problème de TRI PAR RETOURNEMENTS PRÉFIXES représente un résultat important dans les problèmes de réarrangement de génomes. La question de l'APX-difficulté reste cependant ouverte, tout comme la complexité de la variante signée (ou brûlée), ou la question de l'existence d'algorithmes paramétrés pour ce problème.

Partie II – Chapitre 3

Distances exemplaires



Le problème de Distance Exemplaire demande, étant donnés deux génomes présentant des duplications, d'extraire des *exemplarisations* de ces génomes (i.e. des sous-séquences contenant exactement une copie de chaque gène) minimisant une mesure de dissimilarité choisie.

Dans ce chapitre, nous prouvons qu'il est NP-difficile d'approximer le problème de Distance Exemplaire pour deux distances de réarrangement (tri par inversions et DCJ) et pour les distances d'édition de Hamming et de Levenshtein.

3.1 Présentation du problème

Dans ce chapitre, un *génome* est représenté par une séquence signée d'entiers : chaque entier représente un gène (ou plus généralement, un *marqueur*) d'une certaine famille avec une certaine orientation. Étant donnés deux génomes, potentiellement avec des doublons, le problème de Distance Exemplaire [115] consiste à retirer toutes les copies sauf une de chaque marqueur dans chaque génome, tout en minimisant une distance objectif entre les deux génomes. On appelle les génomes réduits des *sous-séquences exemplaires* des génomes d'origine. Cette approche revient à considérer que les duplications sont récentes dans l'histoire évolutive des deux espèces considérées (elles ont eu lieu après la spéciation), ou alors que les données permettent de distinguer les copies provenant d'une duplication plus ancienne. Ainsi, un seul exemplaire de chaque famille de gènes provient de l'ancêtre commun

Les résultats de ce chapitre ont été obtenus en collaboration avec Mighui Jiang. Ils ont été présentés à ISBRA 2012 (*8th International Symposium on Bioinformatics Research and Applications*, Dallas [39]), et sont à paraître dans TCBB (*IEEE Transactions on Computational Biology and Bioinformatics* [40]).

et doit être mis en correspondance avec un exemplaire de l'autre espèce (un gène *orthologue*).

Par exemple, les deux génomes ci-dessous

$$G_1 : \quad \langle -4, +1, +2, +3, -5, +1, +2, +3, -6 \rangle$$
$$G_2 : \quad \langle -1, -4, +1, +2, -5, +3, -2, -6, +3 \rangle$$

peuvent tous deux être réduits à la même séquence

$$G' : \quad \langle -4, +1, +2, -5, +3, -6 \rangle$$

en retirant des doublons, ils ont donc une distance exemplaire de zéro pour n'importe quelle mesure de distance raisonnable. De façon générale, le problème de distance exemplaire est en fait une famille de problèmes dépendants du choix de la mesure de distance (qui n'est pas unique). Nous renvoyons à la Figure 4 pour un exemple de scénario où la distance sous-jacente est la distance d'inversions signée.

Nous notons (s, t)-Distance Exemplaire le problème de calcul de distance exemplaire sur deux génomes G_1 et G_2 où chaque gène apparaît au plus s fois dans G_1 et au plus t fois dans G_2. D'après [26, 96], quelle que soit la distance utilisée (à condition que ce soit une distance, i.e. elle doit valoir zéro pour deux séquences identiques), le problème $(2, 2)$-Exemplar Distance n'admet pas d'approximation. En effet, le simple fait de décider si oui ou non les deux séquences peuvent être réduites à la même sous-séquence exemplaire est en soi un problème NP-difficile. Dans ce chapitre, nous portons nos recherches sur la variante non triviale la plus simple du problème de distance exemplaire : $(1, 2)$-Distance Exemplaire. Remarque : nous utilisons le modèle monochromosomal (un génome est représenté par une seule séquence) afin d'avoir, encore une fois, la définition du problème la plus simple ; ainsi, les résultats de difficulté peuvent s'étendre au modèle multichromosomal.

3.2 État de l'art

Le problème $(1, t)$-Distance Exemplaire a été étudié pour plusieurs distances habituelles utilisées en génomique comparative. Angibaud et al. [7] ont prouvé que pour les distances de cassure, d'intervalles communs et d'intervalles conservés, le problème $(1, 2)$-Distance Exemplaire est APX-difficile. Bonizzoni et al. [28] ont démontré que des variantes du problème de Plus Longue Sous-Séquence Commune sous le modèle exemplaire est également APX-difficile. Pour les mesures de *perturbation d'adjacence maximale* (MAD, pour *Maximum Adjacency Disruption*) et *perturbation d'adjacence totale* (SAD, pour *Summed Adjacency Disruption*), introduites par Sankoff et Haque [116], calculer la distance exemplaire est APX-difficile [24]. Plus précisément, $(1, 2)$-Distance MAD Exemplaire est NP-difficile à approximer sous un facteur $2 - \epsilon$ pour tout $\epsilon > 0$, et $(1, 2)$-Distance SAD Exemplaire est NP-difficile à approximer sous un facteur 1.3606 [40]. Voir aussi [48, 46] pour des résultats liés.

3.3 Distance d'inversions

Étant donnée une permutation signée $\pi = \langle \pi_1, \ldots, \pi_n \rangle$, une *inversion non signée* (i, j) avec $1 \leq i \leq j \leq n$ transforme π en $\langle \pi_1, \ldots, \pi_{i-1}, \ \pi_j, \ldots, \pi_i, \ \pi_{j+1}, \ldots, \pi_n \rangle$, où

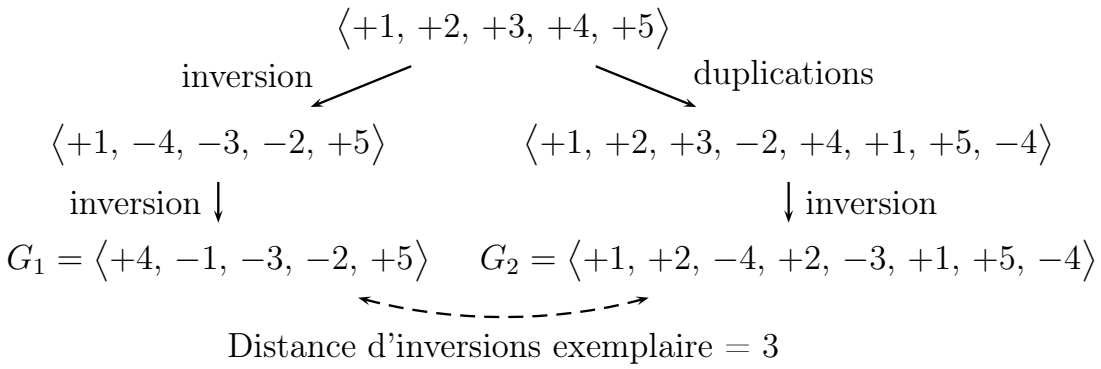


FIGURE 4 – Au cours de l'évolution de deux espèces depuis la spéciation au niveau de l'ancêtre commun, des duplications ont lieu dans G_2 et des inversions se produisent à la fois dans G_1 et G_2. Selon le principe de parcimonie, la distance exemplaire de 3 entre G_1 et G_2 correspond au nombre d'inversions dans le scénario le plus probable de l'histoire évolutive de ces deux espèces.

le facteur $\pi_i \dots \pi_j$ est *inversé*. Pour une permutation signée $\sigma = \langle \sigma_1, \dots, \sigma_n \rangle$, une *inversion signée* (i, j) avec $1 \leq i \leq j \leq n$ transforme σ en la permutation signée $\langle \sigma_1, \dots, \sigma_{i-1}, -\sigma_j, \dots, -\sigma_i, \sigma_{j+1}, \dots, \sigma_n \rangle$, où le facteur $\langle \sigma_i, \dots, \sigma_j \rangle$ est *retourné* (voir Figure 5a). La *distance d'inversions non signée* (resp. *distance d'inversions signée*) entre deux permutations non signées (resp. signées) est le nombre minimal d'inversions non signées (resp. signées) requises pour transformer une permutation en l'autre. Calculer la distance d'inversions non signée est APX-difficile [20], et la distance d'inversions signée est quant à elle calculable en temps polynomial [84].

Le théorème ci-dessous répond à la question posée par Blin et al. [24] sur l'approximabilité du problème de distance exemplaire pour les inversions signées :

Theorème 1. *Voir Théorème 3.1 page 77. Il est* NP*-difficile d'approximer la* $(1, 2)$-DISTANCE D'INVERSIONS SIGNÉE EXEMPLAIRE *sous un facteur* $1237/1236 - \epsilon$ *pour tout* $\epsilon > 0$.

Méthode

La preuve du Théorème 1 se fait depuis le problème de TRI PAR INVERSIONS NON SIGNÉ.

La réduction est basée sur une idée simple : la difficulté du tri par inversions non signé tient au fait qu'il est difficile de choisir "dans quelle sens" il faut lire chaque gène, c'est-à-dire si un élément non signé π_i devrait être lu "$+\pi_i$" ou "$-\pi_i$". Puisque le problème de distance exemplaire consiste justement à devoir choisir un élément parmi deux, il suffit de remplacer π_i dans l'instance de TRI PAR INVERSIONS NON SIGNÉ par $\langle +\pi_i, -\pi_i \rangle$ dans l'instance de $(1, 2)$-DISTANCE D'INVERSIONS SIGNÉE EXEMPLAIRE. Le théorème découle du fait que ces deux instances ont des solutions optimales identiques pour leurs problèmes respectifs, et de l'APX-difficulté du TRI PAR INVERSIONS NON SIGNÉ.

3.4 Distance de DCJ

L'opération de DCJ (pour *Double-Cut-And-Join*) proposée par Yancopoulos et al. [124] consiste à couper une permutation à deux positions, et à recoller les quatre

(a) $\langle +1, +2, +3, +4, +5, +6\rangle \longrightarrow \langle +1, +2, -5, -4, -3, +6\rangle$

(b) $\langle +1, +2, +3, +4, +5, +6\rangle \longrightarrow \langle +1, +2, +6\rangle\ (+3, +4, +5)$

(c) $\langle +1, +2, +3\rangle\ (+4, +5, +6) \longrightarrow \langle +1, +2, -5, -4, -6, +3\rangle$

FIGURE 5 – Les opérations autorisées pour le tri par DCJ signé sont (a) les inversions, (b) les excisions, et (c) les insertions. Une permutation circulaire est notée avec des parenthèses, c'est-à-dire que $(+1\ +2\ +3)$ est égale à $(+2\ +3\ +1)$ et à $(-3\ -2\ -1)$.

extrémités ainsi créées de n'importe quelle manière. Ainsi, une opération de DCJ peut correspondre soit à une inversion, soit à l'excision d'un facteur pour former une permutation circulaire, soit à l'insertion d'une permutation circulaire dans la séquence principale, à n'importe quelle position (voir Figure 5). Le problème consistant à calculer la distance de DCJ entre deux permutations est polynomial dans le cas signé [124], et NP-difficile dans le cas non signé [43]. Le théorème ci-dessous prouve l'inapproximabilité de la distance DCJ exemplaire :

Theorème 2. *Voir Théorème 3.2 page 78. Il est* NP*-difficile d'approximer la* $(1, 2)$-DISTANCE DCJ EXEMPLAIRE *sous un facteur* $1237/1236 - \epsilon$ *pour tout* $\epsilon > 0$.

Méthode

La preuve du Théorème 2 se fait depuis le problème de TRI PAR DCJ NON SIGNÉ, de la même façon que pour le Théorème 1.

En revanche, pour pouvoir conclure la démonstration nous étendons la preuve de NP-difficulté [43] en preuve d'APX-difficulté en prouvant le résultat suivant : il est NP-difficile d'approximer le problème de TRI PAR DCJ NON SIGNÉ sous un facteur $1237/1236 - \epsilon$ pour tout $\epsilon > 0$.

3.5 Distances d'édition

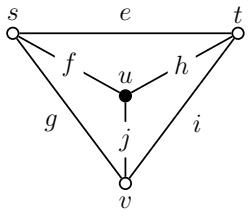
Dans le dernier théorème de ce chapitre, nous présentons le premier résultat d'inapproximabilité pour le problème de distance exemplaire en utilisant une distance d'édition classique :

Theorème 3. *Voir Théorème 3.3 page 78. Le problème* $(1, 2)$-DISTANCE D'ÉDITION EXEMPLAIRE *est* APX*-difficile lorsque le coût d'une substitution est* 1 *et que le coût d'une insertion ou d'une suppression est au moins* 1.

Les distances de Hamming et de Levenshtein sont chacune une distance d'édition particulière : pour la distance de Levenshtein, le coût de chaque opération (substitution, insertion, suppression) est 1 ; pour la distance de Hamming, le coût d'une substitution est 1 et le coût d'une insertion ou d'une suppression est $+\infty$. Nous avons donc les deux corollaires suivants.

Corollaire 4. *Le problème* $(1, 2)$-DISTANCE DE LEVENSHTEIN EXEMPLAIRE *est* APX*-difficile.*

Corollaire 5. *Le problème* $(1, 2)$-DISTANCE DE HAMMING EXEMPLAIRE *est* APX*-difficile.*



$G_1 = \langle\ e,\ \mathbf{S},\ f,\ \mathbf{S},\ g,\ \mathbf{S},\ h,\ \mathbf{S},\ i,\ \mathbf{S},\ j,\ -$
$G_2 = \langle\ e_s, \underline{e_t},\ \mathbf{S},\ f_s, \underline{f_u},\ \mathbf{S},\ g_s, \underline{g_v},\ \mathbf{S},\ h_t, \underline{h_u},\ \mathbf{S},\ \underline{i_t},\ i_v,\ \mathbf{S},\ \underline{j_u},\ j_v,\ -$

$-\ \mathbf{S},\quad s,\ s'_1,\ s'_2,\ s'_3,\quad \mathbf{S},\quad t,\ t'_1,\ t'_2,\ t'_3,\quad \mathbf{S},\quad \mathbf{u},\ u'_1,\ u'_2,\ u'_3,\quad \mathbf{S},\quad v,\ v'_1,\ v'_2,\ v'_3,\quad -$
$-\ \underline{\mathbf{S},\ e_s,\ f_s,\ g_s,\ s},\ e,\ f,\ g,\ \underline{\mathbf{S}},\ e_t, \underline{h_t},\ i_t, \underline{t,\ e},\ h, \underline{i,\ \mathbf{S}},\ f_u,\ h_u,\ j_u, \underline{\mathbf{u},\ f,\ h,\ j,\ \mathbf{S}},\ g_v, \underline{i_v,\ j_v,\ v,\ g},\ i,\ j,\ -$

$-\ \mathbf{S},\ e_s,\ e_t,\ f_s,\ f_u,\ g_s,\ g_v,\ h_t,\ h_u,\ i_t,\ i_v,\ j_u,\ j_v\ \rangle$
$-\ \underline{\mathbf{S},\ s'_1,\ s'_2,\ s'_3,\ t'_1,\ t'_2,\ t'_3,\ u'_1,\ u'_2,\ u'_3,\ v'_1,\ v'_2,\ v'_3}\ \rangle$

FIGURE 6 – Exemple pour la réduction du problème de couverture à $(1,2)$-DISTANCE D'ÉDITION EXEMPLAIRE. Haut : un graphe cubique G avec une couverture optimale $\{s,t,v\}$ et l'ensemble indépendant complémentaire $\{u\}$. Bas : Les séquences G_1 et G_2 créées à partir de G, où nous utilisons un symbole unique S pour tous les séparateurs. Une sous-séquence exemplaire optimale de G_2 est soulignée, et les éléments alignés pour le calcul de la distance d'édition sont en gras.

Méthode

La réduction se fait depuis le problème COUVERTURE DE SOMMETS MINIMALE DANS UN GRAPHE CUBIQUE, ses grandes étapes sont présentées ci-dessous.

Construction. Soit $G = (V, E)$ un graphe cubique de n sommets et m arêtes (ainsi $3n = 2m$). Nous construisons deux séquences G_1 et G_2. Pour chaque arête $e = \{u, v\} \in E$, nous définissons trois *marqueurs d'arête* e, e_u, et e_v. Pour chaque sommet $v \in V$, nous définissons un *marqueur de sommet* v et trois *marqueurs tampons* v'_1, v'_2, v'_3. Finalement, nous utilisons $2(m+7n)+2(m-1)+(n-1)$ marqueurs supplémentaires servant de séparateurs. La construction est illustrée Figure 6 pour le graphe complet K_4.

Nous définissons deux types de gadgets. Le premier, pour une arête $e = \{u, v\}$:

$$G_1[e] = \langle e \rangle,$$
$$G_2[e] = \langle e_u, e_v \rangle.$$

Et le second, pour un sommet v incident à trois arêtes e, f, g :

$$G_1[v] = \langle v,\ v'_1, v'_2, v'_3 \rangle$$
$$G_2[v] = \langle e_v, f_v, g_v,\ v,\ e,\ f,\ g \rangle$$

La séquence G_1 est obtenue comme étant la concaténation des $G_1[e]$, $e \in E$, séparés par des séparateurs courts (2 marqueurs), suivis d'un séparateur long ($m+7n$ marqueurs), suivi des $G_1[v]$, $v \in V$, séparés par des séparateurs courts (1 marqueur), suivis d'un séparateur long et de tous les marqueurs e_u, e_v pour $e \in E$. La séquence G_2 est construite de la même façon, avec une différence : les marqueurs finaux sont les v'_1, v'_2, v'_3 pour $v \in V$.

Équivalence La preuve du Théorème 3 découle de l'équivalence suivante (voir Lemme 3.8 page 81) : le graphe G a un ensemble de sommets couvrant de taille k si et seulement si G_2 a une sous-séquence exemplaire G'_2 ayant une distance d'édition au plus $m + 6n + k$ de G_1.

Nous détaillons ici l'implication directe : la sous-séquence exemplaire G'_2 est construite de la façon suivante à partir d'un ensemble couvrant X de taille k. Pour toute arête $e = \{u, v\}$ avec $u \in X$, e_u est conservé dans $G_2[u]$ et supprimé dans $G_2[e]$, et e_v est supprimé dans $G_2[v]$ et conservé dans $G_2[e]$. Puis, pour chaque sommet v, entre 0 et 4 marqueurs tampons sont supprimés dans $G_2[v]$ afin qu'il reste exactement 4 marqueurs dans cette sous-séquence. On constate alors que tous les séparateurs sont alignés entre G_1 et G'_2 et ils ont un coût de zéro pour la distance d'édition. Tous les autres marqueurs, à l'exception des marqueurs de sommet v pour $v \notin X$, ne sont pas alignés entre G_1 et G'_2, et ont un coût de une substitution pour la distance d'édition. Les marqueurs v pour $v \notin X$ sont eux alignés et ont un coût de zéro. La distance totale est donc $m + 6n + k$ (les marqueurs non alignés sont les m marqueurs e pour $e \in E$, les $6n$ marqueurs $e_u, f_u, g_u, u'_1, u'_2, u'_3$ pour $u \in V$, et k marqueurs u pour $u \in X$).

3.6 Conclusion

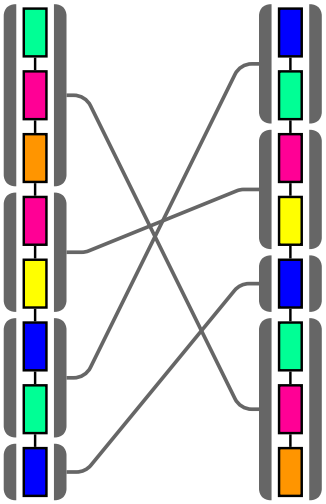
Dans ce chapitre, nous avons étudié la complexité du calcul de sous-séquences exemplaires optimisant plusieurs distances de similarités.

Notre choix de mesures de distances est basé sur deux considérations. D'une part, pour avoir les résultats les plus généraux possibles, nous avons choisi d'explorer une variété maximale de mesures de distances qui sont pertinentes pour des applications biologiques variées. Les distances d'éditions mesurent les différences locales, alors que les distances d'inversions et de DCJ calculent des scénarios de réarrangements globaux. D'autre part, d'un point de vue de la complexité algorithmique, la généralisation "exemplaire" de n'importe quelle mesure pour des séquences avec duplications ne peut qu'être plus difficile que pour des séquences sans duplications. Afin de savoir précisément où apparaît la difficulté du problème de distance exemplaire, nous nous sommes restreints aux mesures dont la version classique était simplement calculable. Ainsi, pour deux séquences, les distances de Hamming et de Levenshtein sont calculables respectivement en temps linéaire et quadratique. Des algorithmes plus recherchés mais néanmoins polynomiaux permettent également de calculer les distances d'inversion signées [84] et de DCJ [124].

Finalement, pour toutes les mesures de distance génomique envisagées, le problème de (1,2)-Distance exemplaire est APX-difficile. Obtenir un algorithme d'approximation ou paramétré pour l'une de ces distances reste un défi ouvert.

Partie II – Chapitre 4

Plus petite partition commune



Le problème de Plus petite partition commune (MCSP, pour Minimum Common String Partition) a en entrée deux séquences x et y et un entier k, et demande à partitionner ces deux séquences en utilisant un ensemble commun d'au plus k facteurs, appelés blocs (seul l'ordre des blocs pouvant être différent pour couvrir les deux séquences).

Dans ce chapitre, nous présentons un algorithme paramétré par k pour le problème MCSP.

4.1 Présentation du problème

Le problème étudié dans ce chapitre s'inscrit dans le cadre de la génomique comparative sur les chaînes, où l'objectif est de déterminer le nombre d'opérations d'un type donné permettant de transformer un génome en un autre. L'entrée consiste en une paire de chaînes x et y. L'opération permettant de transformer x en y consiste, de façon informelle, à découper x en un nombre minimal de morceaux, de les réordonner et de les concaténer de façon à obtenir exactement la chaîne y. Cette transformation se formalise grâce à la notion de *partition commune* (CSP, pour *common string partition*) : une partition $\mathcal{P}$ de deux chaînes x et y en k blocs $x_1x_2\cdots x_k$ et $y_1y_2\cdots y_k$ est une partition commune s'il existe une bijection M entre $\{x_i \mid 1 \leq i \leq k\}$ et $\{y_i \mid 1 \leq i \leq k\}$ telle que x_i représente la même chaîne de caractères que $M(x_i)$ pour tout $1 \leq i \leq k$ (voir l'exemple Figure 7). Le nombre de blocs k est appelé la *taille* de la partition commune $\mathcal{P}$. Nous étudions le problème consistant à découvrir une CSP de taille minimum.

Les résultats de ce chapitre ont été obtenus en collaboration avec Christian Komusiewicz. Ils ont été présentés aux *Journées GTGC* (2012, Lille), et ont été soumis à ESA 2013 (*21st European Symposium on Algorithms*, Sophia Antipolis [41]).

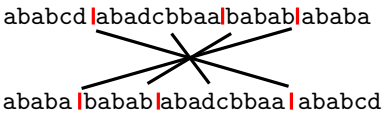


FIGURE 7 – Une instance de MCSP avec une plus petite partition commune de taille quatre.

Problème	PLUS PETITE PARTITION COMMUNE (MCSP)
Entrée	Deux chaînes x et y de longueur n et un entier k
Question	Existe-t-il une partition commune $\mathcal{P}$ de taille au plus k de x et y ?

Vu sous un autre angle, MCSP est un problème d'association. Étant données les deux séquences x et y, nous cherchons à calculer une bijection entre les marqueurs de x et ceux de y, de sorte que (1) deux marqueurs associés représentent le même caractère et (2) il existe un maximum de paires de marqueurs consécutifs qui soient associés à deux marqueurs eux-même consécutifs. L'utilisation de MCSP comme un prétraitement avant d'appliquer d'autres algorithmes donne une méthode optimale (pour la distance de cassure) permettant de transformer des chaînes en permutations.

4.2 État de l'art

MCSP a été originalement présenté indépendamment par Chen et al. [45] et Swenson et al. [120]. MCSP est APX-difficile, même dans le cas contraint où chaque caractère a au plus deux occurrences [77]. Damaschke [61] a initié l'étude de MCSP dans le contexte des algorithmes paramétrés en prouvant que ce problème est FPT si l'on considère le paramètre combiné "taille de la partition k et nombre de répétitions r dans les chaînes d'entrée". Par la suite, Jiang et al. [91] ont prouvé que MCSP peut être résolu en temps $(d!)^k \cdot \text{poly}(n)$, où d est le nombre maximum d'occurrences d'un même caractère dans les chaînes d'entrée. MCSP peut également être résolu en temps $2^n \cdot \text{poly}(n)$ [72]. Enfin, Shapira and Storer [119] ont présenté une heuristique gloutonne pour MCSP.

4.3 Notre contribution

Dans ce chapitre, nous répondons à une question ouverte [61, 72, 91] en montrant que MCSP est FPT pour le seul paramètre k, i.e. nous présentons un algorithme s'exécutant un temps $f(k) \cdot \text{poly}(n)$.

La méthode utilisée par notre algorithme est (en quelques lignes) la suivante. Nous cherchons à maintenir – et faire croître – un ensemble de facteurs des chaînes x et y qui ne contiennent aucun point de cassure. De tels facteurs sont appelés *morceaux solides*, par opposition aux *morceaux fragiles* qu'ils encadrent. Nous utilisons également une variable β représentant (à un ratio 2 près) la taille du plus grand bloc ne contenant pas encore de morceau solide. Initialement, les deux chaînes x et y sont considérées comme des morceaux fragiles, et β vaut au moins $n/2k$.

Dans un premier temps, nous découpons tous les morceaux fragiles en plus petits morceaux dont la taille est de l'ordre de $\beta/3$: au moins un nouveau bloc contient intégralement un ou plusieurs de ces morceaux. Nous explorons les cas où chacun de ces morceaux est solide ou fragile.

Dans un second temps, nous mettons à jour β, et cherchons à réduire les morceaux fragiles de façon à ce que leur taille soit bornée par une fonction de β et de k. L'application de plusieurs règles de réduction permet d'arriver à ce résultat.

Enfin, nous itérons ce processus (premier et second temps) jusqu'à ce que les morceaux fragiles soient suffisamment petits, avec $\beta < 4$. Alors, le nombre de positions possibles pour les points de cassure est suffisamment faible pour pouvoir tester tous les cas possibles, et ainsi obtenir une partition commune optimale.

4.4 Perspectives

Outre l'amélioration du temps d'exécution (qui est pour le moment de l'ordre de $O^*(k^{21k^2})$) qui pourrait se faire via le calcul d'un noyau polynomial, nous avons identifié plusieurs pistes qui permettraient de tenir compte de données plus générales. Nous cherchons donc à savoir si l'algorithme peut être étendu de façon à tenir compte :

A. de signes pour chaque marqueur ? Plus précisément, une direction est associée à chaque marqueur, et les blocs peuvent être retournés avant d'être mis en correspondance. Des variantes signées de MCSP ont été définies dans [45, 69, 120].

B. de trois séquences d'entrée ou plus ?

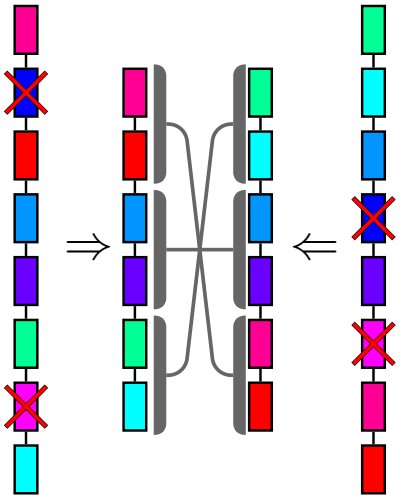
C. de la possibilité qu'un nombre borné de marqueurs soient faux et aient besoin d'être retirés ? Ces marqueurs pourraient se placer entre deux blocs consécutifs (ils correspondent alors à des gènes n'apparaissant dans aucun bloc de synténie) ou bien à l'intérieur d'un même bloc (ils sont alors dus à des erreurs lors de la reconstruction du génome).

D. de la présence d'incertitudes dans le nombre de répétitions consécutives d'un même facteur ? En effet, une difficulté majeure du problème de reconstruction est de déterminer avec exactitude le nombre de répétitions des chaînes périodiques. Les données d'entrée, au lieu d'être de la forme "`c a b a b a b a c`", pourraient par exemple être "`c (a b)`{de 2 à 4 copies} `a c`".

Pour A. et B., nous conjecturons que l'algorithme pourrait être assez simplement adaptable, puisque notamment (pour A.) il est possible, étant donné un morceau solide, de tester les cas où le bloc contenant ce morceau doit être retourné ou non. Pour C. et D., les modifications sont nécessairement plus profondes et impliquent de redéfinir des outils clé. Nous conjecturons cependant que la méthode générale devrait permettre de résoudre ces problèmes plus difficiles.

Partie III – Chapitre 5

Extraction de bandes maximales



Le problème d'Extraction de bandes maximales (MSR, pour Maximal Strip Recovery) cherche à identifier des blocs de synténie dans des séquences génomiques pouvant contenir des erreurs. Ainsi, un nombre minimal d'éléments doivent être supprimés afin de pouvoir partitionner les séquences en bandes identiques de longueur au moins 2.

Dans ce chapitre, nous étudions plusieurs variantes de MSR, en particulier la variante δ-*gap* où au plus δ éléments consécutifs peuvent être supprimés, et nous présentons dans chaque cas des résultats de complexité algorithmique, ainsi que des approximations ou des algorithmes paramétrés.

5.1 Présentation des problème MSR et CMSR

Une tâche essentielle en génomique comparative consiste à décomposer deux ou plusieurs génomes en blocs de synténie, c'est-à-dire à identifier des fragments de chromosomes avec des contenus similaires. Les blocs de synténie représentent des zones du génome n'ayant pas été affectées par les réarrangements de grande échelle, tels que les transpositions ou les inversions, et ainsi peuvent servir de données d'entrée pour les problèmes de réarrangements avancés. Ils donnent aussi des indices importants sur le rôle de chaque gène, étant donné que des gènes appartenant à un même bloc de synténie produisent fréquemment des protéines ayant des fonctionnalités liées. Extraire les blocs de synténie des cartes génomiques est, cependant, une

Ce chapitre regroupe des résultats provenant de deux articles. Le premier a été présenté à ISAAC 2009 (20th International Symposium on Algorithms and Computation, Honolulu [31]), et a été publié dans JDA 2013 (Journal of Discrete Algorithms [36]). Le second, créé en collaboration avec Minghui Jiang, a été présenté à CPM 2011 (22nd Annual Symposium on Combinatorial Pattern Matching, Palermo [29]) et a été publié dans TCS 2012 (Theoretical Computer Science [30]).

tâche non triviale lorsque les données sont bruitées ou ambiguës. Il faut alors retirer les marqueurs erronés de façon à pouvoir proposer une décomposition claire en blocs de synténie. Ceci motive le problème d'Extraction de bandes maximales (MSR, pour maximal Strip Recovery) [128] : retirer un ensemble de marqueurs (gènes) des cartes génomiques de façon à ce que les marqueurs restants puissent être partitionnés en bandes (blocs de synténie) ayant une taille totale maximale.

Les données génomiques prennent ici la forme de *carte génomique*, i.e. une séquence signée. Une *bande* pour $d \geq 2$ cartes génomiques est un facteur commun à chacune de ces cartes (potentiellement en ordre inverse avec les signes opposés). Le problème MSR sur d cartes, noté MSR-d (ou directement MSR pour $d = 2$) est le problème de maximisation suivant.

Problème	MSR-d
Entrée	d cartes génomiques $\mathcal{M}_{1\ldots d}$ [1] contenant chacune n marqueurs sans doublons
Sortie	d sous-séquences $\mathcal{M}'_{1\ldots d}$ de $\mathcal{M}_{1\ldots d}$ respectivement, chacune contenant les mêmes ℓ marqueurs, telles que les marqueurs de $\mathcal{M}'_{1\ldots d}$ peuvent être partitionnés en bandes
Maximiser	le nombre ℓ de marqueurs sélectionnés

Exemple 2. *Soient les trois cartes génomiques ci-dessous.*

$\mathcal{M}_1 = \langle$ 1 2 3 12 4 5 8 9 10 6 7 11 $\rangle$

$\mathcal{M}_2 = \langle$ 1 -7 -6 2 12 3 4 10 5 11 8 9 $\rangle$

$\mathcal{M}_3 = \langle$ 12 4 1 5 2 3 10 11 6 -9 7 -8 $\rangle$

Les trois sous-séquences de longueur $\ell = 8$ ci-dessous forment une solution optimale de MSR-3 sur ($\mathcal{M}_1$, $\mathcal{M}_2$, $\mathcal{M}_3$). Ces sous-séquences peuvent être partitionnées par l'ensemble de bandes $\{\langle 1, 2\rangle, \langle 12, 4\rangle, \langle 10, 11\rangle, \langle 8, 9\rangle\}$. Les sous-séquences correspondant à chaque bande sont mises en valeur à la fois dans les cartes originales et dans la solution.

$\mathcal{M}'_1 = \langle$ 1 2 12 4 8 9 10 11 $\rangle$

$\mathcal{M}'_2 = \langle$ 1 2 12 4 10 11 8 9 $\rangle$

$\mathcal{M}'_3 = \langle$ 12 4 1 2 10 11 -9 -8 $\rangle$

Le problème de maximisation MSR-d a un problème de minimisation complémentaire, dénoté CMSR-d [122, 95], qui minimise le paramètre $k = n - \ell$ représentant le nombre de marqueurs supprimés. Pour des cartes génomiques provenant d'espèces suffisamment proches avec peu d'erreurs, le paramètre k est plus petit que ℓ, rendant ainsi les algorithmes FPT ou les approximations plus pertinents pour CMSR-d que pour MSR-d.

La variante ci-dessous [47] a été proposée pour le cas plus général où les doublons sont autorisés dans les cartes génomiques :

1. Par simplicité, une liste de d cartes génomiques $(\mathcal{M}_1, \ldots, \mathcal{M}_d)$ est abréviée en $\mathcal{M}_{1\ldots d}$ et $(\mathcal{M}'_1, \ldots, \mathcal{M}'_d)$ en $\mathcal{M}'_{1\ldots d}$

Problème	MSR-DU-d
Entrée	d cartes génomiques $\mathcal{M}_{1...d}$ contenant chacune n marqueurs
Sortie	d sous-séquences $\mathcal{M}'_{1...d}$ de $\mathcal{M}_{1...d}$ respectivement, chacune contenant les mêmes ℓ marqueurs, telles que les marqueurs de $\mathcal{M}'_{1...d}$ peuvent être partitionnés en bandes
Maximiser	le nombre ℓ de marqueurs sélectionnés

5.2 Contrainte de *gap*

Étant données d sous-séquences $\mathcal{M}'_{1...d}$ de d cartes génomiques $\mathcal{M}_{1...d}$, respectivement, le *gap* entre deux marqueurs consécutifs a et b de $\mathcal{M}'_i$ est le nombre de marqueurs apparaissant entre a et b dans $\mathcal{M}_i$, a et b exclus. Le *gap* d'une bande s est le gap maximum entre deux marqueurs consécutifs de s sur toutes les cartes $\mathcal{M}'_i$. Dans l'Exemple 2, la bande $\langle 1, 2 \rangle$ dans la solution proposée a un gap de 2, les marqueurs 1 et 2 étant séparés par les marqueurs 6 et 7 dans la carte $\mathcal{M}_2$. Les marqueurs retirés entre deux marqueurs consécutifs d'une bande correspondent au bruit et aux ambiguïtés : il est donc improbable d'avoir un bloc de synténie ayant un gap très grand. Cela nous amène donc à définir la version contrainte ci-dessous :

Problème	δ-gap-MSR-d
Entrée	d cartes génomiques $\mathcal{M}_{1...d}$ contenant chacune n marqueurs sans doublons
Sortie	d sous-séquences $\mathcal{M}'_{1...d}$ de $\mathcal{M}_{1...d}$ respectivement, chacune contenant les mêmes ℓ marqueurs, telles que les marqueurs de $\mathcal{M}'_{1...d}$ peuvent être partitionnés en bandes, et telles que chaque bande a un gap d'au plus δ
Maximiser	le nombre ℓ de marqueurs sélectionnés

Dans l'Exemple 2, les bandes des sous-séquences $(\mathcal{M}'_1, \mathcal{M}'_2, \mathcal{M}'_3)$ ont un gap d'au plus 2, elles forment donc une solution optimale de δ-gap-MSR-3 pour $\delta = 2$. Pour $\delta = 1$, la solution optimale contient seulement 6 marqueurs répartis en 3 bandes : $\{\langle 2, 3 \rangle, \langle 4, 5 \rangle, \langle 6, 7 \rangle\}$, voir ci-dessous.

$\mathcal{M}_1 = \langle$ 1 **2 3** 12 **4 5** 8 9 10 **6 7** 11 $\rangle$

$\mathcal{M}_2 = \langle$ 1 **-7 -6** **2** 12 **3** **4** 10 **5** 11 8 9 $\rangle$

$\mathcal{M}_3 = \langle$ 12 **4** 1 **5** **2 3** 10 11 **6** -9 **7** -8 $\rangle$

Les variantes contraintes de MSR-DU-d et CMSR-d, dénotées respectivement δ-gap-MSR-DU-d et δ-gap-CMSR-d, peuvent être définies de façon similaire. De même que pour MSR-d et CMSR-d, le paramètre pour δ-gap-MSR-d est ℓ, et le paramètre pour δ-gap-CMSR-d est k. Il n'y a pas de réduction directe de δ-gap-MSR-d à MSR-d ou vice versa : le gap est une contrainte supplémentaire à prendre en compte, mais qui permet néanmoins de réduire le champ de recherche puisque moins de bandes sont autorisées.

5.3 Résultats

Les résultats de complexité et les algorithmes existant pour les nombreuses variantes de MSR sont présentés dans la Table 1. Les résultats les plus importants

TABLE 1 – Résultats algorithmiques pour les variantes de MSR. La variable d représente le nombre de cartes génomiques d'entrée ($d = 2$ pour MSR, MSR-DU et CMSR). Pour chaque résultat, une référence est donnée à la section de ce manuscrit et/ou à l'article le présentant. Il n'y a pas de colonne pour MSR-DU-d avec $d \geq 3$ par manque de résultats spécifiques à ce problème.

Classe de complexité

contrainte de gap	MSR	MSR-d $d\geq3$	MSR-DU	CMSR	CMSR-d $d\geq3$
$\delta = 0$	P 5.3.1			P 5.3.1	
$\delta = 1$	NP-complet 5.2.2		APX-complet	NP-complet 5.2.2	
$\delta = 2$	APX-complet				
$\delta \geq 3$	5.2.3 et [95]		5.2.4	APX-complet	
non	W[1]-difficile ($d \geq 4$) et APX-complet [94, 95, 123]			[95, 123]	

Meilleur ratio d'approximation

contrainte de gap	MSR	MSR-d $d\geq3$	MSR-DU	CMSR	CMSR-d $d\geq3$
$\delta = 0$	P 5.3.1		2.25 5.3.5	P 5.3.1	
$\delta = 1$	1.8 5.3.3	$0.75d+0.75+\epsilon$		2.778 5.3.4	
$\delta = 2$	$1.5d + \epsilon$				
$\delta = 3$	$1.5d + 0.75 + \epsilon$ 5.3.2			$d + 1.5$	
$\delta \geq 4$	$2d$				
non			5.3.1 et [47, 95]	2.667 [102]	5.3.6

Meilleur temps d'exécution d'un algorithme paramétré

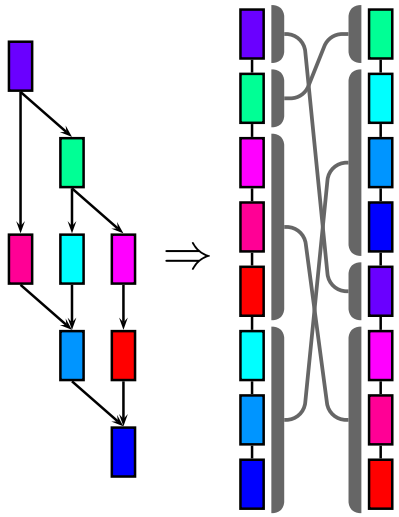
contrainte de gap	MSR	MSR-d $d\geq3$	MSR-DU	CMSR	CMSR-d $d\geq3$
$\delta = 0$	P			P	
$\delta = 1$	$O(2^t td\delta^2 + nd\delta)$			$O(2^k \operatorname{poly}(nd))$ 5.4.3	
$\delta = 2$	avec $t = \ell(1 + \frac{3}{2}d\delta)$		?		
$\delta \geq 3$	5.4.1			$O(2.36^k \operatorname{poly}(nd))$	
non	? (W[1]-difficile pour $d \geq 4$ [94])			Noyau linéaire [90] 5.4.2	

sont les suivants.

Toutes les variantes sont au moins NP-difficiles (exceptées les variantes "triviales" 0-gap-MSR et 0-gap-CMSR). MSR-d est, de plus, W[1]-difficile pour $d \geq 4$ [95]. Une $2d$-approximation est disponible pour MSR (avec ou sans contrainte de gap, avec ou sans doublons). Nous proposons des algorithmes plus précis pour δ-gap-MSR avec des petites valeurs de δ, notamment une 1.8-approximation pour 1-gap-MSR. CMSR (et δ-gap-CMSR-d) est également approximable, avec un ratio plus intéressant : $d + 1.5$. Enfin, nous proposons un algorithme paramétré pour CMSR-d et δ-gap-CMSR-d de complexité $O(2.36^k \operatorname{poly}(nd))$.

Partie III – Chapitre 6

Linéarisation avec distance de cassure minimale



Le problème NP-difficile de Linéarisation avec distance de cassure minimale (MBL, pour Minimum Breakpoint Linearization) a pour objectif de reconstruire un génome linéaire à partir de données partiellement ordonnées, en utilisant en référence le génome d'une espèce proche. Formellement, l'objectif est, étant donné un ordre partiel Π, de créer une permutation compatible avec l'ordre partiel et minimisant la distance de cassure à la permutation identité de référence.

Dans ce chapitre, nous présentons trois algorithmes d'approximation pour ce problème, ainsi qu'un algorithme paramétré.

6.1 Présentation du problème

Dans la majorité des algorithmes de génomique comparative, une connaissance complète de l'ordre des gènes sur chaque chromosome des espèces étudiées est requise (c'est le cas dans tous les problèmes étudiés jusqu'à présent). Cependant, malgré des avancées rapides en séquençage ADN, une connaissance complète de l'ordre des gènes n'est acquise que pour un nombre limité d'espèces, alors que pour les autres espèces l'information se présente plus souvent comme un ensemble de cartes génomiques où des incertitudes sur l'ordre précis des gènes persistent. Ainsi, le problème qui consiste à inférer un ordre total à la fois compatible avec l'information partielle des cartes génomiques et qui optimise une fonction objectif pertinente est une première étape nécessaire pour pouvoir étudier les génomes d'un maximum d'espèces. Durant les dernières années, une attention croissante a été portée sur ce problème, dans

Les résultats de ce chapitre ont été présentés à TAMC 2010 (7th Annual Conference on Theory and Applications of Models of Computation, Prague [32]), et sont à paraître dans TCS (Theoretical Computer Science, [38]).

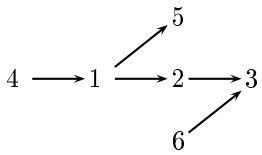


FIGURE 8 – Graphe orienté acyclique représentant l'ordre partiel Π de l'Exemple 3.

lequel la fonction objectif est une distance d'évolution mesurée par rapport à un génome de référence (e.g. le nombre de réarrangements [127], d'inversions [126, 73], de cassures [73, 23, 44], ou d'intervalles communs [23]).

Dans ce chapitre, nous nous focalisons sur le problème de LINÉARISATION AVEC DISTANCE DE CASSURE MINIMALE (MBL, pour MINIMUM BREAKPOINT LINEARIZATION), dont le but est de calculer une linéarisation d'un ordre partiel minimisant la distance de cassure par rapport à un génome de référence.

Pour décrire le problème MBL, nous représentons les données génomiques incomplètes par un ordre partiel Π sur un ensemble donné de marqueurs $\Sigma = [\![1\,;\,n]\!]$. Une *linéarisation* de Π est une permutation (i.e. un ordre total) $\pi = \langle \pi[1], \pi[2], \ldots, \pi[n] \rangle$ sur Σ, telle que, pour toute paire de marqueurs i, j, si $i <_\Pi j$ alors $i <_\pi j$ (de façon équivalente, si i précède j dans Π, alors i précède j dans π). Dans ce cas, π est dite *compatible* avec Π. Une *adjacence* dans une permutation π est un facteur de longueur 2. La *distance de cassure* $d_B(\pi_1, \pi_2)$ entre deux permutations π_1 et π_2 (du même ensemble Σ) est définie comme étant le nombre d'adjacences de π_1 qui ne sont pas des adjacences de π_2.

Le problème de LINÉARISATION AVEC DISTANCE DE CASSURE MINIMALE est défini comme suit :

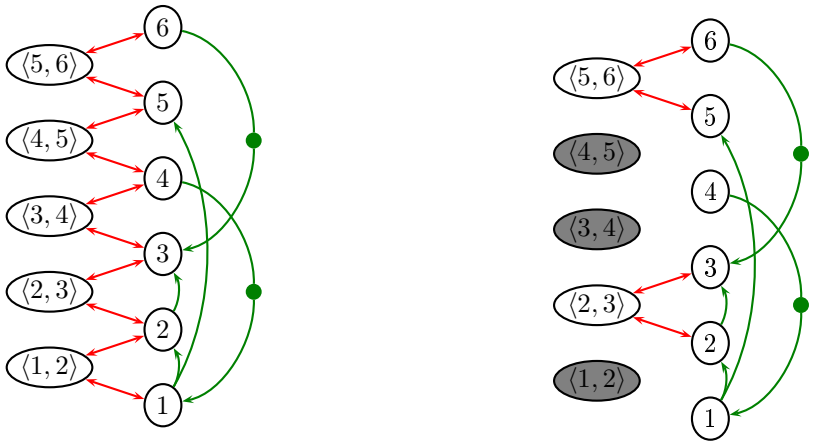
Problème	MBL
Entrée	Un ordre partiel Π
Sortie	Une linéarisation π de Π
Minimiser	$k = d_B(\pi, \mathcal{I}_n)$

Exemple 3. *Soit* Π *l'ordre partiel défini par* $1 <_\Pi 2 <_\Pi 3$ *;* $4 <_\Pi 1 <_\Pi 5$ *et* $6 <_\Pi 3$ *(voir Figure 8). Cet ordre partiel engendre quatre linéarisations optimales pour le problème MBL, avec une distance de cassure de 3. Elles sont énumérées ci-dessous. Les adjacences communes avec* $\mathcal{I}_n$ *sont soulignées.*

$$\langle 4,\ 1,\ \underline{5,\ 6},\ \underline{2,\ 3} \rangle$$
$$\langle 4,\ \underline{1,\ 2},\ \underline{5,\ 6},\ 3 \rangle$$
$$\langle 6,\ 4,\ \underline{1,\ 2,\ 3},\ 5 \rangle$$
$$\langle 4,\ 6,\ \underline{1,\ 2,\ 3},\ 5 \rangle$$

Il est intéressant de souligner qu'en pratique, l'ordre partiel est obtenu en combinant un nombre limité m de *cartes génomiques* [125, 127], où une carte génomique est une suite ordonnée de blocs $B_1, B_2, \ldots, B_q$, qui sont chacun un ensemble non ordonné de marqueurs. Ces blocs induisent un ordre partiel Π de la façon suivante : pour tout $a \in B_i$ et $b \in B_j$, $a <_\Pi b$ ssi $i < j$. Ainsi, il n'est pas nécessaire que chaque carte génomique contienne tous les marqueurs, et nous supposons que l'assemblage des cartes génomiques ne crée pas de conflits. L'ordre Π de l'Exemple 3 est induit, par exemple, par les deux cartes ci-dessous.

$$\{4\} \longrightarrow \{1\} \longrightarrow \{2, 5\},$$
$$\{2, 6\} \longrightarrow \{3\}.$$



(a) Graphe G_Π. Les arcs de Π sont en vert, les arcs de conflit sont marqués d'un point.

(b) Résultat de SUBSET-FVS sur G_Π : $\{\langle 1,2\rangle, \langle 3,4\rangle, \langle 4,5\rangle\}$. Les cycles du graphe réduit ne contiennent pas d'arc de conflit. Les adjacences conservées, $\langle 2,3\rangle$ et $\langle 5,6\rangle$, correspondent à la linéarisation optimale $\langle 4,\ 1,\ \underline{5,\ 6},\ \underline{2,\ 3}\rangle$.

FIGURE 9 – Construction et utilisation du graphe d'ordre-adjacence G_Π pour l'ordre partiel Π défini dans l'Exemple 3.

6.2 État de l'art

Le problème MBL, fondé sur le problème de réarrangement de génome défini par Zheng et Sankoff [127], a été étudié indépendamment dans [23] et [73] (dans ce dernier, le problème est nommé PBD, et traite de deux ordres partiels plutôt qu'un ordre partiel et un ordre total). Dans [23], Blin et al. prouvent que MBL est NP-difficile et donnent deux algorithmes pour le résoudre : (i) une heuristique et (ii) un algorithme exact, et donc de complexité exponentielle, basé sur la programmation dynamique. De plus, ce dernier est efficace plus particulièrement lorsque l'ordre partiel d'entrée est construit à partir d'un nombre m borné de cartes génomiques, chacune ayant des blocs de taille bornée. Il est souligné dans [38] que la preuve de NP-difficulté de [23] est, en fait, une preuve que MBL est APX-difficile.

Dans [73], Fu et Jiang proposent une preuve de NP-difficulté indépendante, et présentent la définition d'un graphe d'ordre-adjacence d'un ordre partiel permettant de produire des algorithmes heuristiques ou d'approximation [44]. Cependant le théorème central dans [73] est faux (l'Exemple 3 est en fait un contre-exemple pour ce théorème [38]), ce qui nous pousse à définir une nouvelle structure permettant de résoudre MBL.

6.3 Notre contribution

Dans ce chapitre, nous présentons une nouvelle définition de graphe d'ordre-adjacence : cette construction permet de réduire MBL à une variante du problème

de couverture de cycles, Subset-FVS. En utilisant des résultats connus pour ce problème [67, 49], nous déduisons deux algorithmes d'approximation et un algorithme paramétré pour MBL. Enfin, nous créons aussi une $O(m^2)$-approximation pour les données obtenues en combinant m cartes génomiques.

Nous détaillons ci-dessous la construction de ce graphe G_Π pour l'ordre partiel Π de l'Exemple 3. Le graphe G_Π (voir Figure 9a) a un sommet pour chaque marqueur, et un sommet pour chaque adjacence de l'identité. Un arc double relie chaque adjacence aux deux marqueurs qu'elle contient. Enfin, les arcs de Π sont reportés entre les marqueurs correspondants. Les arcs de Π allant d'un marqueur i à un marqueur j avec $i < j$ sont appelés arcs de conflits. Nous prouvons que si un cycle contient un arc de conflit, alors les adjacences utilisées par ce cycle ne peuvent pas être réalisées simultanément dans une linéarisation de Π. Ainsi, MBL peut être réduit au problème consistant à retirer un minimum d'adjacences du graphe G_Π pour que ce graphe ne contienne plus aucun cycle utilisant un arc de conflit. Ce problème correspond à trouver un ensemble minimal de sommets couvrant les cycles "interdits" d'un graphe (voir Figure 9b) : il a déjà été étudié sous le nom de Subset-FVS. Les résultats pour ce problème [67, 49] nous permettent d'obtenir (en notant k la distance de cassure optimale et X l'ensemble des arcs de conflits) :

- un algorithme FPT de temps $2^{2^{O(k)}} n^{O(1)}$,
- une $O\left(\log^2 |X|\right)$-approximation,
- une $O\left(\log(k) \log\log(k)\right)$-approximation,

Enfin, nous proposons un algorithme d'approximation ad hoc, dont le ratio est de m^2+4m-4. En revanche, la question de l'existence d'un algorithme d'approximation à ratio constant reste ouverte.

Conclusion et perspectives

En suivant le plan de ce manuscrit, nous rappelons les résultats obtenus dans chaque chapitre et présentons des axes possibles de recherche.

Les problèmes de tri sur des permutations ne sont pas toujours simples

Comme nous l'avons vu dans les Chapitres 1 et 2, la comparaison de séquences génomiques sous le modèle le plus simple (où chaque gène est unique) n'est pas toujours triviale. Et bien qu'il est possible d'y calculer plusieurs mesures de dissimilarité telles que la distances de cassure, des distances d'édition, etc., et même des distances de réarrangement avec les inversions et le DCJ signés, nous avons prouvé que les problèmes de Tri par transpositions et Tri par retournements préfixes sont NP-difficiles. Cependant, nous conjecturons que ces problèmes devraient admettre des algorithmes d'approximation plus précis que ceux existant (avec des ratios 1.375 pour le premier et 2 pour le second).

Un autre problème de réarrangement, pour lequel la complexité est toujours ouverte, est le problème de tri par retournements préfixes sous le modèle signé. La contrainte de signe réduit le nombre d'opérations efficaces et devrait faciliter la recherche de telles opérations, de même que pour le tri par inversions ou par DCJ.

Enfin, existe-t-il des algorithmes paramétrés, avec un paramètre pertinent d'un point de vue biologique, pour ces problèmes NP-difficiles ?

Comparer des séquences avec des duplications est encore plus complexe

Les problèmes de comparaison deviennent immédiatement plus difficiles lorsque les données contiennent des gènes dupliqués. Une approche classique pour traiter ces données consiste à rechercher un scénario de duplications permettant de faire la distinction entre les différentes copies et ainsi retrouver un modèle de permutation pour des applications ultérieures.

Dans le scénario exemplaire (voir Chapitre 3), nous considérons qu'une seule copie de chaque gène provient de l'ancêtre commun, et que les autres copies sont dues à des récentes duplications. Nous cherchons donc à identifier cette copie "originale". Cependant, pour toute fonction d'optimisation considérée, le calcul de la Distance Exemplaire est APX-difficile. Le modèle exemplaire est dominé par les problèmes ouverts, puisqu'aucune approximation à ratio constant ou algorithme FPT n'est connu, quelque soit la distance de référence considérée.

Dans le modèle complet, où les copies de chaque gène proviennent de l'ancêtre commun et peuvent donc être mis en bijection avec les copies de l'autre séquence, les problèmes sont également difficiles. En revanche, nous montrons que le problème MCSP (qui peut être vu comme la distance de cassure sous le modèle complet) est

FPT. Malheureusement, notamment à cause de sa complexité en $O(k^{21k^2}\,\mathrm{poly}(n))$, il n'est pas pertinent pour des applications pratiques. Existe-t-il un algorithme efficace à la fois d'un point de vue théorique et pratique ? En effet, même si ce problème ne tient pas compte des suppressions de gènes, et impose que les chaînes aient le même nombre d'occurrences de chaque lettre, il s'agit d'un problème central lorsqu'il s'agit de traiter des génomes avec duplications.

Corriger et compléter les données génomiques est essentiel

Un défi important de la génomique comparative est de tenir compte des erreurs induites par les processus générant les données d'entrée. Un exemple représentatif est l'assemblage de génome par séquençage à haut débit (NGS) : les données ainsi générées contiennent habituellement des forts pourcentages d'erreur, compensés par une forte redondance des lectures du génome. Mais les erreurs, les ambiguïtés et les manques existent dans quasiment toutes les données génomiques. Idéalement, tous les problèmes de génomique comparative devraient tenir compte de ce fait. Cependant, à cause de la difficulté algorithmique que cela représente, le nettoyage des données est généralement confié à des heuristiques, utilisées en tâche de prétraitement.

Deux de ces tâches sont présentées dans les Chapitres 5 et 6. Dans le Chapitre 5, le problème d'Extraction de bandes maximales et ses variantes, cherche à retirer les marqueurs de séquences génomiques d'espèces proches qui ne semblent appartenir à aucun bloc de synténie. Nous présentons une palette d'algorithmes et de résultats de difficulté, avec en particulier un algorithme paramétré efficace pour ce problème.

Enfin dans le Chapitre 6, nous nous focalisons sur le problème consistant à inférer un ordre total sur les gènes (une *linéarisation*) compatible avec des données incomplètes (représentées par un ordre partiel). Les problèmes traitant de génomes partiellement ordonnés ont souvent pour but de retirer les conflits de façon à obtenir *une* linéarisation. Dans le problème MBL, nous cherchons à calculer une linéarisation *optimale* à partir d'un ordre partiel sans conflits. L'optimalité est mesurée comme étant la distance de cassure à un génome de référence. Nous proposons une méthode de résolution basée sur la théorie des graphes, et plus particulièrement sur le problème Subset-FVS, afin d'obtenir plusieurs algorithmes d'approximation et un algorithme paramétré.

Plus de perspectives

La question centrale reste ouverte : est-ce que les méthodes traditionnelles de comparative génomique peuvent s'étendre pour traiter des données bruitées ou incomplètes ? Par exemple, peut-on calculer une distance de DCJ sur des génomes partiellement ordonnés ? Une voie de réponse potentielle serait par le problème de Plus petite partition commune. En effet, l'algorithme que nous avons présenté semble pouvoir être adapté à des données ayant un nombre borné d'erreurs locales, ou bien avec des incertitudes sur le nombre de répétitions dans les sous-chaînes périodiques. De nombreuses méthodes sont toujours à explorer – ou à inventer – afin de produire des algorithmes à la fois précis et efficaces d'un point de vue théorique, et qui considèrent les nombreuses facettes des problèmes biologiques.

En s'étendant hors de la génomique comparative seule, une large gamme d'aspects de la bioinformatique peuvent faire l'objet de recherches. De nombreux pro-

blèmes peuvent, par exemple, être modélisés par des graphes. Avec des données telles que les réseaux métaboliques ou les interactions entre gènes, on vise à extraire des sous-graphes remarquables, à discerner des motifs, etc. La théorie de la paramétrisation, profondément ancrée en théorie des graphes via par exemple les méthodes de calcul de noyau, apporte certainement une approche idéale à de tels problèmes.

Manuscript

Manuscrit (partie anglaise)

Introduction

Bioinformatics is the field at the intersection of biology and computer science, where one aims at extracting and processing information from biological data, in order to obtain a better understanding of the living. From one side, biology contributes with data gathered through experiments and poses the questions that should enable to understand more about the subject at hand. From the other side, computer science brings formal methods to analyze the problems and to produce solutions as precisely and efficiently as possible. The benefits are obvious for biologists, who can thus verify hypotheses and make predictions in more reliable and efficient ways. They are also important for computer scientists, who are presented with unprecedented challenges. The variety of problems led to the exploration of original data structures and the design of new algorithmic methods, enriching the whole field of computer science with an increased panel of possible approaches for other problems.

The use of bioinformatics becomes even more necessary when the data that needs analyzing becomes larger and larger. The best-known example is DNA sequencing: it is now possible to read DNA sequences off chromosomes at a very high rate (e.g., the Beijing Genome Institute outputs 10 terabytes of raw nucleotide sequences every day [109]). DNA sequences are at the core of the majority of biochemical processes. However, it is still a utopian task to perfectly reconstruct a genome sequence, let alone to understand all the intricate functions that originate from it. Another important field is network analysis where one aims at understanding the complex interactions between different molecules (proteins, RNA, metabolites, etc.) in the cell. The input data consists of a set of possible reactions, such as "protein A can transform metabolite B into metabolite C" or "proteins A and B can combine to form protein C". One aims at understanding large-scale processes, such as cell cycles or protein synthesis, with the possible objective of designing a drug acting on one specific process.

In this thesis, we study combinatorial responses to problems stemming from comparative genomics. The common denominator of our problems is the presence in input of genomic information, usually in the form of gene or marker sequences, from two different species. As Nature would have it, both genomes have evolved from a common ancestor and now display a number of similarities: the objectives of comparative genomics problems are to pinpoint these similarities, measure them, and use them in order to gather relevant biological information.

Depending on the modeling of the genomes and the biological question at hand, we obtain many challenging combinatorial problems, which can be studied in the usual computational complexity framework. For each problem, we try to provide a polynomial-time algorithm or, failing that, to prove that the problem is NP-hard. In

the latter case, efficient algorithms can be sought in two directions, either as approximation algorithms (that is, polynomial-time algorithms giving a sub-optimal solution, but with a bounded error factor) or as parameterized algorithms (exponential-time exact algorithms which are nevertheless efficient for small values of a relevant parameter). We refer the reader to [69] for a detailed review dedicated to combinatorial problems in comparative genomics.

The chapters of this manuscript are each devoted to specific comparative genomics problems; they are organized in three parts based on the models used for input genomes. Part I is focused on rearrangement distances for the most favorable model, where each gene is unique within a genome. In Part II we explore two problems where input sequences may present duplications. Finally in Part III we extend our study to problems where input data is incomplete or corrupted. In this Introduction we give general motivation and some comparative genomics literature results, whereas in each chapter we present a more specific state of the art focused on the respective problems.

Some biological background.

As the genetic information supported by DNA is copied again and again, from cell to cell and from ancestor to descendant in repetitive biochemical processes, a large number of errors may occur: they are the steps of evolution. These steps sometimes have no effect at all, but they can also lead to the "discovery" of new biological capabilities... or to the death of the cell. Moreover, they may occur either at a small scale, affecting only a few nucleotides in the whole DNA, or at a larger scale when whole strips of DNA are cut, reinserted or duplicated.

A key tool in the study of genetic evolution is the notion of synteny blocks, i.e., strips of DNA with similar content across different species: we assume that such blocks belonged to the common ancestor of the species, and have remained unaffected by evolution. Two species with scattered and small synteny blocks are more likely to have evolved as different species for a long time, which means they have a comparatively old common ancestor. Moreover, an unusually large synteny block belonging to a large number of species is likely to contain genes somehow working together, e.g. the proteins they code for have interacting roles, one gene regulates another, etc. The counterpart of synteny blocks is the notion of breakpoints. Formally, in the comparison of two genomes, there is a breakpoint between two consecutive genes of one genome if these genes are separated in the other genome, which implies they do not belong to the same synteny block. The number of breakpoints is easy to compute – provided each gene appears exactly once in each genome and the data is error-free – and can be considered as a first measure of dissimilarity between two genomes.

Some difficulties arise in the presence of *paralogs*. Genes are paralogs if they are within the same genome and have similar sequences. (They are opposed to *orthologs*, which are similar genes in the genomes of different species). Paralogs create an ambiguity to discover synteny blocks, since two regions with similar contents may not have evolved from the same region of the common ancestor. It is thus a task of comparative genomics to distinguish, among all paralogs in two species, which ones have evolved from a single gene of the common ancestor, and which ones have undergone different evolution processes.

Some computational complexity background.

We assume that the reader is already familiar with usual notions of computational complexity such as polynomial-time exact and approximation algorithms, NP-hardness, and APX-hardness. In a nutshell, we consider that an algorithm is efficient if its required running time for an input of size n is bounded by a polynomial on n. A problem in P can be solved exactly by such an efficient algorithm, while NP-hard problems are unlikely to be solvable by exact polynomial-time algorithms. A usual strategy for NP-hard optimization problems is to try to create approximation algorithms, that is, algorithms running in polynomial time, whose output solution can be at some ratio r away from the optimal solution (where r should be as close to 1 as possible). APX-hardness gives a lower bound on the approximation ratio; without such bound, it is possible to create a polynomial-time approximation scheme (PTAS), i.e. an approximation algorithm that can reach any approximation ratio $1 + \epsilon$ for $\epsilon > 0$ but whose time complexity increases as ϵ decreases.

Another possible approach is the creation of fixed-parameter tractable (FPT) algorithms which aim at solving exactly yet efficiently NP-hard problems. The first step consists in identifying a parameter of the instances, typically written k, which summarizes the level of complexity of the problem for each particular instance. Then, an FPT algorithm would run in time $f(k)n^d$, where d is a fixed integer and f is any function (usually an exponential) that does not depend on n. Such algorithms are still exponential in the general case, but they are efficient when k is bounded to small values, which should be the case for the majority of "real-life" instances.

For more detailed theoretical background, classic techniques and results, we refer the reader to Garey & Johnson's founding textbook [75] for NP-completeness theory, to Papadimitriou & Yannakakis [108] for approximability, and to Niedermeier [107] for parameterized complexity (see also [63, 71]).

Problems Studied in this Thesis

Part I – Distances Between Permutations

In the first part, we consider that genomes can be represented as permutations, that is, the genes of both input sequences are matched one-to-one, and only the order differs. In fact, we have either unsigned or signed permutations: unsigned ones are habitual permutations over $\{1, 2, \ldots, n\}$, and for signed permutation each gene comes with additional information representing its orientation on the chromosome, written with a sign $+$ or $-$. This model may lack some generality, but it allows for the computation of many simple distance measures, starting with the breakpoint distance. Other classic examples include the common and conserved intervals distances [121, 86, 18], or Hamming and Levenshtein distances [82, 101]. In the 1990s were introduced *rearrangement* distances (see e.g. [97, 13]): one considers a natural evolution operation and searches the shortest sequence of such operations needed to transform one genome into the other. The resulting scenario gives a precious insight in the evolution history of the two species since their common ancestor.

Several biologically relevant operations can be considered. The most studied ones are probably reversals (a subsequence of the chromosome is cut out and reinserted in the opposite direction), and transpositions (a subsequence is cut out and reinserted at a different place, in the same direction). More recently, the Double-Cut-and-Join

(DCJ) operation has been introduced [124, 17]: the chromosome is cut at two positions, and the four created endpoints can be reattached in any way (this operation requires a slightly more general model for genomes, where circular permutations are considered). Finally, other possible operations include transreversals (similar to transpositions, but the subsequence is reinserted in the opposite direction), block interchanges (two subsequences exchange their positions), and prefix-constrained variants of any of these operations (in each case, the subsequence must contain one end of the chromosome).

For the signed reversal distance, i.e., the rearrangement distance such that the only operations considered are reversals and the genomes are modeled as signed permutations, Bafna & Pevzner [14] provided a polynomial-time algorithm, which has now been improved into a linear-time algorithm [10]. The signed DCJ and the block interchange distances also benefit from efficient exact algorithms [124, 103]. On the other hand, the unsigned reversal and unsigned DCJ distances are NP-hard to compute [42, 43]. Two important operations remained: transpositions, of which the apparent simplicity is misleading, and prefix reversals, for which the same combinatorial problem had been raised in 1975 under the name of *pancake flipping* [64]. In Chapters 1 and 2, we settle the complexity of computing either the transposition or the unsigned prefix reversal distances (Sorting by Transpositions and Sorting by Prefix Reversals problems) by proving that both are NP-hard.

Part II – Distances Between Strings

In the second part, we lift the unicity constraint and consider models where genes can have several copies in each genome (they have several paralogs). Then, a genome is no longer represented as a permutation, but rather as a general sequence or *string*). Indeed, duplication events can occur within the genomes yielding indistinguishable genes. If a gene appears in several copies within two species, two main scenarios are possible. In the first, the duplications occurred before the speciation, i.e. the common ancestor contained all the copies of the gene. In this case, it is of interest to match the different copies across the two species. For example, if the two genomes considered are $G_1 = axbcxd$ and $G_2 = cxdaxb$, it is likely that the common ancestor contained both axb and cxd as substrings, and thus the first copy of x in G_1 should be matched to the second copy in G_2, and the second copy in G_1 should be matched to the first copy in G_2. This scenario is called the *Matching model* [25]. The opposite scenario, the *Exemplar model* [115], corresponds to the case where all duplications have taken place after the speciation event, entailing that the common ancestor only contains one exemplar of each gene. In this case, we aim at identifying, for each gene in the given genomes, which copy originates from the common ancestor and which copies have been created by duplications. Obviously, all intermediate cases can also appear between these two extreme scenarios, where some genes have several copies in the common ancestor and others have not, or one gene is duplicated in the common ancestor and undergoes additional duplications after the speciation. This approach corresponds to the *Intermediate model* [8].

For any of these models, the typical problems aim at computing a matching optimizing a given distance. However, whatever the model or the distance chosen, hardness results are ever-present and very few constant-ratio approximation algorithms exist (see e.g. [24, 7]). We present, in Chapter 3, several inapproximability results for problems under the exemplar model, in order to have an exhaustive panorama of hardness results for this model. More precisely, we prove that even when the input is

restricted to 1) a first genome without duplications, and 2) a second genome where the genes have at most two occurrences, the Hamming, Levenshtein, reversal and DCJ distances are not approximable within given ratios.

In Chapter 4, we consider the Minimum Common String Partition problem (MCSP). Presented as a string processing problem, it aims at partitioning two strings into a minimum-size common set of blocks. It corresponds to computing the breakpoint distance under the matching model, with the constraint that each gene must have the same number of copies in both genomes. Despite its apparent simplicity (one "simply" has to recover a few common factors between two strings), this problem is APX-hard and no constant-ratio approximation algorithm is known for it. We approach this problem under the fixed-parameter tractability theory, and provide a bounded search-tree FPT algorithm solving MCSP, where the parameter is the number of blocks in an optimal solution. Comparing with previous FPT algorithms [61, 91], which need both the number of blocks and the number of repetitions as a parameter, we prove that the intractability can be confined to the number of blocks, even for highly repetitive strings.

Part III – Dealing with Imprecise Genomic Data

In the third part, we consider models where the knowledge of the gene sequence is not perfect. Indeed, each step in the construction of input sequences, from DNA sequencing to gene annotation and genome reconstruction, may rise uncertainties or even introduce errors in the sequence. Generally speaking, all genomic data should thus be considered with the possibilities of error and incompleteness. However, although one would like to be able to deal with such data when solving any comparative genomics problem, very few problems allow for general enough models. On the other hand, it is sometimes possible to sanitize the data as a preprocessing stage, by e.g. removing what are most likely errors, before running other algorithms which require error-free and complete data. See e.g. the discussion in Sankoff et al. [117, Section 5].

Removing likely errors is precisely the goal of Maximal Strip Recovery (MSR, [128]). For this problem, we consider the genomes of two species as closely related as possible. There, one can assume that virtually all genes should belong to larger synteny blocks, and, when dealing with "clean" data, it should be possible to identify synteny blocks covering the whole genomes (each synteny block containing at least two genes). The objective of MSR is to mark a minimum number of genes as errors until the remaining genomes have this "clean" property. There are several variants of this problem: one can consider in input more than two genomes (MSR-d, where $d \geq 3$) or genomes with duplicated elements (MSR-DU). We also introduce the δ-gap variant ($\delta \geq 0$): if despite the closeness constraint, a gene does not belong to any synteny block, then it can only appear between two different blocks, not within one. Thus, the markers separating two consecutive elements of an output synteny block should only correspond to actual errors, hence there should be few of them. Hence, two consecutive elements of the output can only be separated by at most δ error-marked genes in the input genomes. Finally, for approximation and fixed-parameter tractability point of views, one can consider either the problem of keeping a maximum number ℓ of genes (MSR) or the complement problem of removing a minimum number k of genes (CMSR). In Chapter 5, we study extensively the variants of MSR and CMSR, by providing NP- and APX-hardness results, approximation algorithms and fixed-parameter tractable algorithms. The

main results in this chapter are the NP-hardness of 1-gap-MSR, and for CMSR-d and δ-gap-CMSR-d, a $(d + 1.5)$-approximation algorithm and an $O^*(2.36^k)$ FPT algorithm.

In Chapter 6, we study the Minimum Breakpoint Linearization (MBL, [127, 23, 73]) problem which aims at recovering the sequence of genes in a chromosome whose gene order is only partially known, using as a reference the genome of a close species. Formally, given a partial order and a total order over a common set of elements, find a *linearization* of the partial order which minimizes the breakpoint distance to the total order, where a linearization is a total order (or permutation) that is compatible with the partial order. We propose the construction of a graph structure (the adjacency-order graph) which allows to reduce the MBL problem to a more general problem of graph theory: Subset Feedback Vertex Set [67]. Literature results for this problem allow us to directly obtain an FPT and two approximation algorithms for MBL (the approximation algorithms having non-constant, incomparable ratios). Finally, using the knowledge of how input orders are generated, we design a third approximation algorithm specific to our problem.

General Preliminaries

Notations

In the majority of the problems, an input genome is represented as a *sequence* of integers, or *markers*, written $S = \langle s_1, s_2, \ldots, s_n \rangle$. The ith marker of S is written $S[i]$ (here, $S[i] = s_i$). Given integers a, b, the *interval* $[\![a\,;\,b]\!]$ is the set $\{a, a+1, \ldots, b\}$ if $a \leq b$ and the empty set if $b < a$. A *permutation* of size n is a sequence of n distinct elements over $[\![1\,;\,n]\!]$, and the *Identity* of size n, written $\mathcal{I}_n$, is the permutation $\langle 1, 2, \ldots, n \rangle$. By opposition to permutations, *strings* are sequences that can (and are expected to) contain duplicated elements (*duplicates*).

A *subsequence* S' of S is a sequence $\langle S[i_1], \ldots, S[i_h] \rangle$ of markers of S with increasing indices $i_1 < i_2 < \ldots < i_h$. A *factor* (or *substring*) of S is a subsequence of consecutive elements ($i_{k+1} = i_k + 1$ for $1 \leq k < h$).

L-reduction

The APX-hardness results (presented in Chapters 3 and 5) rely on the notion of L-reduction [108], defined below.

Given $\mathcal{P}$ an optimization problem, x an instance of $\mathcal{P}$ and y a feasible solution of x, we write $c_{\mathcal{P}}(x, y)$ for the cost of y and $\mathsf{OPT}_{\mathcal{P}}(x)$ for the optimal value of $c_{\mathcal{P}}(x, y)$ over all solutions y of x. Let $\mathcal{P}$ and $\mathcal{Q}$ be two optimization problems. An L-reduction from $\mathcal{P}$ to $\mathcal{Q}$ is a pair of polynomial-time computable functions f and g such that:

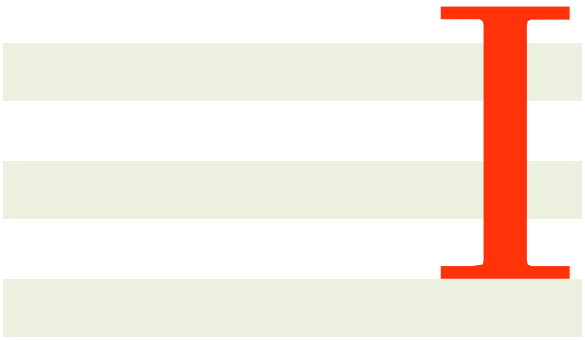
- if x is an instance of $\mathcal{P}$, then $f(x)$ is an instance of $\mathsf{OPT}_{\mathcal{Q}}$,
- if y is a solution of $f(x)$ for some x, then $g(y)$ is a solution of x,
- there exists a positive constant α such that

$$\mathsf{OPT}_{\mathcal{Q}}(f(x)) \leq \alpha \cdot \mathsf{OPT}_{P}(x),$$

- there exists a positive constant β such that

$$|\mathsf{OPT}_{\mathcal{P}}(x) - c_{\mathcal{P}}(x, g(y))| \leq \beta \cdot |\mathsf{OPT}_{\mathcal{Q}}(f(x)) - c_{\mathcal{P}}(f(x), y)|.$$

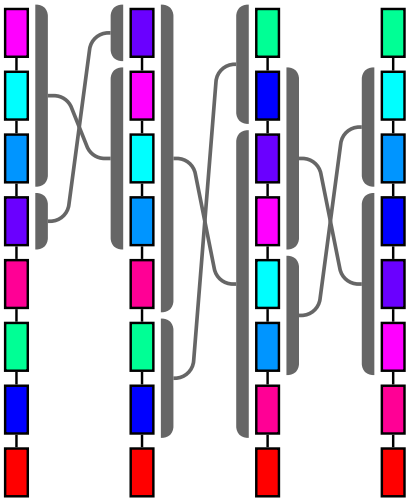
Given such an L-reduction from $\mathcal{P}$ to $\mathcal{Q}$, if $\mathcal{P}$ is NP-hard to approximate within $1 + \delta$, then $\mathcal{Q}$ is NP-hard to approximate within $1 + \delta/(\alpha\beta)$.



Distances Between Permutations

1

Sorting by Transpositions



The objective of the Sorting by Transpositions problem is to compute the minimum number of transpositions needed to transform one genome (represented as a permutation) into another, where a transposition is the operation that exchanges two consecutive sequences in the permutation.

In this chapter, we prove that this problem, and a related bound-checking problem, are NP-hard.

The results in this chapter have been presented at the *Algorithmique, combinatoire du texte et applications en bio-informatique* workshop (Seqbio 2011, Rennes), at the *38th International Colloquium on Automata, Languages and Programming* (ICALP 2011, Zürich [33]), and have been published in *SIAM Journal of Discrete Mathematics* (SIDMA 2012 [35]).

Introduction

Along with reversals, transpositions are one of the elementary large-scale operations that can affect a genome. A transposition consists in swapping two consecutive factors of genes or, equivalently, in moving a factor from one place to another in the genome. The transposition distance between two genomes is the minimum number of such operations that are needed to transform one genome into the other. Computing this distance is a challenge in comparative genomics, see for instance [110], since it gives a maximum parsimony evolution scenario between the two studied genomes.

The Sorting by Transpositions problem is the problem of computing the transposition distance between genomes represented by permutations: see [69] for a detailed review on this problem and its variants. Since its introduction by Bafna and Pevzner [12, 14], a number of studies [14, 55, 79, 85, 65, 16, 68] have aimed at designing approximation algorithms or heuristics; the best known fixed-ratio algorithm being a 1.375-approximation [65]. Other works [80, 55, 66, 98, 65, 16] aim at computing bounds on the transposition distance of a permutation. Studies have also been devoted to variants of this problem, by considering, for example, sorting by prefix transpositions [62, 99, 51] (in which one of the transposed factors has to be a prefix of the sequence), or sorting by transpositions in strings [56, 60, 118, 113] (where multiple occurrences of each element are allowed in the sequences), possibly using weighted or prefix transpositions [111, 27, 6, 51, 5]. Note also that sorting a permutation by block-interchanges (i.e., exchanges of not necessarily consecutive factors) is solvable in polynomial time [54]. Finally, one can consider the possibility to bound the size of the factors transposed. The most drastic variant, authorizing only *3-bounded* transpositions (where at most three markers can be affected by each transposition) has already received a particular attention [104, 92]. However, its computational complexity is still open.

In this chapter, we address the long-standing issue of determining the complexity class of the Sorting by Transpositions problem, by giving a polynomial-time reduction from SAT, thus proving the NP-hardness of this problem. Our reduction is based on the study of transpositions that remove three breakpoints. A corollary of our result is the NP-hardness of the following problem, introduced in [55]: given a permutation π, is it possible to sort π using exactly $d_b(\pi)/3$ transpositions, where $d_b(\pi)$ is the number of breakpoints of π? The first section provides some preliminary definitions and results about the transposition distance and its relationship with breakpoints. In the second section, we introduce an intermediate structure used in our reduction, the 3DT-instance, and provide results giving some equivalences between 3DT-instances and permutations. The third section is the core of the reduction; it is devoted to the construction of "NP-hard" 3DT-instances, using an intricate assembling of basic blocks. Finally, the fourth section concludes the reduction by porting the previous construction to permutations, thus proving the NP-hardness of the Sorting By Transpositions problem.

$$\begin{array}{rcl} \pi & = & \langle \pi_0\pi_1 \dots \pi_{i-1}\underline{\pi_i \dots \pi_{j-1}}\ \underline{\pi_j \quad \dots \quad \pi_{k-1}}\pi_k \dots \pi_n \rangle \\ \pi \circ \tau_{i,j,k} & = & \langle \pi_0\pi_1 \dots \pi_{i-1}\underline{\pi_j \quad \dots \quad \pi_{k-1}}\ \underline{\pi_i \dots \pi_{j-1}}\pi_k \dots \pi_n \rangle \end{array}$$

Figure 1.1: Transposition $\tau_{i,j,k}$, with $0 < i < j < k \leq n$.

1.1 Preliminaries

1.1.1 Transpositions and Breakpoints

In this chapter, we are given a positive integer n, and we consider only permutations of $[\![0\,;\,n]\!]$ such that 0 and n are fixed-points. Hence, $\mathcal{I}_n$ denotes the identity permutation over $[\![0\,;\,n]\!]$.

A transposition is an operation that exchanges two consecutive factors of a sequence. As we only work with permutations, it is defined as a permutation $\tau_{i,j,k}$, which, once composed with a permutation π, realizes this operation (see Figure 1.1). The transposition $\tau_{i,j,k}$ is formally defined as follows.

Definition 1.1 (Transposition)**.** *Given three integers i, j, k such that $0 < i < j < k \leq n$, the* transposition $\tau_{i,j,k}$ *over $[\![0\,;\,n]\!]$ is the following permutation (we write $q(j) = k + i - j$):*

$$\begin{array}{ll} \textit{For any } 0 \leq x < i, & \tau_{i,j,k}(x) = x \\ \textit{For any } i \leq x < q(j), & \tau_{i,j,k}(x) = x + j - i \\ \textit{For any } q(j) \leq x < k, & \tau_{i,j,k}(x) = x + j - k \\ \textit{For any } k \leq x \leq n, & \tau_{i,j,k}(x) = x \end{array}$$

Note that the inverse function of $\tau_{i,j,k}$ is also a transposition. More precisely, we have $\tau_{i,j,k}^{-1} = \tau_{i,q(j),k}$. The following two properties directly follow from the definition of a transposition:

Property 1.1. *Let $\tau = \tau_{i,j,k}$ be a transposition, $q(j) = k + i - j$, and $u, v \in [\![0\,;\,n]\!]$ be two integers such that $u < v$. Then:*

$$\begin{array}{rcl} \tau(u) > \tau(v) & \Leftrightarrow & i \leq u < q(j) \leq v < k \\ \tau^{-1}(u) > \tau^{-1}(v) & \Leftrightarrow & i \leq u < j \leq v < k \end{array}$$

Property 1.2. *Let τ be the transposition $\tau = \tau_{i,j,k}$, and write $q(j) = k + i - j$. For all $x \in [\![1\,;\,n]\!]$, the values of $\tau(x-1)$ and $\tau^{-1}(x-1)$ are the following:*

$$\begin{array}{rcl} \forall x \notin \{i, q(j), k\}, \quad \tau(x-1) & = & \tau(x) - 1 \\ \forall x \notin \{i, j, k\}, \quad \tau^{-1}(x-1) & = & \tau^{-1}(x) - 1 \end{array}$$

$$\begin{array}{rclrcl} \tau(i-1) & = & \tau(q(j)) - 1 & \tau^{-1}(i-1) & = & \tau^{-1}(j) - 1 \\ \tau(q(j)-1) & = & \tau(k) - 1 & \tau^{-1}(j-1) & = & \tau^{-1}(k) - 1 \\ \tau(k-1) & = & \tau(i) - 1 & \tau^{-1}(k-1) & = & \tau^{-1}(i) - 1 \end{array}$$

$$\begin{array}{lcl} \pi & = & \langle 0, \underline{2}, 4, \underline{3}, \underline{1}, 5 \rangle \\ \pi \circ \tau_{1,3,5} & = & \langle 0, \underline{3}, \underline{1}, \underline{2}, 4, 5 \rangle \\ \pi \circ \tau_{1,3,5} \circ \tau_{1,2,4} & = & \langle 0, 1, 2, 3, 4, 5 \rangle \end{array}$$

Figure 1.2: The transposition distance from $\pi = \langle 0, 2, 4, 3, 1, 5 \rangle$ to $\mathcal{I}_5$ is 2: it is at most 2 since $\pi \circ \tau_{1,3,5} \circ \tau_{1,2,4} = \mathcal{I}_5$, and it cannot be less than 2 since Property 1.4 applies with $d_b(\pi)/3 = 5/3 > 1$.

Definition 1.2 (Breakpoints)**.** *Let π be a permutation of $[\![0\,;n]\!]$. If $x \in [\![1\,;n]\!]$ is an integer such that $\pi(x-1) = \pi(x) - 1$, then $\langle x-1, x \rangle$ is* an adjacency *of π, otherwise it is* a breakpoint*. We write $d_b(\pi)$ for the number of breakpoints of π.*

The following property yields that the number of breakpoints of a permutation can be reduced by at most 3 when a transposition is applied:

Property 1.3. *Let π be a permutation and $\tau = \tau_{i,j,k}$ be a transposition (with $0 < i < j < k \leq n$). Then, for all $x \in [\![1\,;n]\!] - \{i, j, k\}$,*

$$\langle x-1, x \rangle \text{ is an adjacency of } \pi \Leftrightarrow \langle \tau^{-1}(x) - 1, \tau^{-1}(x) \rangle \text{ is an adjacency of } \pi \circ \tau.$$

Overall, we have $d_b(\pi \circ \tau) \geq d_b(\pi) - 3$.

Proof. For all $x \in [\![1\,;n]\!] - \{i, j, k\}$, we have:

$$\begin{aligned} \langle x-1, x \rangle \text{ is an adjacency of } \pi &\Leftrightarrow \pi(x-1) = \pi(x) - 1 \\ &\Leftrightarrow \pi(\tau(\tau^{-1}(x-1))) = \pi(\tau(\tau^{-1}(x))) - 1 \\ &\Leftrightarrow \pi \circ \tau(\tau^{-1}(x) - 1) = \pi \circ \tau(\tau^{-1}(x)) - 1 \text{ by Prop. 1.2} \\ &\Leftrightarrow \langle \tau^{-1}(x) - 1, \tau^{-1}(x) \rangle \text{ is an adjacency of } \pi \circ \tau. \end{aligned}$$

□

1.1.2 Transposition Distance

The transposition distance of a permutation is the minimum number of transpositions needed to transform it into the identity. A formal definition is the following:

Definition 1.3 (Transposition distance)**.** *Let π be a permutation of $[\![0\,;n]\!]$. The* transposition distance *$d_t(\pi)$ from π to $\mathcal{I}_n$ is the minimum value k for which there exist k transpositions $\tau_1, \tau_2, \ldots, \tau_k$, satisfying:*

$$\pi \circ \tau_k \circ \ldots \circ \tau_2 \circ \tau_1 = \mathcal{I}_n$$

The decision problem of computing the transposition distance is the following:

Problem	Sorting by Transpositions [12]
Input	A permutation π, an integer k
Question	Is $d_t(\pi) \leq k$?

The following property directly follows from Property 1.3, since for any n the number of breakpoints of $\mathcal{I}_n$ is 0.

Property 1.4. *Let π be a permutation, then $d_t(\pi) \geq d_b(\pi)/3$.*

Figure 1.2 gives an example of the computation of the transposition distance.

1.2 3-Deletion and Transposition Operations

In this section, we introduce 3DT-instances, which are the cornerstone of our reduction from SAT to the Sorting by Transpositions problem, since they are used as an intermediate between instances of those two problems. We first define 3DT-instances and the possible operations that can be applied to them, then we focus on the equivalence between these instances and permutations. Section 1.2.4 stands out of the flow of the reduction, and gives a way to match 3DT-instances with the cycle graphs of 3-permutations. It should help the readers who are already familiar with the notion of cycle graphs following the definitions introduced here.

1.2.1 3DT-instances

We introduce 3DT-instances as stand-alone mathematical objects, and give a set of tools to handle them. The two main objectives are (1) to accurately reflect the behavior of the breakpoints in a permutation π such that $d_t(\pi) = d_b(\pi)/3$ (this aspect will be formalized in the following sections) and (2) to make 3DT-instances easy to construct (they can be defined as a word whose letters can be partitioned into triples).

Definition 1.4 (3DT-instance)**.** *A* 3DT-instance $I = \langle \Sigma, T, \psi \rangle$ *of span* n *is composed of the following elements:*

- Σ*: an alphabet;*
- $T = \{(a_i, b_i, c_i) \mid 1 \leq i \leq |T|\}$*: a set of (ordered) triples of elements of* Σ*, partitioning* Σ *(i.e., all elements are pairwise distinct, and* $\bigcup_{i=1}^{|T|} \{a_i, b_i, c_i\} = \Sigma$*);*
- $\psi : \Sigma \to [\![1\,;\,n]\!]$*, an injection.*

The domain *of* I *is the image of* ψ*, that is the set* $L = \{\psi(\sigma) \mid \sigma \in \Sigma\}$*.*

The word representation *of* I *is the* n*-letter word* $u_1\ u_2 \dots u_n$ *over* $\Sigma \cup \{\centerdot\}$ *(where* $\centerdot \notin \Sigma$*), such that for all* $i \in L$*,* $\psi(u_i) = i$*, and for* $i \in [\![1\,;\,n]\!] - L$*,* $u_i = \centerdot$*.*

Two examples of 3DT-instances are given in Example 1.1. Note that such instances can be defined by their word representation and by their set of triples T. The word representation of the empty 3DT-instance (in which $\Sigma = \emptyset$) is a sequence of n dot. We may also denote this instance by ε.

Example 1.1. *In this example, we define two 3DT-instances of span 6,* $I = \langle \Sigma, T, \psi \rangle$ *and* $I' = \langle \Sigma', T', \psi' \rangle$*:*

$$\begin{array}{lcll} I & = & a_1\ c_2\ b_1\ b_2\ c_1\ a_2 & \text{with } T = \{(a_1, b_1, c_1), (a_2, b_2, c_2)\} \\ I' & = & \centerdot\ b_2\ \centerdot\ c_2\ \centerdot\ a_2 & \text{with } T' = \{(a_2, b_2, c_2)\} \end{array}$$

Here, I *has an alphabet of size 6,* $\Sigma = \{a_1, b_1, c_1, a_2, b_2, c_2\}$*, hence* ψ *is a bijection* ($\psi(a_1) = 1$*,* $\psi(c_2) = 2$*,* $\psi(b_1) = 3$*, etc). The second instance,* I'*, has an alphabet of size 3,* $\Sigma' = \{a_2, b_2, c_2\}$*, with* $\psi'(b_2) = 2$*,* $\psi'(c_2) = 4$*,* $\psi'(a_2) = 6$*.*

Property 1.5. *Let* $I = \langle \Sigma, T, \psi \rangle$ *be a 3DT-instance of span* n *with domain* L*. Then*

$$|\Sigma| = |L| = 3\,|T| \leq n.$$

Proof. We have $|\Sigma| = |L|$ since ψ is an injection with image L. The triples of T partition Σ so $|\Sigma| = 3|T|$, and finally $L \subseteq [\![1\,;\,n]\!]$ so $|L| \leq n$. □

Definition 1.5. *Let $I = \langle \Sigma, T, \psi \rangle$ be a 3DT-instance. The injection ψ gives a total order over Σ, written $\prec_I$ (or $\prec$, if there is no ambiguity), defined by*

$$\forall \sigma_1, \sigma_2 \in \Sigma, \quad \sigma_1 \prec_I \sigma_2 \Leftrightarrow \psi(\sigma_1) < \psi(\sigma_2) \tag{1.1}$$

Two elements σ_1 and σ_2 of Σ are called consecutive *if there exists no element $x \in \Sigma$ such that $\sigma_1 \prec_I x \prec_I \sigma_2$. In this case, we write $\sigma_1 \lhd_I \sigma_2$ (or simply $\sigma_1 \lhd \sigma_2$).*

An equivalent definition is that $\sigma_1 \prec \sigma_2$ if $\sigma_1\ \sigma_2$ is a subsequence of the word representation of I. Also, $\sigma_1 \lhd \sigma_2$ if the word representation of I contains a factor of the kind $\sigma_1 \blacksquare^* \sigma_2$ (where $\blacksquare^*$ represents any sequence of $l \geq 0$ dots).

Using the triples in T, we define a successor function over the domain L:

Definition 1.6. *Let $I = \langle \Sigma, T, \psi \rangle$ be a 3DT-instance with domain L. The function $succ_I : L \to L$ is defined by:*

$$\begin{aligned} \forall (a,b,c) \in T, \quad & \psi(a) \mapsto \psi(b) \\ & \psi(b) \mapsto \psi(c) \\ & \psi(c) \mapsto \psi(a) \end{aligned}$$

Function $succ_I$ is a bijection, with no fixed-points, and such that $succ_I \circ succ_I \circ succ_I$ is the identity over L. In Example 1.1, we have:

$$succ_I = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 6 & 5 & 2 & 1 & 4 \end{pmatrix} \text{ and } succ_{I'} = \begin{pmatrix} 2 & 4 & 6 \\ 4 & 6 & 2 \end{pmatrix}.$$

1.2.2 3DT-steps

We define 3DT-steps on 3DT-instances in order to reflect transpositions removing three breakpoints on permutations. In both points of view (3DT-instances and permutations), we apply a transposition for which the bounds need to fulfill certain conditions (being a well-ordered triple for 3DT-instances, or satisfying a correct pattern of values for permutations).

Definition 1.7. *Let $I = \langle \Sigma, T, \psi \rangle$ be a 3DT-instance, and (a,b,c) be a triple of T. Write $i = \min\{\psi(a), \psi(b), \psi(c)\}$, $j = succ_I(i)$, and $k = succ_I(j)$. The triple $(a,b,c) \in T$ is* well-ordered *if we have $i < j < k$. In such a case, we write $\tau[a,b,c,\psi]$ for the transposition $\tau_{i,j,k}$.*

An equivalent definition is that (a,b,c) is well-ordered iff one of abc, bca, cab is a subsequence of the word representation of I. In Example 1.1, (a_1,b_1,c_1) is well-ordered in I: indeed, we have $i = \psi(a_1)$, $j = \psi(b_1)$, $k = \psi(c_1)$, and $i < j < k$. The triple (a_2,b_2,c_2) is also well-ordered in I' ($i = \psi'(b_2) < j = \psi'(c_2) < k = \psi'(a_2)$), but not in I: $i = \psi(c_2) < k = \psi(b_2) < j = \psi(a_2)$. In this example, we have $\tau[a_1,b_1,c_1,\psi] = \tau_{1,3,5}$ and $\tau[a_2,b_2,c_2,\psi'] = \tau_{2,4,6}$.

Definition 1.8 (3DT-step)**.** *Let $I = \langle \Sigma, T, \psi \rangle$ be a 3DT-instance with $(a,b,c) \in T$ a well-ordered triple. The* 3DT-step *of parameter (a,b,c) is the operation written $\xrightarrow{(a,b,c)}$, transforming I into the 3DT-instance $I' = \langle \Sigma', T', \psi' \rangle$ such that:*

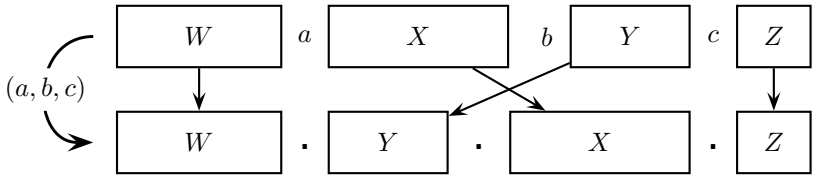


Figure 1.3: The 3DT-step $\xrightarrow{(a,b,c)}$ has two effects, here represented on the word representation of a 3DT-instance: the triple (a, b, c) is deleted (and replaced by dots in this word representation), and the factors X and Y are swapped.

- $\Sigma' = \Sigma - \{a, b, c\}$
- $T' = T - \{(a, b, c)\}$
- $\psi' : \begin{array}{ccc} \Sigma' & \to & [\![1\,;\,n]\!] \\ \sigma & \mapsto & \tau^{-1}(\psi(\sigma)) \end{array}$ $(with\ \tau = \tau[a, b, c, \psi]).$

A 3DT-step has two effects on a 3DT-instance, as represented in Figure 1.3. The first is to remove a necessarily well-ordered triple from T (hence from Σ). The second is, by applying a transposition to ψ, to shift the position of some of the remaining elements. Note that a triple that is not well-ordered in I can become well-ordered in I', or vice-versa. In Example 1.1, I' can be obtained from I via a 3DT-step: $I \xrightarrow{(a_1,b_1,c_1)} I'$. Moreover, $I' \xrightarrow{(a_2,b_2,c_2)} \varepsilon$. A more complex example is given in Figure 1.4.

Note that a 3DT-step transforms the function $succ_I$ into $succ_{I'} = \tau^{-1} \circ succ_I \circ \tau$, restricted to L', the domain of the new instance I'. Indeed, for all $(a, b, c) \in T'$, we have

$$\begin{aligned} succ_{I'}(\psi'(a)) &= \psi'(b) \\ &= \tau^{-1}(\psi(b)) \\ &= \tau^{-1}(succ_I(\psi(a))) \\ &= \tau^{-1}(succ_I(\tau(\psi'(a)))) \\ &= (\tau^{-1} \circ succ_I \circ \tau)(\psi'(a)) \end{aligned}$$

The computation is similar for $\psi'(b)$ and $\psi'(c)$.

Definition 1.9 (3DT-collapsibility)**.** *A 3DT-instance $I = \langle \Sigma, T, \psi \rangle$ is said to be* 3DT-collapsible *if there exists a sequence of 3DT-instances $I_k, I_{k-1}, \ldots, I_0$ such that*

$$\begin{aligned} &I_k = I \\ &\forall i \in [\![1\,;\,k]\!], \quad \exists (a, b, c) \in T, \quad I_i \xrightarrow{(a,b,c)} I_{i-1} \\ &I_0 = \varepsilon \end{aligned}$$

In Example 1.1, I and I' are 3DT-collapsible, since $I \xrightarrow{(a_1,b_1,c_1)} I' \xrightarrow{(a_2,b_2,c_2)} \varepsilon$. Another example is the 3DT-instance defined in Figure 1.4. Note that in the example of Figure 1.4, there are in fact two distinct paths leading to the empty instance.

1.2.3 Equivalence with the Transposition Distance

Definition 1.10. *Let $I = \langle \Sigma, T, \psi \rangle$ be a 3DT-instance of span n with domain L, and π be a permutation of $[\![0\,;\,n]\!]$. We say that I and π are equivalent, and we write*

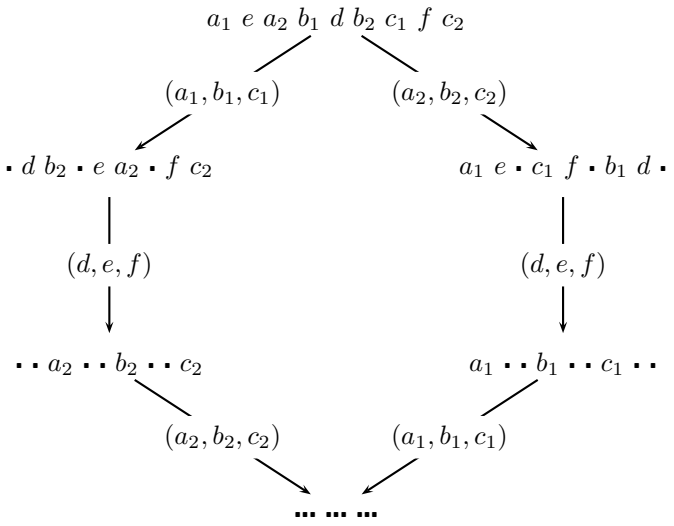


Figure 1.4: Possible 3DT-steps from the instance I defined by the word $a_1\ e\ a_2\ b_1\ d\ b_2\ c_1\ f\ c_2$ and the set of triples $T = \{(a_1, b_1, c_1), (a_2, b_2, c_2), (d, e, f)\}$. We can see that there is a path from I to ε, hence I is 3DT-collapsible. Note that both (a_1, b_1, c_1) and (a_2, b_2, c_2) are well-ordered in the initial instance, each one loses this property after applying the 3DT-step associated to the other, and becomes well-ordered again after applying the 3DT-step associated to (d, e, f).

$I \sim \pi$, if:

$$\begin{aligned} \pi(0) &= 0, \\ \forall v \in [\![1\,;\,n]\!] - L, \pi(v) &= \pi(v-1) + 1, \\ \forall v \in L, \quad \pi(v) &= \pi(succ_I^{-1}(v) - 1) + 1. \end{aligned}$$

With such an equivalence $I \sim \pi$, the two following properties hold:
- The breakpoints of π correspond to the elements of L (see Property 1.6).
- The triples of breakpoints that may be resolved immediately by a single transposition correspond to the well-ordered triples of T (see Figure 1.5 and Lemma 1.8).

Property 1.6. *Let $I = \langle \Sigma, T, \psi \rangle$ be a 3DT-instance of span n with domain L, and π be a permutation of $[\![0\,;\,n]\!]$, such that $I \sim \pi$. Then the number of breakpoints of π is $d_b(\pi) = |L| = 3|T|$.*

Proof. Let $v \in [\![1\,;\,n]\!]$. By Definition 1.10, we have:

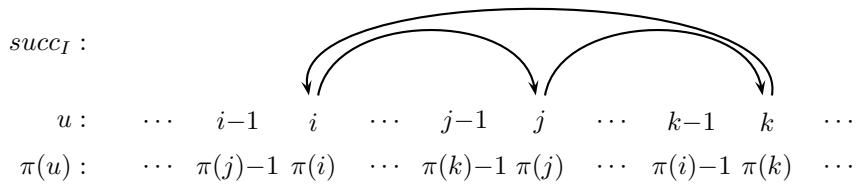


Figure 1.5: Illustration of the equivalence $I \sim \pi$ on three integers (i, j, k) such that $j = succ_I(i)$ and $k = succ_I(j)$. It can be checked that $\pi(v) = \pi(u-1) + 1$ for any $(u, v) \in \{(i, j), (j, k), (k, i)\}$.

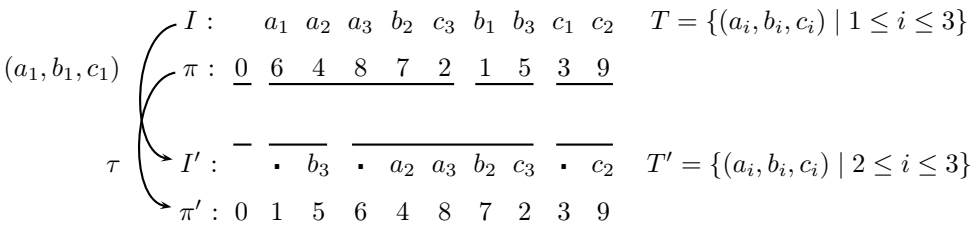


Figure 1.6: Illustration of Lemma 1.7: since $I \sim \pi$ and $I \xrightarrow{(a_1, b_1, c_1)} I'$, then $I' \sim \pi' = \pi \circ \tau$, where $\tau = \tau[a_1, b_1, c_1, \psi]$.

If $v \notin L$, then $\pi(v) = \pi(v-1) + 1$, so $\langle v-1, v \rangle$ is an adjacency of π.

If $v \in L$, we write $u = succ_I^{-1}(v)$, so $\pi(v) = \pi(u-1)+1$. Since $succ_I$ has no fixed-point, we have $u \neq v$, which implies $\pi(u-1) \neq \pi(v-1)$. Hence, $\pi(v) \neq \pi(v-1)+1$, and $\langle v-1, v \rangle$ is a breakpoint of π.

Consequently the number of breakpoints of π is exactly $|L|$, and $|L| = 3|T|$ by Property 1.5. □

With the following lemma, we show that the equivalence between a 3DT-instance and a permutation is preserved after a 3DT-step, see Figure 1.6.

Lemma 1.7. *Let $I = \langle \Sigma, T, \psi \rangle$ be a 3DT-instance of span n, and π be a permutation of $[\![0\,;\,n]\!]$, such that $I \sim \pi$. If there exists a 3DT-step $I \xrightarrow{(a,b,c)} I'$ for some well-ordered triple $(a, b, c) \in T$, then I' and $\pi' = \pi \circ \tau$, where $\tau = \tau[a, b, c, \psi]$, are equivalent.*

Proof. We write (i, j, k) for the indices such that $\tau = \tau_{i,j,k}$ (by definition, $i = \min\{\psi(a), \psi(b), \psi(c)\}$, $j = succ_I(i)$, $k = succ_I(j)$). Since (a, b, c) is well-ordered, we have $i < j < k$.

We have $I' = \langle \Sigma', T', \psi' \rangle$, with $\Sigma' = \Sigma - \{a, b, c\}$, $T' = T - \{(a, b, c)\}$, and $\psi' : \sigma \mapsto \tau^{-1}(\psi(\sigma))$. We write respectively L and L' for the domains of I and I'. For all $v' \in [\![1\,;\,n]\!]$, we have

$$\begin{aligned} v' \in L' &\Leftrightarrow \exists \sigma \in \Sigma - \{a, b, c\},\ v' = \tau^{-1}(\psi(\sigma)) \\ &\Leftrightarrow \tau(v') \in L - \{i, j, k\} \end{aligned}$$

We prove the 3 required properties (see Definition 1.10) sequentially:

- $\pi'(0) = \pi(\tau(0)) = \pi(0) = 0$,
- $\forall v' \in [\![1\,;\,n]\!] - L'$, let $v = \tau(v')$. Since $v' \notin L'$, we have either $v \in \{i, j, k\}$, or $v \notin L$. In the first case, we write $u = succ_I^{-1}(v)$ (then $u \in \{i, j, k\}$). By Property 1.2, $\tau^{-1}(u-1)$ is equal to $\tau^{-1}(succ_I(u))-1$, so $\tau^{-1}(u-1) = \tau^{-1}(v)-1$. Hence,

$$\begin{aligned} \pi'(v'-1)+1 &= \pi(\tau(\tau^{-1}(v)-1))+1 \\ &= \pi(u-1)+1 \\ &= \pi(v) \text{ by Def. 1.10, since } v \in L \text{ and } v = succ_I(u) \\ &= \pi'(v') \end{aligned}$$

In the second case, $v \notin L$, we have

$$\begin{aligned} \pi'(v'-1)+1 &= \pi(\tau(\tau^{-1}(v)-1))+1 \\ &= \pi(\tau(\tau^{-1}(v-1)))+1 \text{ by Prop. 1.2, since } v \notin \{i, j, k\} \end{aligned}$$

$$\begin{aligned}
&= \pi(v-1)+1 \\
&= \pi(v) \text{ by Def. 1.10, since } v \notin L \\
&= \pi'(v')
\end{aligned}$$

In both cases, we indeed have $\pi'(v'-1)+1 = \pi'(v')$.

– Let v' be an element of L'. We write $v = \tau(v')$, $u = succ_I^{-1}(v)$, and $u' = \tau^{-1}(u)$. Then $v' = \tau^{-1}(succ_I(\tau(u'))) = succ_{I'}(u')$. Moreover, $v \notin \{i,j,k\}$, hence $u \notin \{i,j,k\}$.

$$\begin{aligned}
\pi'(u'-1)+1 &= \pi(\tau(\tau^{-1}(u)-1))+1 \\
&= \pi(\tau(\tau^{-1}(u-1)))+1 \text{ by Prop. 1.2, since } u \notin \{i,j,k\} \\
&= \pi(u-1)+1 \\
&= \pi(v) \text{ by Def. 1.10, since } v \in L \text{ and } u = succ_I^{-1}(v) \\
&= \pi(\tau(\tau^{-1}(v))) \\
&= \pi'(v')
\end{aligned}$$

□

Lemma 1.8. *Let $I = \langle \Sigma, T, \psi \rangle$ be a 3DT-instance of span n, and π a permutation of $[\![0\,;\,n]\!]$, such that $I \sim \pi$. If there exists a transposition $\tau = \tau_{i,j,k}$ such that $d_b(\pi \circ \tau) = d_b(\pi) - 3$, then T contains a well-ordered triple (a,b,c) such that $\tau = \tau[a,b,c,\psi]$.*

Proof. We write $i' = \tau^{-1}(i)$, $j' = \tau^{-1}(j)$, and $k' = \tau^{-1}(k)$. Note also that $i < j < k$.

Let $\pi' = \pi \circ \tau$. For all $x \in [\![1\,;\,n]\!] - \{i,j,k\}$, we have, by Property 1.3, that $\langle x-1, x\rangle$ is an adjacency of π iff $\langle \tau^{-1}(x)-1, \tau^{-1}(x)\rangle$ is an adjacency of π'. Hence, since $d_b(\pi') = d_b(\pi) - 3$, we necessarily have that $\langle i-1, i\rangle$, $\langle j-1, j\rangle$ and $\langle k-1, k\rangle$ are breakpoints of π, and $\langle i'-1, i'\rangle$, $\langle j'-1, j'\rangle$ and $\langle k'-1, k'\rangle$ are adjacencies of π'. We have

$$\begin{aligned}
\pi(i) &= \pi(\tau(i')) \\
&= \pi'(i') \\
&= \pi'(i'-1)+1 \text{ since } \langle i'-1, i'\rangle \text{ is an adjacency of } \pi' \\
&= \pi'(\tau^{-1}(i)-1)+1 \\
&= \pi'(\tau^{-1}(k-1))+1 \text{ by Prop. 1.2} \\
&= \pi(k-1)+1
\end{aligned}$$

Since $I \sim \pi$ and $i \neq k$, by Definition 1.10, we necessarily have $i \in L$ (where L is the domain of I), and $i = succ_I(k)$.

Using the same method with $(j'-1, j')$ and $(k'-1, k')$, we obtain $j, k \in L$, $j = succ_I(i)$ and $k = succ_I(j)$. Hence, T contains one of the following three triples: $(\psi^{-1}(i), \psi^{-1}(j), \psi^{-1}(k))$, $(\psi^{-1}(j), \psi^{-1}(k), \psi^{-1}(i))$, or $(\psi^{-1}(k), \psi^{-1}(i), \psi^{-1}(j))$. Writing (a,b,c) this triple, we indeed have $\tau_{i,j,k} = \tau[a,b,c,\psi]$ since $i < j < k$. □

Theorem 1.9. *Let $I = \langle \Sigma, T, \psi \rangle$ be a 3DT-instance of span n with domain L, and π be a permutation of $[\![0\,;\,n]\!]$, such that $I \sim \pi$. Then I is 3DT-collapsible iff $d_t(\pi) = |T| = d_b(\pi)/3$.*

Proof. We prove the theorem by induction on $k = |T|$. For $k = 0$, necessarily $I = \varepsilon$ and $L = T = \emptyset$, and by Definition 1.10, $\pi = \mathcal{I}_n$ ($\pi(0) = 0$, and for all

$v > 0$, $\pi(v) = \pi(v-1)+1$). In this case, I is trivially 3DT-collapsible, and $d_t(\pi) = 0 = |T| = d_b(\pi)/3$.

Suppose now $k = k'+1$, with $k' \geq 0$, and the theorem is true for k'. By Property 1.6, we have $d_b(\pi) = 3k$, and by Property 1.4, $d_t(\pi) \geq 3k/3 = k$.

Assume first that I is 3DT-collapsible. Then there exist both a triple $(a,b,c) \in T$ and a 3DT-instance $I' = \langle \Sigma', T', \psi' \rangle$ such that $I \xrightarrow{(a,b,c)} I'$ and I' is 3DT-collapsible. Since $T' = T - \{(a,b,c)\}$, the size of T' is $k-1 = k'$. By Lemma 1.7, we have $I' \sim \pi' = \pi \circ \tau$, with $\tau = \tau[a,b,c,\psi]$. Using the induction hypothesis, we know that $d_t(\pi') = k'$. So the transposition distance from $\pi = \pi' \circ \tau^{-1}$ to the identity is at most, hence exactly, $k'+1 = k$.

Assume now that $d_t(\pi) = k$. We can decompose π into $\pi = \pi' \circ \tau^{-1}$, where τ is a transposition and π' a permutation such that $d_t(\pi') = k-1 = k'$. Since π has $3k$ breakpoints (Property 1.6), and $\pi' = \pi \circ \tau$ has at most $3k-3$ breakpoints (Property 1.4), τ necessarily removes 3 breakpoints, and we can use Lemma 1.8: there exists a 3DT-step $I \xrightarrow{(a,b,c)} I'$, where $(a,b,c) \in T$ is a well-ordered triple and $\tau = \tau[a,b,c,\psi]$. We can now use Lemma 1.7, which yields $I' \sim \pi' = \pi \circ \tau$. Using the induction hypothesis, we obtain that I' is 3DT-collapsible, hence I is also 3DT-collapsible. This concludes the proof of the theorem. □

The previous theorem gives a way to reduce the problem of deciding if a 3DT-instance is collapsible to the Sorting by Transpositions problem. However, it must be used carefully, since there exist 3DT-instances to which no permutation is equivalent (for example, $I = a_1\ a_2\ b_1\ b_2\ c_1\ c_2$ admits no permutation π of $[\![0\,;6]\!]$ such that $I \sim \pi$).

1.2.4 Parallel with the Cycle Graph

This section gives another point of view for the definition of 3DT-instances, in such a way that they (almost) match cycle graphs, as defined in [14]. It is given as a side remark and will not be used in the rest of the reduction, hence it is not described with an over-formal vocabulary, and we assume that the reader is already familiar with cycle graphs. This correspondence is summarized in Figure 1.7.

A 3DT-instance behaves similarly to a cycle graph containing only 1- and 3-cycles, that is, the cycle-graph of a so-called 3-permutation. Each $\sigma \in \Sigma$ corresponds to a reality arc in a 3-cycle (a triple corresponds to a 3-cycle), and each other position $i \in [\![1\,;n]\!]$ such that $u_i = \blacksquare$ corresponds to the reality arc of a 1-cycle. The $succ_I$ function corresponds to taking the next reality arc in a cycle. A triple is well-ordered if the corresponding cycle is oriented in the graph, and, for such triples, following a 3DT-step corresponds to breaking the 3-cycle into three 1-cycles (which may also change the orientation of some other cycles).

Given such a correspondence between a 3DT-instance I and the cycle graph G of a permutation π, we obtain that I is equivalent to π, and that I is 3DT-collapsible iff there exists a sequence of transpositions each breaking a 3-cycle of G (i.e., iff $d_t(\pi) = d_b(\pi)/3$).

However, 3DT-instances are more general than cycle graphs: cycle graphs are defined from the permutations they represent while on the other hand 3DT-instances can be constructed without referring to a permutation, hence there is no cycle graph corresponding to certain 3DT-instances (e.g., $I = a_1\ a_2\ b_1\ b_2\ c_1\ c_2$). Consequently, we can focus exclusively on the way triples are ordered and how they overlap – this

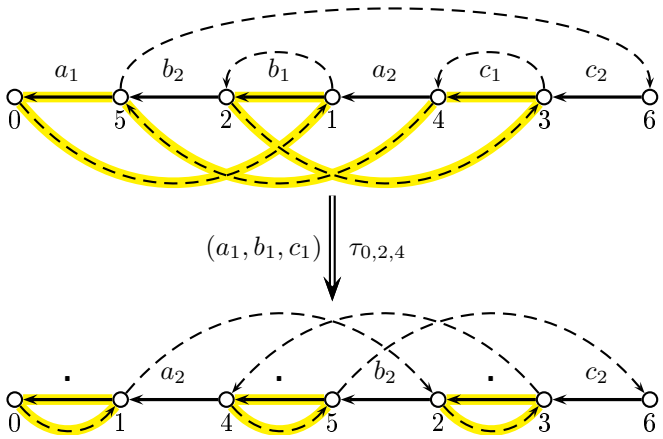


Figure 1.7: The 3DT-instance $a_1\ b_2\ b_1\ a_2\ c_1\ c_2$ seen on the corresponding cycle graph, i.e., the cycle graph of permutation $\langle 0, 5, 2, 1, 4, 3, 6\rangle$. The ordered triple (a_1, b_1, c_1) corresponds to the over-lined oriented 3-cycle, and following the 3DT-step removing this triple corresponds to breaking the cycle into 1-cycles.

task alone is the most complex one, see Section 1.3. The main drawback is that we need to verify, afterwards, that there indeed exist permutations which are equivalent to the 3DT-instances we build (Section 1.4).

1.3 3DT-collapsibility Is NP-Hard to Decide

In this section, we define, for any boolean formula ϕ, a corresponding 3DT-instance I_ϕ. We also prove that I_ϕ is 3DT-collapsible iff ϕ is satisfiable.

1.3.1 Block Structure

The construction of the 3DT-instance I_ϕ uses a decomposition into blocks, defined below. Some triples will be included in a block, in order to define its behavior, while others will be shared between two blocks, in order to pass information. The former are unconstrained, so that we can design blocks with the behavior we need (for example, blocks mimicking usual boolean functions), while the latter need to follow several rules, so that the blocks can conveniently be arranged together.

Definition 1.11 (l-block decomposition)**.** *An l-block decomposition $\mathcal{B}$ of a 3DT-instance I of span n is an l-tuple of integers $(s_1, \dots, s_l)$ such that $s_1 = 0$, $s_h < s_{h+1}$ for all $h \in [\![1\,;\, l-1]\!]$, and $s_l < n$. We write $t_h = s_{h+1}$ for $h \in [\![1\,;\, l-1]\!]$, and $t_l = n$.*

Let $I = \langle \Sigma, T, \psi\rangle$. For $h \in [\![1\,;\, l]\!]$, the factor $u_{s_h+1}\ u_{s_h+2}\ \dots u_{t_h}$ of the word representation $u_1\ u_2\ \dots u_n$ of I is called the full block $\mathcal{B}_h^{\blacksquare}$ *(it is a word over $\Sigma \cup \{\blacksquare\}$). The subsequence of $\mathcal{B}_h^{\blacksquare}$ where every occurrence of $\blacksquare$ is deleted is called the* block $\mathcal{B}_h$.

For $\sigma \in \Sigma$, we write $block_{I,\mathcal{B}}(\sigma) = h$ if $\psi(\sigma) \in [\![s_h + 1\,;\, t_h]\!]$ (equivalently, if σ appears in the word $\mathcal{B}_h$). A triple $(a, b, c) \in T$ is said to be internal *if $block_{I,\mathcal{B}}(a) = block_{I,\mathcal{B}}(b) = block_{I,\mathcal{B}}(c)$, and* external *otherwise.*

If τ is a transposition such that for all $h \in [\![1\,;\, l]\!]$, $\tau(s_h) < \tau(t_h)$, we write $\tau[\mathcal{B}]$ for the l-block decomposition $(\tau(s_1), \dots, \tau(s_l))$.

In the rest of this section, we mostly work with blocks instead of full blocks, since we are only interested in the relative order of the elements, rather than their actual position. Full blocks are only used in definitions, where we want to control the dots in the word representation of the 3DT-instances we define. Note that, for $\sigma_1, \sigma_2 \in \Sigma$ such that $block_{I,\mathcal{B}}(\sigma_1) = block_{I,\mathcal{B}}(\sigma_2) = h$, the relation $\sigma_1 \triangleleft \sigma_2$ is equivalent to $\sigma_1\ \sigma_2$ *is a factor of* $\mathcal{B}_h$.

Property 1.10. *Let $\mathcal{B} = (s_1, \dots, s_l)$ be an l-block decomposition of a 3DT-instance of span n, and $i, j, k \in [\![1\,;\,n]\!]$ be three integers such that (a) $i < j < k$ and (b) $\exists h_0$ such that $s_{h_0} < i < j \leq t_{h_0}$ or $s_{h_0} < j < k \leq t_{h_0}$ (or both). Then for all $h \in [\![1\,;\,l]\!]$, $\tau^{-1}_{i,j,k}(s_h) < \tau^{-1}_{i,j,k}(t_h)$, and the l-block decomposition $\tau^{-1}_{i,j,k}[\mathcal{B}]$ is defined.*

Proof. For any $h \in [\![1\,;\,l]\!]$, we show that we cannot have $i \leq s_h < j \leq t_h < k$. Indeed, $s_h < j$ implies $h \leq h_0$ (since $s_h < j \leq t_{h_0} = s_{h_0+1}$), and $j \leq t_h$ implies $h \geq h_0$ (since $t_{h_0-1} = s_{h_0} < j \leq t_h$). Hence $s_h < j \leq t_h$ implies $h = h_0$, but $i \leq s_h, t_h < k$ contradicts both conditions $s_{h_0} < i$ and $k \leq t_{h_0}$: hence the relation $i \leq s_h < j \leq t_h < k$ is impossible.

By Property 1.1, since $s_h < t_h$ for all $h \in [\![1\,;\,l]\!]$, and $i \leq s_h < j \leq t_h < k$ does not hold, we have $\tau^{-1}_{i,j,k}(s_h) < \tau^{-1}_{i,j,k}(t_h)$, which is sufficient to define $\tau^{-1}_{i,j,k}[\mathcal{B}]$. □

The above property yields that, if (a, b, c) is a well-ordered triple of a 3DT-instance $I = \langle \Sigma, T, \psi \rangle$ ($\tau = \tau[a, b, c, \psi]$), and $\mathcal{B}$ is an l-block decomposition of I, then $\tau^{-1}[\mathcal{B}]$ is defined if (a, b, c) is an internal triple, or an external triple such that one of the following equalities is satisfied: $block_{I,\mathcal{B}}(a) = block_{I,\mathcal{B}}(b)$, $block_{I,\mathcal{B}}(b) = block_{I,\mathcal{B}}(c)$ or $block_{I,\mathcal{B}}(c) = block_{I,\mathcal{B}}(a)$. In this case, the 3DT-step $I \xrightarrow{(a,b,c)} I'$ is written $(I, \mathcal{B}) \xrightarrow{(a,b,c)} (I', \mathcal{B}')$, where $\mathcal{B}' = \tau^{-1}[\mathcal{B}]$ is an l-block decomposition of I'.

Definition 1.12 (Variable)**.** *A* variable *A of a 3DT-instance $I = \langle \Sigma, T, \psi \rangle$ is a pair of triples $A = [(a, b, c), (x, y, z)]$ of T. It is* valid *in an l-block decomposition $\mathcal{B}$ if*

(i) *$\exists h_0 \in [\![1\,;\,l]\!]$ such that $block_{I,\mathcal{B}}(b) = block_{I,\mathcal{B}}(x) = block_{I,\mathcal{B}}(y) = h_0$*

(ii) *$\exists h_1 \in [\![1\,;\,l]\!]$, $h_1 \neq h_0$, such that $block_{I,\mathcal{B}}(a) = block_{I,\mathcal{B}}(c) = block_{I,\mathcal{B}}(z) = h_1$*

(iii) *if $x \prec y$, then we have $x \triangleleft b \triangleleft y$*

(iv) *$a \prec z \prec c$*

For such a valid variable A, the block $\mathcal{B}_{h_0}$ containing $\{b, x, y\}$ is called the source *of A (we write $source(A) = h_0$), and the block $\mathcal{B}_{h_1}$ containing $\{a, c, z\}$ is called the* target *of A (we write $target(A) = h_1$). For $h \in [\![1\,;\,l]\!]$, the variables of which $\mathcal{B}_h$ is the source (resp. the target) are called the* output *(resp. the* input*) of $\mathcal{B}_h$. The 3DT-step $I \xrightarrow{(x,y,z)} I'$ is called the* activation *of A (it requires that (x, y, z) is well-ordered).*

Note that since a valid variable $A = [(a, b, c), (x, y, z)]$ satisfies $block_{I,\mathcal{B}}(x) = block_{I,\mathcal{B}}(y)$, its activation can be written $(I, \mathcal{B}) \xrightarrow{(x,y,z)} (I', \mathcal{B}')$.

Example 1.2. *Consider the 3DT-instance I with the 2-block decomposition $\mathcal{B}$ such that:*

$$I = d\ y\ e\ x\ b\ f\ d'\ a\ f'\ z\ e'\ c$$
$$\text{with triples } (a, b, c),\ (d, e, f),\ (d', e', f'),\ (x, y, z)$$
$$\mathcal{B} = (0, 6)$$

This decomposition yields two blocks $\mathcal{B}_1 = d\,y\,e\,x\,b\,f$ and $\mathcal{B}_2 = d'\,a\,f'\,z\,e'\,c$, which contain two internal triples (d,e,f) and (d',e',f') and two external triples forming a variable $A = [(a,b,c),(x,y,z)]$. We give below a sequence of 3DT-steps leading to the empty instance (for each step, the three deleted elements are in bold font, the elements that are swapped by the corresponding transposition are underlined, vertical bars give the limits of the blocks in the 2-block decomposition, and dot symbols are omitted). Note that in this example, variable A is valid, and remains valid until its activation.

$$
\begin{aligned}
I &= |\, \boldsymbol{d}\, \underline{y}\, \boldsymbol{e}\, \underline{x}\, b\, \boldsymbol{f} \,|\, d'\, a\, f'\, z\, e'\, c \,| \\
&\downarrow \quad (d,e,f) && \textit{Internal triple of } \mathcal{B}_1 \\
I_3 &= |\, \boldsymbol{x}\, \underline{b}\, \boldsymbol{y} \,|\, \underline{d'\, a\, f'}\, \boldsymbol{z}\, e'\, c \,| \\
&\downarrow \quad (x,y,z) && \textit{Activation of } A \\
I_2 &= |\, \varepsilon \,|\, d'\, \boldsymbol{a}\, \underline{f'}\, \boldsymbol{b}\, \underline{e'}\, \boldsymbol{c} \,| \\
&\downarrow \quad (a,b,c) && \textit{Internal triple of } \mathcal{B}_2 \\
I_1 &= |\, \varepsilon \,|\, \boldsymbol{d'}\, \boldsymbol{e'}\, \boldsymbol{f'} \,| \\
&\downarrow \quad (d',e',f') && \textit{Internal triple of } \mathcal{B}_2 \\
I_0 &= |\, \varepsilon \,|\, \varepsilon \,| = \varepsilon
\end{aligned}
$$

Property 1.11. *Let $(I,\mathcal{B})$ be a 3DT-instance with an l-block decomposition, and A be a variable of I that is valid in $\mathcal{B}$. Write $A = [(a,b,c),(x,y,z)]$. Then (x,y,z) is well-ordered iff $x \prec y$; and (a,b,c) is not well-ordered.*

Proof. Note that for all $\sigma, \sigma' \in \Sigma$, $block_{I,\mathcal{B}}(\sigma) < block_{I,\mathcal{B}}(\sigma') \Rightarrow \sigma \prec \sigma'$. Write $I = \langle \Sigma, T, \psi \rangle$, $h_0 = source(A)$ and $h_1 = target(A)$: we have $h_0 \neq h_1$ by condition (ii) of Definition 1.12.

If $h_0 < h_1$, then, with condition (iv) of Definition 1.12, $b \prec a \prec c$, and either $x \prec y \prec z$ or $y \prec x \prec z$. Hence, (a,b,c) is not well-ordered, and (x,y,z) is well-ordered iff $x \prec y$.

Likewise, if $h_1 < h_0$, we have $a \prec c \prec b$, and $z \prec x \prec y$ or $z \prec y \prec x$. Again, (a,b,c) is not well-ordered, and (x,y,z) is well-ordered iff $x \prec y$. □

Property 1.12. *Let $(I,\mathcal{B})$ be a 3DT-instance with an l-block decomposition, such that the external triples of $I = \langle \Sigma, T, \psi \rangle$ can be partitioned into a set of valid variables $\mathcal{A}$. Let (d,e,f) be a well-ordered triple of I, such that there exists a 3DT-step $(I,\mathcal{B}) \xrightarrow{(d,e,f)} (I',\mathcal{B}')$, with $I' = \langle \Sigma', T', \psi' \rangle$. Then one of the following two cases is true:*

- *(d,e,f) is an internal triple. We write $h_0 = block_{I,\mathcal{B}}(d) = block_{I,\mathcal{B}}(e) = block_{I,\mathcal{B}}(f)$. Then for all $\sigma \in \Sigma'$, $block_{I',\mathcal{B}'}(\sigma) = block_{I,\mathcal{B}}(\sigma)$. Moreover if $\sigma_1, \sigma_2 \in \Sigma'$ with $block_{I',\mathcal{B}'}(\sigma_1) = block_{I',\mathcal{B}'}(\sigma_2) \neq h_0$ and $\sigma_1 \prec_I \sigma_2$, then $\sigma_1 \prec_{I'} \sigma_2$.*
- *There exists a variable $A = [(a,b,c),(x,y,z)] \in \mathcal{A}$, such that $(d,e,f) = (x,y,z)$. Then $block_{I',\mathcal{B}'}(b) = target(A)$ and for all other $\sigma \in \Sigma' - \{b\}$, $block_{I',\mathcal{B}'}(\sigma) = block_{I,\mathcal{B}}(\sigma)$. Moreover if $\sigma_1, \sigma_2 \in \Sigma' - \{b\}$, such that $\sigma_1 \prec_I \sigma_2$, then $\sigma_1 \prec_{I'} \sigma_2$.*

Proof. We write τ for the transposition and i, j, k for the three integers such that $\tau = \tau_{i,j,k} = \tau[d, e, f, \psi]$ (necessarily, $0 < i < j < k \leq n$). We also write $\mathcal{B} = (s_0, s_1, \ldots, s_l)$. The triple (d, e, f) is either internal or external in $\mathcal{B}$.

If (d, e, f) is internal, with $h_0 = block_{I,\mathcal{B}}(d) = block_{I,\mathcal{B}}(e) = block_{I,\mathcal{B}}(f)$, we have (see Figure 1.8a):

$$s_{h_0} < i < j < k \leq t_{h_0}.$$

Hence for all $h \in [\![1\,;\,l]\!]$, either $s_h < i$ or $k \leq s_h$, and $\tau^{-1}(s_h) = s_h$ by Definition 1.1. Moreover, for all $\sigma \in \Sigma$, we have

$$\begin{array}{rcl} i \leq \psi(\sigma) < k & \Rightarrow & \psi(\sigma) \in [\![s_{h_0} + 1\,;\,t_{h_0}]\!] \\ & & \quad \text{and } \tau^{-1}(s_{h_0}) < i \leq \tau^{-1}(\psi(\sigma)) < k \leq \tau^{-1}(t_{h_0}) \\ & \Rightarrow & block_{I,\mathcal{B}}(\sigma) = h_0 = block_{I',\mathcal{B}'}(\sigma) \\ \psi(\sigma) < i \text{ or } k \leq \psi(\sigma) & \Rightarrow & \tau^{-1}(\psi(\sigma)) = \psi(\sigma) \\ & \Rightarrow & block_{I',\mathcal{B}'}(\sigma) = block_{I,\mathcal{B}}(\sigma) \end{array}$$

Finally, if $\sigma_1, \sigma_2 \in \Sigma'$ with $block_{I',\mathcal{B}'}(\sigma_1) = block_{I',\mathcal{B}'}(\sigma_2) \neq h_0$, then we have both $\tau^{-1}(\psi(\sigma_1)) = \psi(\sigma_1)$ and $\tau^{-1}(\psi(\sigma_2)) = \psi(\sigma_2)$. Hence $\sigma_1 \prec_I \sigma_2 \Leftrightarrow \sigma_1 \prec_{I'} \sigma_2$.

If (d, e, f) is external, then, since the set of external triples can be partitioned into variables, there exists a variable $A = [(a, b, c), (x, y, z)] \in \mathcal{A}$, such that $(d, e, f) = (a, b, c)$ or $(d, e, f) = (x, y, z)$. Since (d, e, f) is well-ordered in I, we have, by Property 1.11, $(d, e, f) = (x, y, z)$ and $x \prec_I y$, see Figure 1.8b. And since A is valid, by condition (iv) of Definition 1.12, $x \lhd_I b \lhd_I y$. We write $h_0 = source(A)$ and $h_1 = target(A)$, and we assume that $h_0 < h_1$, which implies $x \prec_I y \prec_I z$ (the case $h_1 < h_0$ with $z \prec_I x \prec_I y$ is similar): thus, we have

$$i = \psi(x),\ j = \psi(y),\ k = \psi(z),\ \text{and } s_{h_0} < i < j \leq t_{h_0} \leq s_{h_1} < k \leq t_{h_1}.$$

We define a set of indices U by

$$U = \{s_h \mid h \in [\![1\,;\,l]\!]\} \cup \{n\} \cup \{\psi(\sigma) \mid \sigma \in \Sigma' - \{b\}\}.$$

We now show that for all $u \in U$, we have $u < i$ or $j \leq u$. Indeed, if $u = s_h$ for some $h \in [\![1\,;\,l]\!]$, then either $h \leq h_0$ and $u \leq s_{h_0} < i$, or $h_0 < h$ and $j \leq t_{h_0} \leq u$. Also, if $u = n$, then $j \leq u$. Finally, assume $u = \psi(\sigma)$, with $\sigma \in \Sigma' - \{b\}$. We then have $x \prec_I \sigma \prec_I y \Leftrightarrow \sigma = b$, since $x \lhd_I b \lhd_I y$. Hence either $\sigma \prec_I x$ and $u < \psi(x) = i$, or $y \prec_I \sigma$ and $\psi(y) = j < u$.

By Property 1.1, if $u, v \in U$ are such that $u < v$, then $\tau^{-1}(u) < \tau^{-1}(v)$. This implies that elements of $\Sigma' - \{b\} = \Sigma - \{b, x, y, z\}$ do not change blocks after applying τ^{-1} on ψ, and that the relative order of any two elements is preserved. Finally, for b, we have $x \prec_I b \prec_I y$, hence

$$i \leq \psi(b) < j \leq s_{h_1} < k \leq t_{h_1}.$$

Thus, by Property 1.1, $\tau^{-1}(s_{h_1}) < \tau^{-1}(\psi(b)) < \tau^{-1}(t_{h_1})$, and $block_{I',\mathcal{B}'}(b) = h_1 = target(A)$. This completes the proof. □

Definition 1.13 (Valid context)**.** *A 3DT-instance with an l-block decomposition $(I, \mathcal{B})$ is a* valid context *if the set of external triples of I can be partitioned into valid variables.*

With the following property, we ensure that a valid context remains *almost* valid after applying a 3DT-step: the partition of the external triples into variables is kept through this 3DT-step, but conditions (iii) and (iv) of Definition 1.12 are not necessarily satisfied for all variables.

Property 1.13. *Let $(I, \mathcal{B})$ be a valid context and $(I, \mathcal{B}) \xrightarrow{(d,e,f)} (I', \mathcal{B}')$ be a 3DT-step. Then the external triples of $(I', \mathcal{B}')$ can be partitioned into a set of variables, each satisfying conditions* (i) *and* (ii) *of Definition 1.12.*

Proof. Let $I = \langle \Sigma, T, \psi \rangle$, $I' = \langle \Sigma', T', \psi' \rangle$, $\mathcal{A}$ be the set of variables of I, and E (resp. E') be the set of external triples of I (resp. I'). From Property 1.12, two cases are possible.

First case: $(d, e, f) \notin E$. Then for all $\sigma \in \Sigma'$, $block_{I',\mathcal{B}'}(\sigma) = block_{I,\mathcal{B}}(\sigma)$. Hence $E' = E$, and $(I', \mathcal{B}')$ has the same set of variables as $(I, \mathcal{B})$, that is $\mathcal{A}$. The source and target blocks of every variable remain unchanged, hence conditions (i) and (ii) of Definition 1.12 are still satisfied for each $A \in \mathcal{A}$ in $\mathcal{B}'$.

Second case: $(d, e, f) \in E$, and there exists a variable $A = [(a, b, c), (x, y, z)]$ in $\mathcal{A}$ such that $(d, e, f) = (x, y, z)$, by Property 1.12. Then $block_{I',\mathcal{B}'}(b) = target(A)$ and for all $\sigma \in \Sigma' - \{b\}$, $block_{I',\mathcal{B}'}(\sigma) = block_{I,\mathcal{B}}(\sigma)$. Hence $block_{I',\mathcal{B}'}(b) = block_{I',\mathcal{B}'}(a) = block_{I',\mathcal{B}'}(c)$, and $E' = E - \{(x, y, z), (a, b, c)\}$: indeed, (x, y, z) is deleted in T' so $(x, y, z) \notin E'$, (a, b, c) is internal in I', and every other triple is untouched. Finally, for every $A' = [(a', b', c'), (x', y', z')] \in \mathcal{A} - \{A\}$, we have $block_{I',\mathcal{B}'}(\sigma) = block_{I,\mathcal{B}}(\sigma)$ for $\sigma \in \{a', b', c', x', y', z'\}$, hence A' satisfies conditions (i) and (ii) of Definition 1.12 in $\mathcal{B}'$. □

Consider a block B in a valid context $(I, \mathcal{B})$ (there exists $h \in [\![1\,;\, l]\!]$ such that $B = \mathcal{B}_h$), and (d, e, f) a triple of I such that $(I, \mathcal{B}) \xrightarrow{(d,e,f)} (I', \mathcal{B}')$ (we write $B' = \mathcal{B}'_h$). Then, following Property 1.12, four cases are possible:

- $h \notin \{block_{I,\mathcal{B}}(d), block_{I,\mathcal{B}}(e), block_{I,\mathcal{B}}(f)\}$, hence $B' = B$, since, by Property 1.12, the relative order of the elements of B remains unchanged after the 3DT-step $\xrightarrow{(d,e,f)}$.
- (d, e, f) is an internal triple of B. We write

$$\boxed{B} \text{ —— } (d, e, f) \longrightarrow \boxed{B'}$$

- $\exists A = [(a, b, c), (x, y, z)]$ such that $h = source(A)$ and $(d, e, f) = (x, y, z)$ (A is an output of B), see Figure 1.9 (left). We write

$$\boxed{B} \text{ ═══ } A \Longrightarrow \boxed{B'}$$

- $\exists A = [(a, b, c), (x, y, z)]$ such that $h = target(A)$ and $(d, e, f) = (x, y, z)$ (A is an input of B), see Figure 1.9 (right). We write

$$\boxed{B} \text{ —— } A \longrightarrow \boxed{B'}$$

The graph obtained from a block B by following exhaustively the possible arcs as defined above (always assuming this block is in a valid context) is called the *behavior graph* of B.

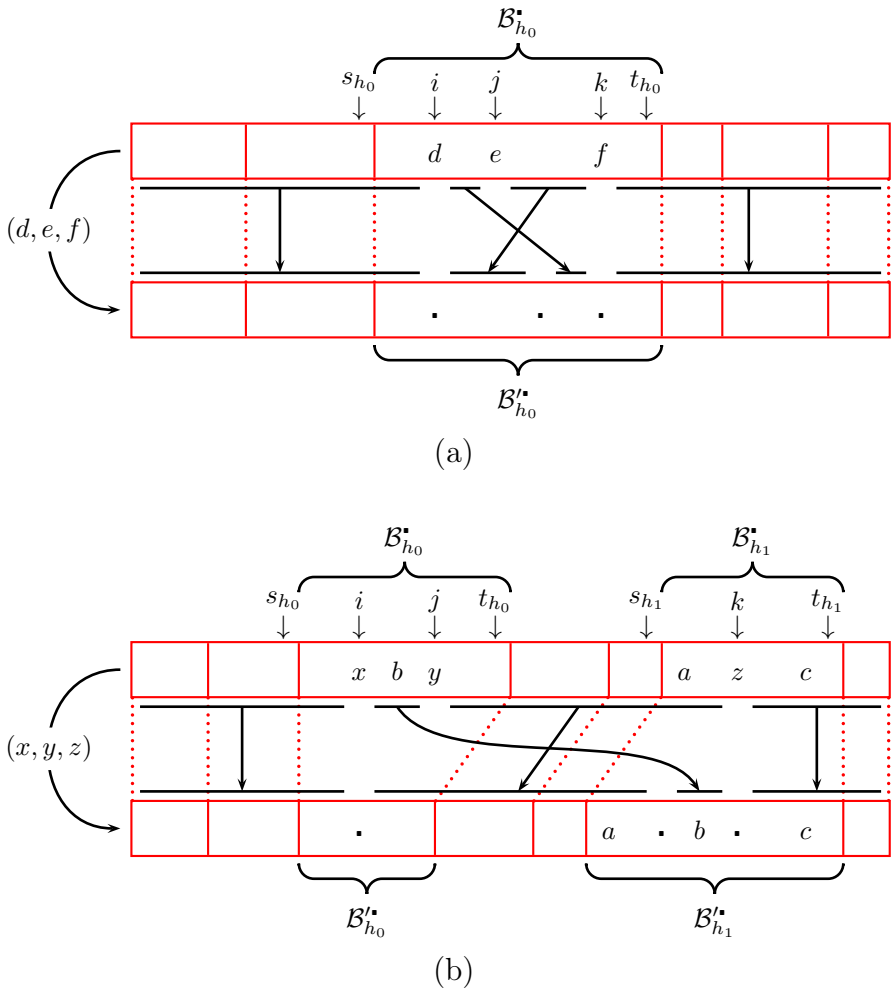


Figure 1.8: Effects of a 3DT-step $\xrightarrow{(d,e,f)}$ on an l-block decomposition if (a) (d, e, f) is an internal triple, or (b) there exists a variable $A = [(a, b, c), (x, y, z)]$ such that $(d, e, f) = (x, y, z)$. Both figures are in fact derived from Figure 1.3 in the context of an l-block decomposition.

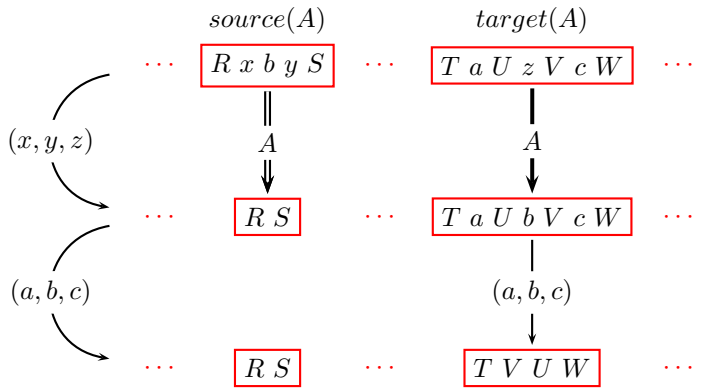


Figure 1.9: The activation of a variable $A = [(a, b, c), (x, y, z)]$ is written with a double arc in the behavior graph of the source block of A and with a thick arc in the behavior graph of its target block. It can be followed by the 3DT-step $\xrightarrow{(a,b,c)}$, impacting only the target block of A. Dot symbols ($\cdot$) are omitted. We denote by R, S, T, U, V, W some factors of the source and target blocks of A: the consequence of activating A is to allow U and V to be swapped in $target(A)$.

1.3.2 Basic Blocks

We now define four basic blocks: copy, and, or, and var. They are studied independently in this section, before being assembled in Section 1.3.3. Each of these blocks is defined by a word and a set of triples. We distinguish internal triples, for which all three elements appear in a single block, from external triples, which are part of an input/output variable, and for which only one or two elements appear in the block. Note that each external triple is part of an input (resp. output) variable, which itself must be an output (resp. input) of another block, the other block containing the remaining elements of the triple.

We then draw the behavior graph of each of these blocks (Figures 1.10 to 1.13): in each case, we assume that the block is in a valid context, and follow exhaustively the 3DT-steps that can be applied to it. We then give another graph (Figures 1.14a to 1.14d), obtained from the behavior graph by contracting all arcs corresponding to 3DT-steps using internal triples, i.e., we assimilate every pair of nodes linked by such an arc. Hence, only the arcs corresponding to the activation of an input/output variable remain. From this second figure, we derive a property describing the behavior of the block, in terms of activating input and output variables (always provided this block is in a valid context). It must be kept in mind that for any variable, the state of the source block determines whether it can be activated, whereas the activation itself affects mostly the target block.

The Block copy

This block aims at duplicating a variable: any of the two output variables can only be activated after the input variable has been activated.
Input variable: $A = [(a, b, c), (x, y, z)]$.
Output variables: $A_1 = [(a_1, b_1, c_1), (x_1, y_1, z_1)]$ and $A_2 = [(a_2, b_2, c_2), (x_2, y_2, z_2)]$.

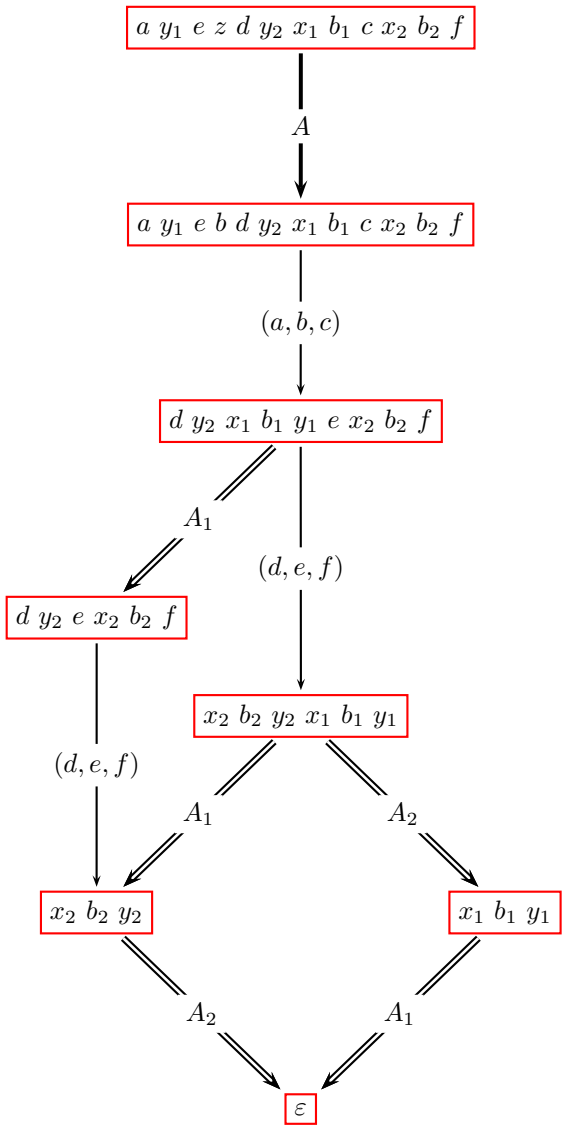


Figure 1.10: Behavior graph of the block $[A_1, A_2] = \mathsf{copy}(A)$. A thick (resp. double) arc corresponds to the 3DT-step $\xrightarrow{(x,y,z)}$ for an input (resp. output) variable $[(a,b,c),(x,y,z)]$.

Internal triple: (d, e, f).
Definition:

$$\boxed{[A_1, A_2] = \mathsf{copy}(A)} \quad = \quad \boxed{a\ y_1\ e\ z\ d\ y_2\ x_1\ b_1\ c\ x_2\ b_2\ f}$$

Property 1.14. *In a block $[A_1, A_2] = \mathsf{copy}(A)$ in a valid context, the possible orders in which A, A_1 and A_2 can be activated are (A, A_1, A_2) and (A, A_2, A_1).*

Proof. See Figures 1.10 and 1.14a. □

The Block and

This block aims at simulating a conjunction: the output variable can only be activated after both input variables have been activated.

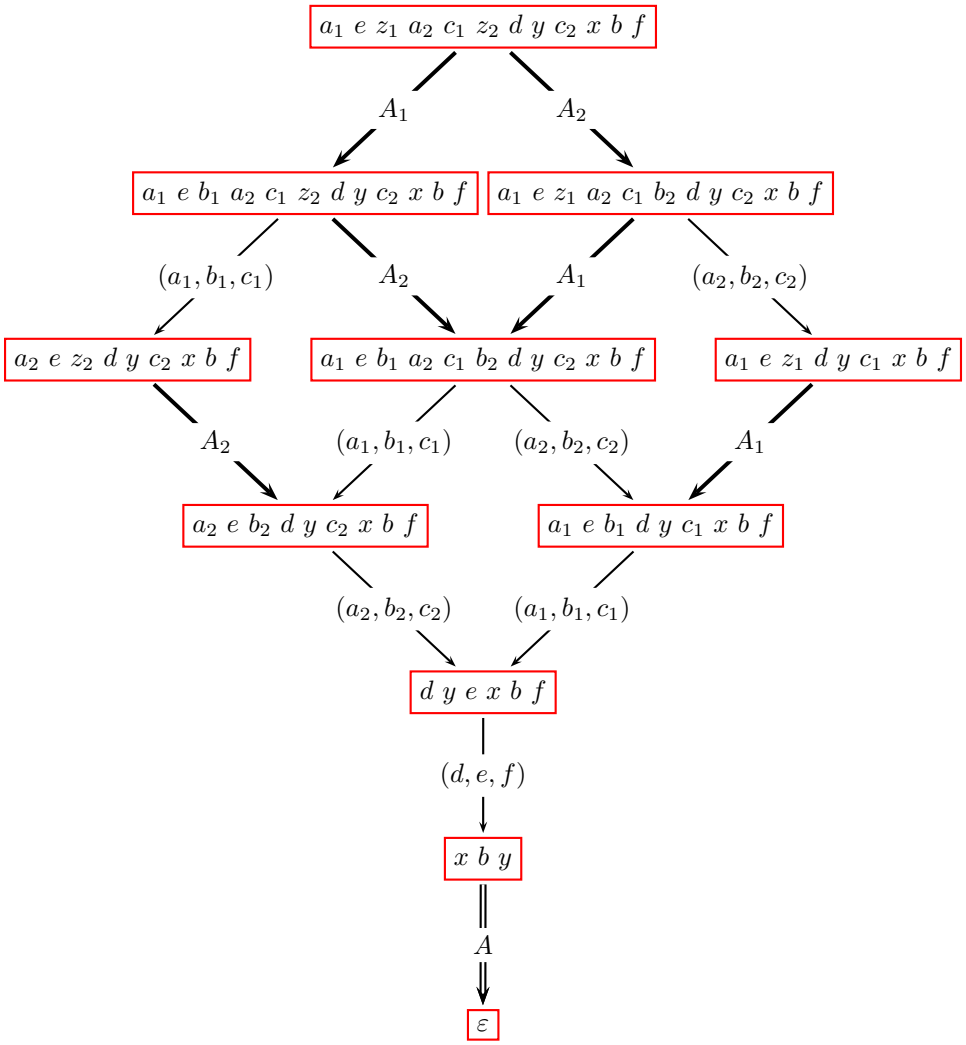


Figure 1.11: Behavior graph of the block $A = \mathsf{and}(A_1, A_2)$.

Input variables: $A_1 = [(a_1, b_1, c_1), (x_1, y_1, z_1)]$ and $A_2 = [(a_2, b_2, c_2), (x_2, y_2, z_2)]$.
Output variable: $A = [(a, b, c), (x, y, z)]$.
Internal triple: (d, e, f).
Definition:

$$\boxed{A = \mathsf{and}(A_1, A_2)} \quad = \quad \boxed{a_1\ e\ z_1\ a_2\ c_1\ z_2\ d\ y\ c_2\ x\ b\ f}$$

Property 1.15. *In a block $A = \mathsf{and}(A_1, A_2)$ in a valid context, the possible orders in which A, A_1 and A_2 can be activated are (A_1, A_2, A) and (A_2, A_1, A).*

Proof. See Figures 1.11 and 1.14b. □

The Block or

This block aims at simulating a disjunction: the output variable can be activated as soon as any of the two input variables is activated.
Input variables: $A_1 = [(a_1, b_1, c_1), (x_1, y_1, z_1)]$ and $A_2 = [(a_2, b_2, c_2), (x_2, y_2, z_2)]$.
Output variable: $A = [(a, b, c), (x, y, z)]$.
Internal triples: (a', b', c') and (d, e, f).

Definition:

$$\boxed{A = \mathsf{or}(A_1, A_2)} \quad = \quad \boxed{a_1\ b'\ z_1\ a_2\ d\ y\ a'\ x\ b\ f\ z_2\ c_1\ e\ c'\ c_2}$$

Property 1.16. *In a block $A = \mathsf{or}(A_1, A_2)$ in a valid context, the possible orders in which A, A_1 and A_2 can be activated are (A_1, A, A_2), (A_2, A, A_1), (A_1, A_2, A) and (A_2, A_1, A).*

Proof. See Figures 1.12 and 1.14c. □

The Block var

This block aims at simulating a boolean variable: in a first stage, only one of the two output variables can be activated. The other needs the activation of the input variable to be activated.
Input variable: $A = [(a, b, c), (x, y, z)]$.
Output variables: $A_1 = [(a_1, b_1, c_1), (x_1, y_1, z_1)]$, $A_2 = [(a_2, b_2, c_2), (x_2, y_2, z_2)]$.
Internal triples: (d_1, e_1, f_1), (d_2, e_2, f_2), and (a', b', c').
Definition:

$$\boxed{[A_1, A_2] = \mathsf{var}(A)} \quad = \quad \boxed{d_1\ y_1\ a\ d_2\ y_2\ e_1\ a'\ e_2\ x_1\ b_1\ f_1\ c'\ z\ b'\ c\ x_2\ b_2\ f_2}$$

Property 1.17. *In a block $[A_1, A_2] = \mathsf{var}(A)$ in a valid context, the possible orders in which A, A_1 and A_2 can be activated are (A_1, A, A_2), (A_2, A, A_1), (A, A_1, A_2) and (A, A_2, A_1).*

Proof. See Figures 1.13 and 1.14d. □

With such a block, if A is not activated first, one needs to make a choice between activating A_1 or A_2. Once A is activated, however, all the remaining output variables are activable.

Assembling the Blocks copy, and, or, var.

Definition 1.14 (Assembling of basic blocks)**.** *An* assembling of basic blocks $(I, \mathcal{B})$ *is composed of a 3DT-instance I and an l-block decomposition $\mathcal{B}$ obtained by the following process:*

- *Create a set of variables $\mathcal{A}$.*
- *Define $I = \langle \Sigma, T, \psi \rangle$ by its word representation, as a concatenation of l factors $\mathcal{B}_1^{\blacksquare}\ \mathcal{B}_2^{\blacksquare}\ \ldots\ \mathcal{B}_l^{\blacksquare}$ and a set of triples T, where each $\mathcal{B}_h^{\blacksquare}$ is one of the blocks $[A_1, A_2] = \mathsf{copy}(A)$, $A = \mathsf{and}(A_1, A_2)$, $A = \mathsf{or}(A_1, A_2)$ or $[A_1, A_2] = \mathsf{var}(A)$, with $A_1, A_2, A \in \mathcal{A}$ (such that each $X \in \mathcal{A}$ appears in the input of exactly one block, and in the output of exactly one other block); and where T is the union of the set of internal triples needed in each block, and the set of external triples defined by the variables of $\mathcal{A}$.*

Example 1.3. *We create a 3DT-instance I with a 2-block decomposition $\mathcal{B}$ such that $(I, \mathcal{B})$ is an assembling of basic blocks, defined as follows:*

- *I uses three variables, $\mathcal{A} = \{X_1, X_2, Y\}$*
- *the word representation of I is the concatenation of $[X_1, X_2] = \mathsf{var}(Y)$ and $Y = \mathsf{or}(X_1, X_2)$*

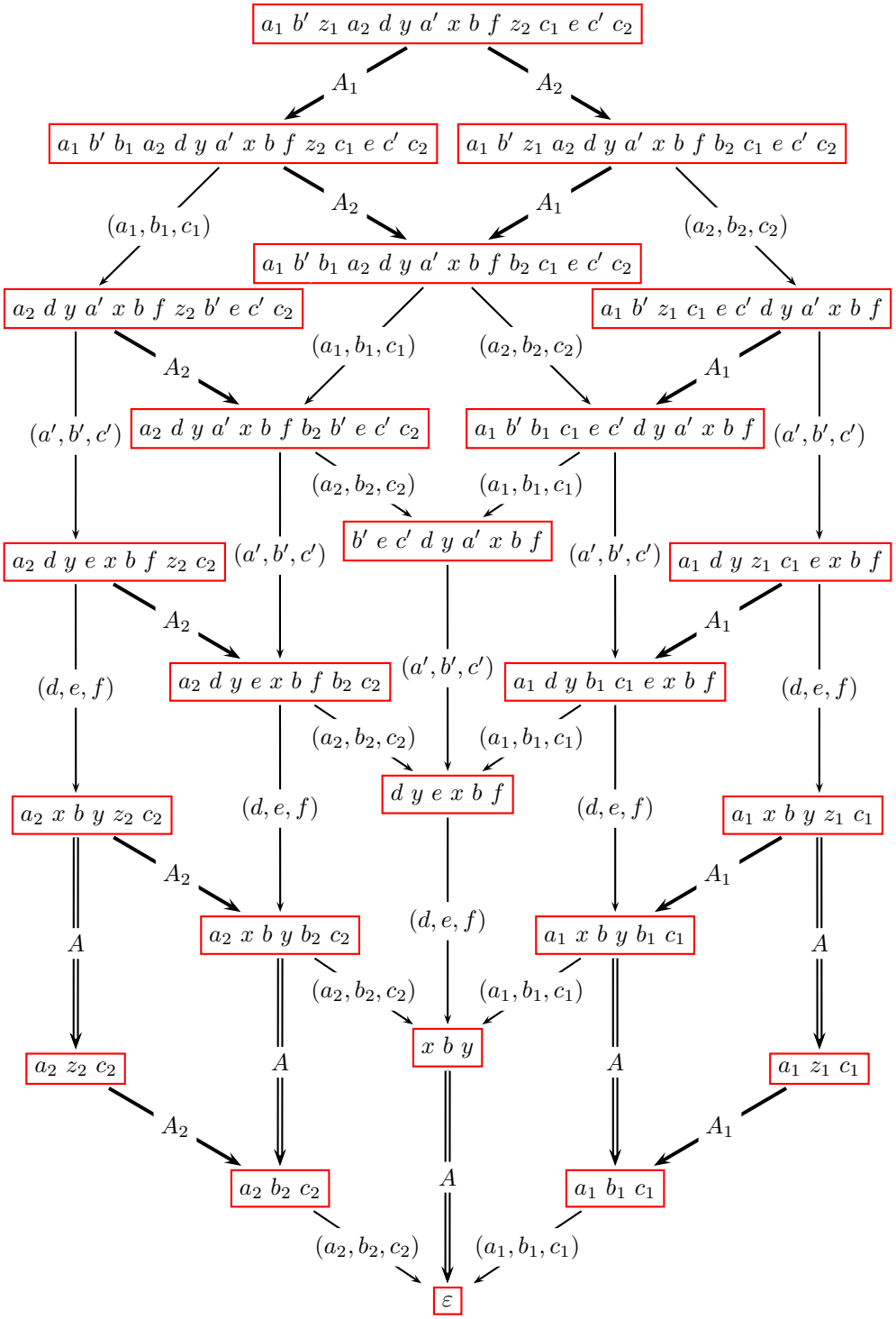


Figure 1.12: Behavior graph of the block $A = \mathsf{or}(A_1, A_2)$.

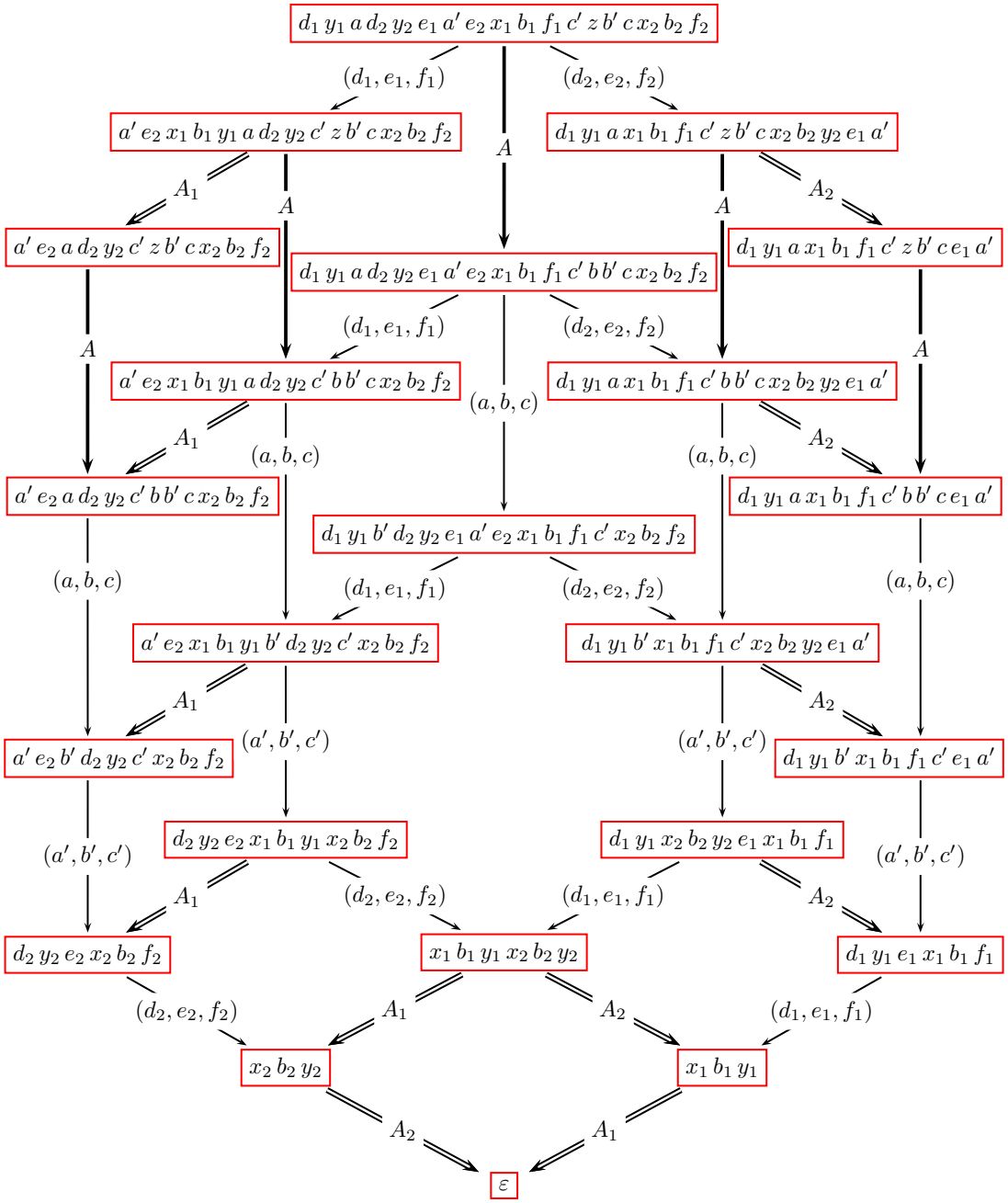


Figure 1.13: Behavior graph of the block $[A_1, A_2] = \mathsf{var}(A)$.

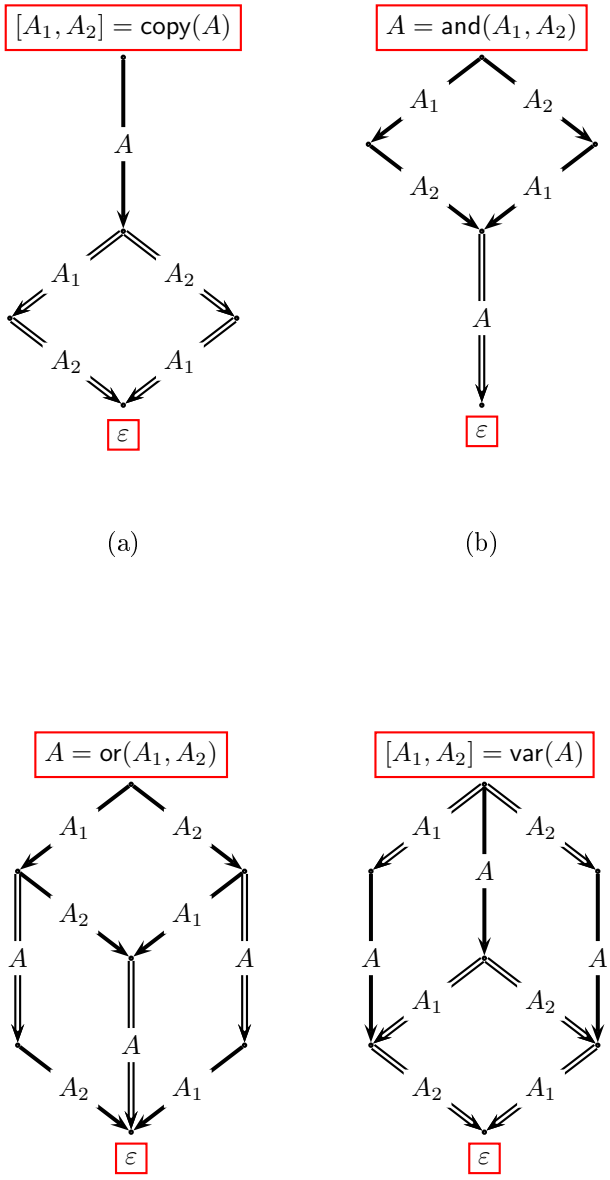


Figure 1.14: Abstract representations of the blocks copy, and, or, and var, obtained from each behavior graph (Figures 1.10, 1.11, 1.12, and 1.13) by contracting arcs corresponding to internal triples, and keeping only the arcs corresponding to variables. We see, for each block, which output variables are activable, depending on which variables have been activated.

With variables $X_1 = [(a_1, b_1, c_1), (x_1, y_1, z_1)]$, $X_2 = [(a_2, b_2, c_2), (x_2, y_2, z_2)]$, $Y = [(a, b, c), (x, y, z)]$, *and the internal triples* $(d_1, e_1, f_1), (d_2, e_2, f_2), (a', b', c')$ *for the block* var, *and* $(a'', b'', c''), (d, e, f)$ *for the block* or, *the word representation of* I *is the following (note that its 2-block decomposition, emphasized with the vertical bars, is* $(0, 18)$*):*

$$I{=}|d_1\ y_1\ a\ d_2\ y_2\ e_1\ a'\ e_2\ x_1\ b_1\ f_1\ c'\ z\ b'\ c\ x_2\ b_2\ f_2|a_1\ b''\ z_1\ a_2\ d\ y\ a''\ x\ b\ f\ z_2\ c_1\ e\ c''\ c_2|$$

A possible sequence of 3DT-steps leading from I *to* ε *is given in Figure 1.15, hence* I *is 3DT-collapsible.*

Lemma 1.18. *Let* I' *be a 3DT-instance with an* l*-block decomposition* $\mathcal{B}'$*, such that* $(I', \mathcal{B}')$ *is obtained from an assembling of basic blocks* $(I, \mathcal{B})$ *after any number of 3DT-steps, i.e., there exist* $k \geq 0$ *triples* (d_i, e_i, f_i), $i \in [\![1\,;\,k]\!]$, *such that* $(I, \mathcal{B}) \xrightarrow{(d_1, e_1, f_1)} \cdots \xrightarrow{(d_k, e_k, f_k)} (I', \mathcal{B}')$.

Then $(I', \mathcal{B}')$ *is a valid context. Moreover, if the set of variables of* $(I', \mathcal{B}')$ *is empty, then* I' *is 3DT-collapsible.*

Proof. Write $\mathcal{A}$ the set of variables used to define $(I, \mathcal{B})$. We write $I = \langle \Sigma, T, \psi \rangle$ and $I' = \langle \Sigma', T', \psi' \rangle$. We prove that $(I', \mathcal{B}')$ is a valid context by induction on k (the number of 3DT-steps between $(I, \mathcal{B})$ and $(I', \mathcal{B}')$). We also prove that for each $h \in [\![1\,;\,l]\!]$, $\mathcal{B}'_h$ appears as a node in the behavior graph of $\mathcal{B}_h$.

Suppose first that $k = 0$. We show that the set of external triples of $(I, \mathcal{B}) = (I', \mathcal{B}')$ can be partitioned into valid variables, namely into $\mathcal{A}$. Indeed, from the definition of each block, for each $\sigma \in \Sigma$, σ is either part of an internal triple, or appears in a variable $A \in \mathcal{A}$. Conversely, for each $A = [(a, b, c), (x, y, z)] \in \mathcal{A}$, b, x and y appear in the block having A for output, and a, c and z appear in the block having A for input. Hence (a, b, c) and (x, y, z) are indeed two external triples of $(I, \mathcal{B})$. Hence each variable satisfies conditions (i) and (ii) of Definition 1.12. Conditions (iii) and (iv) can be checked in the definition of each block: we have, for each output variable, $y \prec x$, and for each input variable, $a \prec z \prec c$. Finally, each $\mathcal{B}_h$ appears in its own behavior graph.

Suppose now that $(I', \mathcal{B}')$ is obtained from $(I, \mathcal{B})$ after k 3DT-steps, $k > 0$. Then there exists a 3DT-instance with an l-block decomposition $(I'', \mathcal{B}'')$ such that:

$$(I, \mathcal{B}) \xrightarrow{(d_1, e_1, f_1)} \cdots \xrightarrow{(d_{k-1}, e_{k-1}, f_{k-1})} (I'', \mathcal{B}'') \xrightarrow{(d_k, e_k, f_k)} (I', \mathcal{B}').$$

Consider $h \in [\![1\,;\,l]\!]$. By induction hypothesis, since $\mathcal{B}''_h$ is in a valid context $(I'', \mathcal{B}'')$, then, depending on (d_k, e_k, f_k), either $\mathcal{B}'_h = \mathcal{B}''_h$, or there is an arc from $\mathcal{B}''_h$ to $\mathcal{B}'_h$ in the behavior graph. Hence $\mathcal{B}'_h$ is indeed a node in this graph. By Property 1.13, we know that the set of external triples of $(I', \mathcal{B}')$ can be partitioned into variables satisfying conditions (i) and (ii) of Definition 1.12. Hence we need to prove that each variable satisfies conditions (iii) and (iv): by inspecting each node of the behavior graph, we verify that $x \prec y \Rightarrow x \lhd b \lhd y$ (resp. $a \prec z \prec c$) for each output (resp. input) variable $A = [(a, b, c), (x, y, z)]$ of the block. This concludes the induction proof.

We finally need to consider the case where the set of variables of $(I', \mathcal{B}')$ is empty. Then for each $h \in [\![1\,;\,l]\!]$ we either have $\mathcal{B}'_h = \varepsilon$, or $\mathcal{B}'_h = a_h\ b_h\ c_h$ for some internal triple (a_h, b_h, c_h) (in the case where $\mathcal{B}_h$ is a block or). Then $(I', \mathcal{B}')$ is indeed 3DT-collapsible: simply follow in any order the 3DT-step $\xrightarrow{(a_h, b_h, c_h)}$ for each remaining triple (a_h, b_h, c_h). □

$$I = |\, \boldsymbol{d_1}\, \underline{y_1\, a\, d_2\, y_2}\, \boldsymbol{e_1}\ \underline{a'\, e_2\, x_1\, b_1}\, \boldsymbol{f_1}\, c'\, z\, b'\, c\, x_2\, b_2\ f_2 \,|\, a_1\, b''\, z_1\, a_2\, d\, y\, a''\, x\, b\, f\, z_2\, c_1\, e\, c''\, c_2 \,|$$

$\downarrow\ (d_1, e_1, f_1)$ Internal triple of $\mathcal{B}_1$

$$I_{10} = |\, a'\, e_2\, \boldsymbol{x_1}\, \underline{b_1}\, \boldsymbol{y_1}\, \underline{a\ d_2\, y_2\, c'\, z\, b'\, c\, x_2\, b_2\, f_2 \,|\, a_1\, b''}\, \boldsymbol{z_1}\, a_2\, d\, y\, a''\, x\, b\, f\, z_2\, c_1\, e\, c''\, c_2\ |$$

$\downarrow\ (x_1, y_1, z_1)$ **Activation of** X_1

$$I_9 = |\, a'\, e_2\, a\, d_2\, y_2\, c'\, z\, b'\, c\, x_2\, b_2\, f_2 \,|\, \boldsymbol{a_1}\, \underline{b''}\, \boldsymbol{b_1}\, \underline{a_2\, d\, y\, a''\ x\, b\, f\, z_2}\, \boldsymbol{c_1}\, e\, c''\, c_2 \,|$$

$\downarrow\ (a_1, b_1, c_1)$ Internal triple of $\mathcal{B}_2$

$$I_8 = |\, a'\, e_2\, a\, d_2\, y_2\, c'\, z\, b'\, c\, x_2\, b_2\, f_2 \,|\, a_2\, d\, y\, \boldsymbol{a''}\, \underline{x\, b\, f\, z_2}\, \boldsymbol{b''}\ \underline{e}\, \boldsymbol{c''}\, c_2 \,|$$

$\downarrow\ (a'', b'', c'')$ Internal triple of $\mathcal{B}_2$

$$I_7 = |\, a'\, e_2\, a\, d_2\, y_2\, c'\, z\, b'\, c\, x_2\, b_2\, f_2 \,|\, a_2\, \boldsymbol{d}\, \underline{y}\, \boldsymbol{e}\, \underline{x\, b}\, \boldsymbol{f}\ z_2\, c_2 \,|$$

$\downarrow\ (d, e, f)$ Internal triple of $\mathcal{B}_2$

$$I_6 = |\, a'\, e_2\, a\, d_2\, y_2\, c'\, \boldsymbol{z}\, \underline{b'\, c\, x_2\, b_2\ f_2 \,|\, a_2}\, \boldsymbol{x}\, \underline{b}\, \boldsymbol{y}\, z_2\, c_2 \,|$$

$\downarrow\ (x, y, z)$ **Activation of** Y

$$I_5 = |\, a'\, e_2\, \boldsymbol{a}\, \underline{d_2\, y_2\, c'}\, \boldsymbol{b}\ \underline{b'}\, \boldsymbol{c}\, x_2\, b_2\, f_2 \,|\, a_2\, z_2\, c_2 \,|$$

$\downarrow\ (a, b, c)$ Internal triple of $\mathcal{B}_1$

$$I_4 = |\, \boldsymbol{a'}\, \underline{e_2}\, \boldsymbol{b'}\, \underline{d_2\, y_2}\ \boldsymbol{c'}\, x_2\, b_2\, f_2 \,|\, a_2\, z_2\, c_2 \,|$$

$\downarrow\ (a', b', c')$ Internal triple of $\mathcal{B}_1$

$$I_3 = |\, \boldsymbol{d_2}\, \underline{y_2}\, \boldsymbol{e_2}\, \underline{x_2\, b_2}\ \boldsymbol{f_2} \,|\, a_2\, z_2\, c_2 \,|$$

$\downarrow\ (d_2, e_2, f_2)$ Internal triple of $\mathcal{B}_1$

$$I_2 = |\, \boldsymbol{x_2}\, \underline{b_2}\, \boldsymbol{y_2} \,|\, \underline{a_2}\, \boldsymbol{z_2}\, c_2 \,|$$

$\downarrow\ (x_2, y_2, z_2)$ **Activation of** X_2

$$I_1 = |\, \varepsilon \,|\, \boldsymbol{a_2}\, \boldsymbol{b_2}\, \boldsymbol{c_2} \,|$$

$\downarrow\ (a_2, b_2, c_2)$ Internal triple of $\mathcal{B}_2$

$$I_0 = |\, \varepsilon \,|\, \varepsilon \,| = \varepsilon$$

Figure 1.15: 3DT-collapsibility of the assembling of basic blocks $[X_1, X_2] = \mathsf{var}(Y)$ and $Y = \mathsf{or}(X_1, X_2)$ from Example 1.3. We use the same notations as in Example 1.2.

1.3.3 Construction

Let ϕ be a boolean formula over the boolean variables $x_1, \ldots, x_m$, given in conjunctive normal form: $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_\gamma$. Each clause C_c ($c \in [\![1\,;\,\gamma]\!]$) is the disjunction of a number of literals, x_i or $\neg x_i$, $i \in [\![1\,;\,m]\!]$. We write q_i (resp. $\bar{q}_i$) for the number of occurrences of the literal x_i (resp. $\neg x_i$) in ϕ, $i \in [\![1\,;\,m]\!]$. We also write $k(C_c)$ the number of literals appearing in the clause C_c, $c \in [\![1\,;\,\gamma]\!]$. We can assume that $\gamma \geq 2$, that for each $c \in [\![1\,;\,\gamma]\!]$, we have $k(C_c) \geq 2$, and that for each $i \in [\![1\,;\,m]\!]$, $q_i \geq 2$ and $\bar{q}_i \geq 2$. (Otherwise, we can always add clauses of the form $(x_i \vee \neg x_i)$ to ϕ, or duplicate the literals appearing in the clauses C_c such that $k(C_c) = 1$.) In order to distinguish variables of an l-block decomposition from $x_1, \ldots, x_m$, we always use the term *boolean variable* for the latter.

The 3DT-instance I_ϕ is defined as an assembling of basic blocks: we first define a set of variables, then we list the blocks of which the word representation of I_ϕ is the concatenation. It is necessary that each variable is part of the input (resp. the output) of exactly one block. Note that the relative order of the blocks is of no importance. We simply try, for readability reasons, to ensure that the source of a variable appears before its target, whenever possible. We say that a variable *represents* a term, i.e., a literal, clause or formula, if it can be activated iff this term is true (for some fixed assignment of the boolean variables), or if ϕ is satisfied by this assignment. We also say that a block *defines* a variable if it is the source block of this variable.

The construction of I_ϕ is done as follows (see Figure 1.16 for an example):

Create a set of variables:

- For each $i \in [\![1\,;\,m]\!]$, create $q_i + 1$ variables representing x_i: X_i and X_i^j, $j \in [\![1\,;\,q_i]\!]$, and $\bar{q}_i + 1$ variables representing $\neg x_i$: $\bar{X}_i$ and $\bar{X}_i^j$, $j \in [\![1\,;\,\bar{q}_i]\!]$.
- For each $c \in [\![1\,;\,\gamma]\!]$, create a variable Γ_c representing the clause C_c.
- Create $m + 1$ variables, A_ϕ and A_ϕ^i, $i \in [\![1\,;\,m]\!]$, representing the formula ϕ. We will show that A_ϕ has a key role in the construction: it can be activated only if ϕ is satisfiable, and, once activated, it allows every remaining variable to be activated.
- We also use a number of intermediate variables, with names U_i^j, $\bar{U}_i^j$, V_c^i, W_c, and Y_i.

Add blocks successively, starting with an empty 3DT-instance ε:

- For each $i \in [\![1\,;\,m]\!]$, add the following $q_i + \bar{q}_i - 1$ blocks defining the variables X_i, X_i^j ($j \in [\![1\,;\,q_i]\!]$), and $\bar{X}_i$, $\bar{X}_i^j$ ($j \in [\![1\,;\,\bar{q}_i]\!]$):

$$
\begin{array}{cc}
[X_i, \bar{X}_i] = \mathsf{var}(A_\phi^i) & \\
[X_i^1, U_i^2] = \mathsf{copy}(X_i) & [\bar{X}_i^1, \bar{U}_i^2] = \mathsf{copy}(\bar{X}_i) \\
[X_i^2, U_i^3] = \mathsf{copy}(U_i^2) & [\bar{X}_i^2, \bar{U}_i^3] = \mathsf{copy}(\bar{U}_i^2) \\
\vdots & \vdots \\
[X_i^{q_i-2}, U_i^{q_i-1}] = \mathsf{copy}(U_i^{q_i-2}) & [\bar{X}_i^{\bar{q}_i-2}, \bar{U}_i^{\bar{q}_i-1}] = \mathsf{copy}(\bar{U}_i^{\bar{q}_i-2}) \\
[X_i^{q_i-1}, X_i^{q_i}] = \mathsf{copy}(U_i^{q_i-1}) & [\bar{X}_i^{\bar{q}_i-1}, \bar{X}_i^{\bar{q}_i}] = \mathsf{copy}(\bar{U}_i^{\bar{q}_i-1})
\end{array}
\qquad (*)
$$

- For each $c \in [\![1\,;\,\gamma]\!]$, let $C_c = \lambda_1 \vee \lambda_2 \vee \ldots \vee \lambda_k$, with $k = k(C_c)$. Consider $p \in [\![1\,;\,k]\!]$. There exist integers i, j such that λ_p is the j-th occurrence of

a literal x_i or $\neg x_i$, we respectively write $L_p = X_i^j$ or $L_p = \bar{X}_i^j$. We add the following $k-1$ blocks defining Γ_c:

$$\begin{aligned} V_c^2 &= \mathsf{or}(L_1, L_2) \\ V_c^3 &= \mathsf{or}(V_c^2, L_3) \\ &\vdots \\ V_c^{k-1} &= \mathsf{or}(V_c^{k-2}, L_{k-1}) \\ \Gamma_c &= \mathsf{or}(V_c^{k-1}, L_k) \end{aligned} \qquad (**)$$

– Since $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_\gamma$, the formula variable A_ϕ is defined by the following $\gamma - 1$ blocks:

$$\begin{aligned} W_2 &= \mathsf{and}(\Gamma_1, \Gamma_2) \\ W_3 &= \mathsf{and}(W_2, \Gamma_3) \\ &\vdots \\ W_{\gamma-1} &= \mathsf{and}(W_{\gamma-2}, \Gamma_{\gamma-1}) \\ A_\phi &= \mathsf{and}(W_{\gamma-1}, \Gamma_\gamma) \end{aligned} \qquad (***)$$

– The m copies $A_\phi^1, \ldots, A_\phi^m$ of A_ϕ are defined with the following $m-1$ blocks:

$$\begin{aligned} [A_\phi^1, Y_2] &= \mathsf{copy}(A_\phi) \\ [A_\phi^2, Y_3] &= \mathsf{copy}(Y_2) \\ &\vdots \\ [A_\phi^{m-2}, Y_{m-1}] &= \mathsf{copy}(Y_{m-2}) \\ [A_\phi^{m-1}, A_\phi^m] &= \mathsf{copy}(Y_{m-1}) \end{aligned} \qquad (****)$$

1.3.4 The Main Result

Theorem 1.19. *Let ϕ be a boolean formula, and I_ϕ the 3DT-instance defined in Section 1.3.3. The construction of I_ϕ is polynomial in the size of ϕ, and ϕ is satisfiable iff I_ϕ is 3DT-collapsible.*

Proof. The polynomial-time complexity of the construction of I_ϕ is trivial. We use the same notations as in the construction, with $\mathcal{B}$ the block decomposition of I_ϕ. One can easily check that each variable in (*), (**), (***), and (****) has exactly one source block and one target block. Then, by Lemma 1.18, we know that $(I_\phi, \mathcal{B})$ is a valid context, and remains so after any number of 3DT-steps, hence Properties 1.14, 1.15, 1.16, and 1.17 are satisfied by respectively each block copy, and, or and var, of I_ϕ.

$\boxed{\Rightarrow}$ Assume first that ϕ is satisfiable. Consider a truth assignment satisfying ϕ: let P be the set of indices $i \in [\![1\,;\,m]\!]$ such that x_i is assigned to true. Starting from I_ϕ, we can follow a path of 3DT-steps that activates all the variables of I_ϕ in the following order:

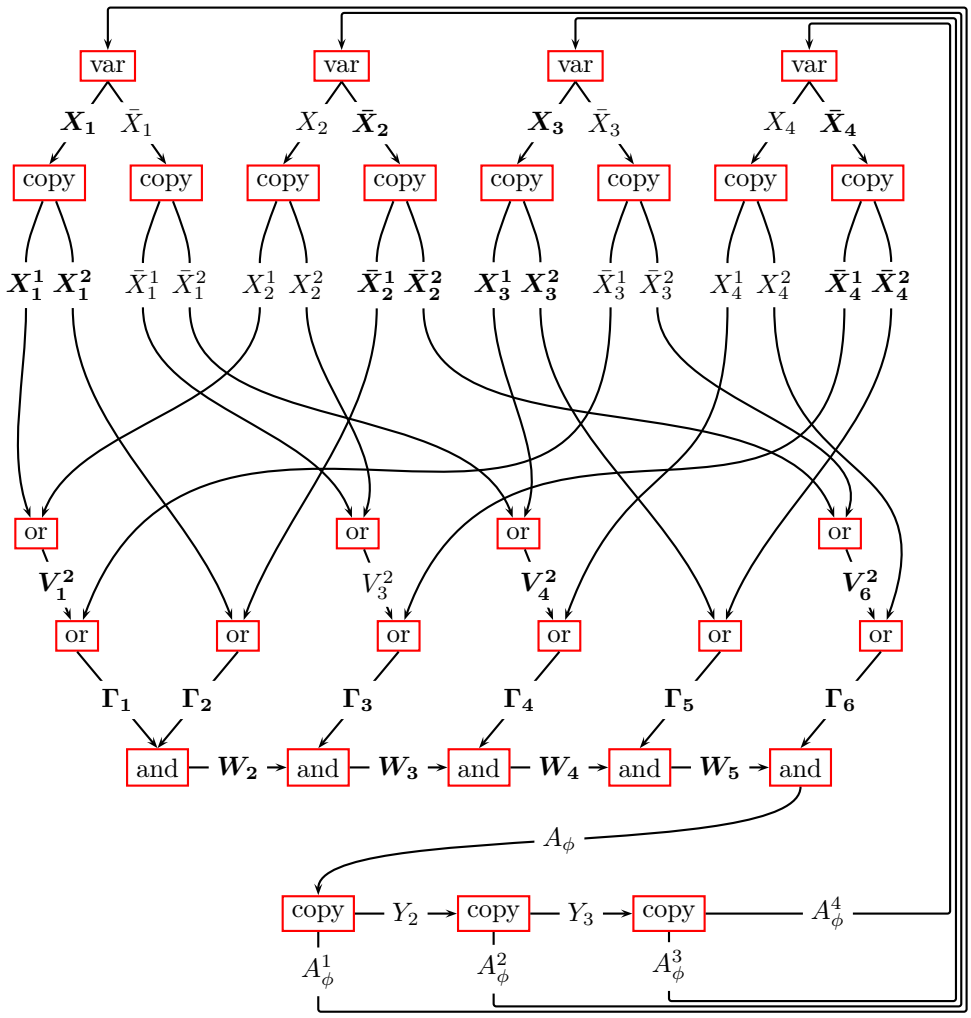


Figure 1.16: Schematic diagram of the blocks defining I_ϕ for $\phi = (x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee \neg x_4) \wedge (\neg x_1 \vee x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \wedge (\neg x_2 \vee \neg x_3 \vee x_4)$. For each variable, we draw an arc between its source and target block. Note that ϕ is satisfiable (e.g., with the assignment $x_1 = x_3 = \mathit{true}$ and $x_2 = x_4 = \mathit{false}$). A set of variables that can be activated before A_ϕ is in bold, they correspond to the terms being true in ϕ for the assignment $x_1 = x_3 = \mathit{true}$ and $x_2 = x_4 = \mathit{false}$.

- For $i \in [\![1\,;\,m]\!]$, if $i \in P$, activate X_i in the corresponding block var in $(*)$. Then, with the blocks copy, activate successively all intermediate variables U_i^j for $j = 2$ to $q_i - 1$, and variables X_i^j for $j \in [\![1\,;\,q_i]\!]$.
  Otherwise, if $i \notin P$, activate $\bar{X}_i$, all intermediate variables $\bar{U}_i^j$ for $j = 2$ to $\bar{q}_i - 1$, and the variables $\bar{X}_i^j$ for $j \in [\![1\,;\,\bar{q}_i]\!]$
- For each $c \in [\![1\,;\,\gamma]\!]$, let $C_c = \lambda_1 \vee \lambda_2 \vee \ldots \vee \lambda_k$, with $k = k(C_c)$. Since C_c is true with the selected truth assignment, at least one literal λ_{p_0}, $p_0 \in [\![1\,;\,k]\!]$, is true. If λ_{p_0} is the j-th occurrence of a literal x_i or $\neg x_i$, then the corresponding variable L_{p_0} ($L_{p_0} = X_i^j$ or $L_{p_0} = \bar{X}_i^j$) has been activated previously. Using the blocks or in $(**)$, we activate successively each intermediate variable V_c^p for $p = p_0, \ldots, k-1$, and finally we activate the variable Γ_c.
- Since all variables Γ_c, $c \in [\![1\,;\,\gamma]\!]$, have been activated, using the blocks and in $(***)$, we activate each intermediate variable W_c for $c = 2$ to $c = \gamma - 1$, and the formula variable A_ϕ.
- With the blocks copy in $(****)$, we activate successively all the intermediate variables Y_i, $i \in [\![2\,;\,m-1]\!]$ and the m copies $A_\phi^1, \ldots, A_\phi^m$ of A_ϕ.
- For $i \in [\![1\,;\,m]\!]$, since the variable A_ϕ^i has been activated, we activate in the block var of $(*)$ the remaining variable X_i or $\bar{X}_i$. We also activate all its copies and corresponding intermediate variables U_i^j or $\bar{U}_i^j$.
- For $c \in [\![1\,;\,\gamma]\!]$, in $(**)$, since all variables L_p have been activated, we activate the remaining intermediate variables V_c^p.
- At this point, every variable has been activated. Using again Lemma 1.18, we know that the resulting instance is 3DT-collapsible, and can be reduced down to the empty 3DT-instance ε.

Hence I_ϕ is 3DT-collapsible.

$\boxed{\Leftarrow}$ Assume now that I_ϕ is 3DT-collapsible: we consider a sequence of 3DT-steps reducing I_ϕ to ε. This sequence gives a total order on the set of variables: the order in which they are activated. We write Q for the set of variables activated before A_ϕ, and $P \subseteq [\![1\,;\,m]\!]$ for the set of indices i such that $X_i \in Q$ (see the variables in bold in Figure 1.16). We show that the truth assignment defined by $(x_i = \mathit{true} \Leftrightarrow i \in P)$ satisfies the formula ϕ.

- For each $i \in [\![1\,;\,m]\!]$, A_ϕ^i cannot belong to Q, using the property of the block copy in $(****)$ (each A_ϕ^i can only be activated after A_ϕ). Hence, with the block var in $(*)$, we have $X_i \notin Q$ or $\bar{X}_i \notin Q$. Moreover, with the block copy, we have

$$\forall 1 \leq j \leq q_i, \quad X_i^j \in Q \Rightarrow X_i \in Q \tag{a}$$
$$\forall 1 \leq j \leq \bar{q}_i, \quad \bar{X}_i^j \in Q \Rightarrow \bar{X}_i \in Q \Rightarrow X_i \notin Q \tag{b}$$

- Since A_ϕ is defined in a block $A_\phi = \mathsf{and}(W_{\gamma-1}, \Gamma_\gamma)$ in $(***)$, we necessarily have $W_{\gamma-1} \in Q$ and $\Gamma_\gamma \in Q$. Likewise, since $W_{\gamma-1}$ is defined by $W_{\gamma-1} = \mathsf{and}(W_{\gamma-2}, \Gamma_{\gamma-1})$, we also have $W_{\gamma-2} \in Q$ and $\Gamma_{\gamma-1} \in Q$. Applying this reasoning recursively, we have $\Gamma_c \in Q$ for each $c \in [\![1\,;\,\gamma]\!]$.
- For each $c \in [\![1\,;\,\gamma]\!]$, consider the clause $C_c = \lambda_1 \vee \lambda_2 \vee \ldots \vee \lambda_k$, with $k = k(C_c)$. Using the property of the block or in $(**)$, there exists some $p_0 \in [\![1\,;\,k]\!]$ such that the variable L_{p_0} is activated before Γ_c: hence $L_{p_0} \in Q$. If the corresponding literal λ_{p_0} is the j-th occurrence of x_i (respectively, $\neg x_i$), then $L_{p_0} = X_i^j$ (resp., $L_{p_0} = \bar{X}_i^j$), thus by (a) (resp. (b)), $X_i \in Q$ (resp., $X_i \notin Q$), and consequently $i \in P$ (resp., $i \notin P$). In both cases, the literal λ_{p_0} is true in the truth assignment defined by $(x_i = \mathit{true} \Leftrightarrow i \in P)$.

If I_ϕ is 3DT-collapsible, we have found a truth assignment such that at least one literal is true in each clause of the formula ϕ, and thus ϕ is satisfiable. □

1.4 Sorting by Transpositions Is NP-Hard

As noted previously, there is no guarantee that any 3DT-instance I has an equivalent permutation π. However, with the following theorem, we show that such a permutation can be found in the special case of assemblings of basic blocks, which is the case we are interested in, in order to complete our reduction.

Theorem 1.20. *Let I be a 3DT-instance of span n with $\mathcal{B}$ an l-block decomposition such that $(I, \mathcal{B})$ is an assembling of basic blocks. Then there exists a permutation π_I, computable in polynomial time in n, such that $I \sim \pi_I$.*

An example of the construction of π_I for the 3DT-instance defined in Example 1.3 is given in Figure 1.17.

The rough idea of the proof is as follows. Each block $\mathcal{B}_h$ of $\mathcal{B}$ (corresponding to positions $[\![s_h + 1\,;\, t_h]\!]$) is assigned a unique interval of integers $[\![p_h + 1\,;\, q_h]\!]$. Then we create a permutation of $[\![p_h + 1\,;\, q_h]\!]$, depending on the kind of the block, and π_I is obtained as the concatenation of these permutations. However, external triples (in variables) need a special treatment: when a variable is activated, exactly three integers are moved from one block to another. Hence, for each variable, some integers which should appear in the target block are originally present in the source block. In other words, the part of π_I corresponding to each block has three extra integers for each output variable, and three missing integers for each input variable. We keep track of elements which are displaced with two functions, α and β: for a variable A, the three affected integers are $\alpha(A) + 1$, $\alpha(A) + 2$ and $\beta(A) + 1$.

Proof. Let $\mathcal{A}$ be the set of variables of the l-block decomposition $\mathcal{B}$ of $I = \langle \Sigma, T, \psi \rangle$. Let n be the span of I, and L its domain. Note that $L = [\![1\,;\, n]\!]$. For any $h \in [\![1\,;\, l]\!]$, we write $ni(\mathcal{B}_h)$ (resp. $no(\mathcal{B}_h)$) for the number of input (resp. output) variables of $\mathcal{B}_h$. We also define two integers p_h, q_h by:

$$\begin{aligned} & & p_1 &= 0 \\ \forall h \in [\![1\,;\, l]\!]\,, & & q_h &= p_h + t_h - s_h + 3(ni(\mathcal{B}_h) - no(\mathcal{B}_h)) \\ \forall h \in [\![2\,;\, l]\!]\,, & & p_h &= q_{h-1} \end{aligned}$$

The permutation π_I will be defined such that p_h and q_h have the following property for any $h \in [\![1\,;\, l]\!]$: $\pi_I(s_h) = p_h$, and $\pi_I(t_h) = q_h$.

We also define two applications α, β over the set $\mathcal{A}$ of variables. The permutation π_I will be defined so that, for any variable $A = [(a, b, c), (x, y, z)]$, we have $\pi_I(\psi(a) - 1) = \alpha(A)$ and $\pi_I(\psi(z) - 1) = \beta(A)$. In order to have this property, α and β are defined as follows.

For each $h \in [\![1\,;\, l]\!]$:

– If $\mathcal{B}_h$ is a block of the kind $[A_1, A_2] = \mathsf{copy}(A)$, define

$$\alpha(A) = p_h,\ \beta(A) = p_h + 4.$$

– If $\mathcal{B}_h$ is a block of the kind $A = \mathsf{and}(A_1, A_2)$, define

$$\alpha(A_1) = p_h,\ \beta(A_1) = p_h + 7,\ \alpha(A_2) = p_h + 3,\ \beta(A_2) = p_h + 9.$$

Assembling of two basic blocs:

$[X_1, X_2] = \mathsf{var}(Y)$ $\qquad$ $Y = \mathsf{or}(X_1, X_2)$

Input variable (target of): Y $\qquad$ Input variables (target of): X_1, X_2
Output variables (source of): X_1, X_2 $\qquad$ Output variable (source of): Y

$s_1 = 0 \;\; t_1 = 18$ $\qquad$ $s_2 = 18 \;\; t_2 = 33$
$p_1 = 0 \;\; q_1 = p_1 + 18 - 3 = 15$ $\qquad$ $p_2 = 15 \;\; q_2 = p_2 + 15 + 3 = 33$

$\alpha(Y) = p_1 + 5 = 5$ $\qquad$ $\alpha(X_1) = p_2 = 15$ $\qquad$ $\alpha(X_2) = p_2 + 3 = 18$
$\beta(Y) = p_1 + 9 = 9$ $\qquad$ $\beta(X_1) = p_2 + 13 = 28$ $\qquad$ $\beta(X_2) = p_2 + 16 = 31$

Definition of π_I over $[\![s_1 + 1\,;\, t_1]\!]$:

u	$=0$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18 ...
$\pi_I(u)$	$=0$	17	5	3	20	12	1	14	4	29	16	13	9	8	2	11	32	19	15 ...
$\psi^{-1}(u)$	$=$	d_1	y_1	a	d_2	y_2	e_1	a'	e_2	x_1	b_1	f_1	c'	z	b'	c	x_2	b_2	f_2 ...

over $[\![s_2 + 1\,;\, t_2]\!]$:

u	$=$	...	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
$\pi_I(u)$	$=$	...	22	28	18	24	7	27	26	10	6	31	21	30	25	23	33
$\psi^{-1}(u)$	$=$	...	a_1	b''	z_1	a_2	d	y	a''	x	b	f	z_2	c_1	e	c''	c_2

Figure 1.17: Creation of a permutation π_I equivalent to the assembling of basic blocks $I = \langle \Sigma, T, \psi \rangle$ of span 33 defined in Example 1.3, using the proof of Theorem 1.20.

– If $\mathcal{B}_h$ is a block of the kind $A = \mathsf{or}(A_1, A_2)$, define

$$\alpha(A_1) = p_h, \;\; \beta(A_1) = p_h + 13, \;\; \alpha(A_2) = p_h + 3, \;\; \beta(A_2) = p_h + 16.$$

– If $\mathcal{B}_h$ is a block of the kind $[A_1, A_2] = \mathsf{var}(A)$, define

$$\alpha(A) = p_h + 5, \;\; \beta(A) = p_h + 9.$$

Note that for every $A \in \mathcal{A}$, $\alpha(A)$ and $\beta(A)$ are defined once and only once, depending on the kind of the block $\mathcal{B}_{target(A)}$. As already noted, the permutation π_I is designed in such a way that the image by π_I of an interval $[\![s_h + 1\,;\, t_h]\!]$ is essentially the interval $[\![p_h + 1\,;\, q_h]\!]$. However, there are exceptions: namely, for each variable A, the integers $\alpha(A) + 1, \alpha(A) + 2, \beta(A) + 1$, which are included in $[\![p_{target(A)} + 1\,;\, q_{target(A)}]\!]$, are in the image of $[\![s_{source(A)} + 1\,;\, t_{source(A)}]\!]$. This is formally described as follows. For each $h \in [\![1\,;\, k]\!]$ we define a set P_h by:

$$P_h = [\![p_h + 1\,;\, q_h]\!] \;\cup \bigcup_{A \text{ output of } \mathcal{B}_h} \{\alpha(A) + 1, \alpha(A) + 2, \beta(A) + 1\}$$
$$- \bigcup_{A \text{ input of } \mathcal{B}_h} \{\alpha(A) + 1, \alpha(A) + 2, \beta(A) + 1\}$$

We note that the sets $\{\alpha(A)+1, \alpha(A)+2, \beta(A)+1\}$ are disjoint for different variables A, and are each included in their respective interval $[\![p_{target(A)} + 1\,;\, q_{target(A)}]\!]$. Hence for any $h \in [\![1\,;\, l]\!]$, we have $|P_h| = q_h - p_h + 3no(\mathcal{B}_h) - 3ni(\mathcal{B}_h) = t_h - s_h$. Moreover, the sets P_h, $h \in [\![1\,;\, l]\!]$, form a partition of the set $[\![1\,;\, n]\!]$.

We can now create the permutation π_I. The image of 0 is 0, and for each h_0 from 1 to l, we define the restriction of π_I over $[\![s_{h_0} + 1\,;\, t_{h_0}]\!]$ as a permutation of

P_{h_0}, with the constraint that $\pi_I(t_{h_0}) = q_{h_0}$. Note that, if this condition is fulfilled, then we can assume $\pi_I(s_{h_0}) = p_{h_0}$, since, if $h_0 = 1$, $\pi_I(s_1) = \pi_I(0) = 0 = p_1$, and if $h_0 > 1$, $\pi_I(s_{h_0}) = \pi_I(t_{h_0-1}) = q_{h_0-1} = p_{h_0}$.

The definition of π_I over each interval $[\![s_{h_0} + 1\,;\, t_{h_0}]\!]$, where $\mathcal{B}_{h_0}$ is one of the blocks copy, and, or, var, is given in the following tables. We write $s = s_{h_0}$ and $p = p_{h_0}$. We give the line $\psi^{-1}(u)$ as a reminder of the definition of each block. We also add a column for $u = s$ as a reminder of the fact that $\pi_I(s) = p$.

– If $\mathcal{B}_{h_0}$ is a block of the kind $[A_1, A_2] = \mathsf{copy}(A)$, we write $\alpha_1, \beta_1, \alpha_2, \beta_2$ for the respective values of $\alpha(A_1), \beta(A_1), \alpha(A_2), \beta(A_2)$.

u =	s	$s{+}1$	$s{+}2$	$s{+}3$	$s{+}4$	$s{+}5$	$s{+}6$	$s{+}7$	$s{+}8$	$s{+}9$
$\pi_I(u)$ =	p	$\alpha_1{+}2$	$p{+}8$	$p{+}4$	$p{+}3$	$\alpha_2{+}2$	$p{+}7$	$\beta_1{+}1$	$\alpha_1{+}1$	$p{+}6$
$\psi^{-1}(u)$=		a	y_1	e	z	d	y_2	x_1	b_1	c

u =	$s{+}10$	$s{+}11$	$s{+}12$
$\pi_I(u)$ =	$\beta_2{+}1$	$\alpha_2{+}1$	$p{+}9$
$\psi^{-1}(u)$=	x_2	b_2	f

– If $\mathcal{B}_{h_0}$ is a block of the kind $A = \mathsf{and}(A_1, A_2)$, we write α, β for the respective values of $\alpha(A), \beta(A)$.

u =	s	$s{+}1$	$s{+}2$	$s{+}3$	$s{+}4$	$s{+}5$	$s{+}6$	$s{+}7$	$s{+}8$	$s{+}9$
$\pi_I(u)$ =	p	$p{+}14$	$p{+}7$	$p{+}3$	$p{+}13$	$p{+}9$	$p{+}6$	$\alpha{+}2$	$p{+}12$	$p{+}11$
$\psi^{-1}(u)$=		a_1	e	z_1	a_2	c_1	z_2	d	y	c_2

u =	$s{+}10$	$s{+}11$	$s{+}12$
$\pi_I(u)$ =	$\beta{+}1$	$\alpha{+}1$	$p{+}15$
$\psi^{-1}(u)$=	x	b	f

– If $\mathcal{B}_{h_0}$ is a block of the kind $A = \mathsf{or}(A_1, A_2)$, we write α, β for the respective values of $\alpha(A), \beta(A)$.

u =	s	$s{+}1$	$s{+}2$	$s{+}3$	$s{+}4$	$s{+}5$	$s{+}6$	$s{+}7$	$s{+}8$	$s{+}9$
$\pi_I(u)$ =	p	$p{+}7$	$p{+}13$	$p{+}3$	$p{+}9$	$\alpha{+}2$	$p{+}12$	$p{+}11$	$\beta{+}1$	$\alpha{+}1$
$\psi^{-1}(u)$=		a_1	b'	z_1	a_2	d	y	a'	x	b

u =	$s{+}10$	$s{+}11$	$s{+}12$	$s{+}13$	$s{+}14$	$s{+}15$
$\pi_I(u)$ =	$p{+}16$	$p{+}6$	$p{+}15$	$p{+}10$	$p{+}8$	$p{+}18$
$\psi^{-1}(u)$=	f	z_2	c_1	e	c'	c_2

– If $\mathcal{B}_{h_0}$ is a block of the kind $[A_1, A_2] = \mathsf{var}(A)$, we write $\alpha_1, \beta_1, \alpha_2, \beta_2$ for the respective values of $\alpha(A_1), \beta(A_1), \alpha(A_2), \beta(A_2)$.

u =	s	$s{+}1$	$s{+}2$	$s{+}3$	$s{+}4$	$s{+}5$	$s{+}6$	$s{+}7$	$s{+}8$	$s{+}9$
$\pi_I(u)$ =	p	$\alpha_1{+}2$	$p{+}5$	$p{+}3$	$\alpha_2{+}2$	$p{+}12$	$p{+}1$	$p{+}14$	$p{+}4$	$\beta_1{+}1$
$\psi^{-1}(u)$=		d_1	y_1	a	d_2	y_2	e_1	a'	e_2	x_1

u =	$s{+}10$	$s{+}11$	$s{+}12$	$s{+}13$	$s{+}14$	$s{+}15$	$s{+}16$	$s{+}17$	$s{+}18$
$\pi_I(u)$ =	$\alpha_1{+}1$	$p{+}13$	$p{+}9$	$p{+}8$	$p{+}2$	$p{+}11$	$\beta_2{+}1$	$\alpha_2{+}1$	$p{+}15$
$\psi^{-1}(u)$=	b_1	f_1	c'	z	b'	c	x_2	b_2	f_2

These definitions are obtained by applying the following rules, until $\pi_I(u)$ is defined for all $u \in [\![s_{h_0} + 1\,;\, t_{h_0}]\!]$.

$$\begin{aligned}
&\text{For all input variables of } \mathcal{B}_{h_0}\ A = [(a,b,c),(x,y,z)],\\
&\pi_I(\psi(z)) = \alpha(A) + 3 && (R_1)\\
&\pi_I(\psi(c)) = \beta(A) + 2 && (R_2)\\
&\text{For all output variables of } \mathcal{B}_{h_0}\ A = [(a,b,c),(x,y,z)],\\
&\pi_I(\psi(x)) = \beta(A) + 1 && (R_3)\\
&\pi_I(\psi(b)) = \alpha(A) + 1 && (R_4)\\
&\forall u \in [\![s_{h_0} + 1\,;\, t_{h_0}]\!] \text{ such that } succ_{I_\phi}^{-1}(u) \in [\![s_{h_0} + 1\,;\, t_{h_0}]\!]\\
&\pi_I(u) = \pi_I(succ_I^{-1}(u) - 1) + 1 && (R_5)
\end{aligned}$$

A simple case by case analysis shows that rules (R_1) and (R_2) indeed apply to every input variable, and rules (R_3) and (R_4) apply to every output variable. Moreover the following properties are also satisfied:

$$\begin{aligned}
&\text{Rule } (R_5) \text{ applies to every } u \in [\![s_{h_0+1}\,;\, t_{h_0}]\!] \text{ such that}\\
&u \notin \{\psi(b), \psi(c), \psi(x), \psi(z) \mid A = [(a,b,c),(x,y,z)] \text{ is input/output of } \mathcal{B}_{h_0}\}. && (P_1)
\end{aligned}$$

$$\pi_I \text{ defines a bijection from } [\![s_{h_0} + 1\,;\, t_{h_0}]\!] \text{ to } P_{h_0} \text{ such that } \pi_I(t_{h_0}) = q_{h_0} \qquad (P_2)$$

$$\begin{aligned}
&\text{For all input variables of } \mathcal{B}_{h_0}\ A = [(a,b,c),(x,y,z)],\\
&\pi_I(\psi(a) - 1) = \alpha(A) && (P_3)\\
&\pi_I(\psi(z) - 1) = \beta(A) && (P_4)\\
&\text{For all output variables of } \mathcal{B}_{h_0}\ A = [(a,b,c),(x,y,z)],\\
&\pi_I(\psi(y) - 1) = \alpha(A) + 2 && (P_5)\\
&\pi_I(\psi(b) - 1) = \beta(A) + 1 && (P_6)
\end{aligned}$$

Now that we have defined the permutation π_I, we need to show that π_I is equivalent to I. Following Definition 1.10, we have $\pi_I(0) = 0$. Then, $L = [\![1\,;\, n]\!]$, so let us fix any $u \in [\![1\,;\, n]\!]$, and verify that $\pi_I(u) = \pi_I(succ_I^{-1}(u) - 1) + 1$. Let h be the integer such that $u \in [\![s_h + 1\,;\, t_h]\!]$.

First, consider the most general case, and assume that there is no variable $A = [(a,b,c),(x,y,z)]$ such that $u \in \{\psi(b), \psi(c), \psi(x), \psi(z)\}$. Note that this case includes $u = \psi(d)$, where d is part of some internal triple. Then, by Property (P_1), we know that Rule (R_5) applies to u, hence we directly have $\pi_I(u) = \pi_I(succ_I^{-1}(u) - 1) + 1$.

Suppose now that there exists some variable $A = [(a,b,c),(x,y,z)]$ such that $u \in \{\psi(b), \psi(c), \psi(x), \psi(z)\}$. Then Rules (R_1) and (R_2), and Properties (P_3) and (P_4) apply in the target block of A. Also, Rules (R_3) and (R_4), and Properties (P_5) and (P_6) apply in the source block of A. Combining all these equations, we have:

$$\begin{aligned}
&\pi_I(\psi(b)) = \alpha(A) + 1 = \pi_I(\psi(a) - 1) + 1 && \text{by } (R_3) \text{ and } (P_3)\\
&\pi_I(\psi(c)) = \beta(A) + 2 = \pi_I(\psi(b) - 1) + 1 && \text{by } (R_2) \text{ and } (P_5)\\
&\pi_I(\psi(x)) = \beta(A) + 1 = \pi_I(\psi(z) - 1) + 1 && \text{by } (R_4) \text{ and } (P_4)\\
&\pi_I(\psi(z)) = \alpha(A) + 3 = \pi_I(\psi(y) - 1) + 1 && \text{by } (R_1) \text{ and } (P_6)
\end{aligned}$$

For $u = \psi(b)$ (respectively, $\psi(c), \psi(x), \psi(z)$), we have $succ_I^{-1}(u) = \psi(a)$ (resp. $\psi(b), \psi(z), \psi(y)$). Hence, in all four cases, we have $\pi_I(u) = \pi_I(succ_I^{-1}(u) - 1) + 1$, which completes the proof that π_I is equivalent to I.

□

With the previous theorem, we now have all the necessary ingredients to prove the main result of this chapter.

Theorem 1.21. *The* Sorting by Transpositions *problem is* NP*-hard.*

Proof. The reduction from SAT is as follows: given any instance ϕ of SAT, create a 3DT-instance I_ϕ, being an assembling of basic blocks, which is 3DT-collapsible iff ϕ is satisfiable (Theorem 1.19). Then create a permutation π_{I_ϕ} equivalent to I_ϕ (Theorem 1.20, note that π_{I_ϕ} is a 3-permutation). The above two steps can be done in polynomial time. Finally, set $k = d_b(\pi_{I_\phi})/3 = n/3$. We then have:

$$\begin{aligned} \phi \text{ is satisfiable} &\Leftrightarrow I_\phi \text{ is 3DT-collapsible} \\ &\Leftrightarrow d_t(\pi_{I_\phi}) = k \text{ (by Theorem 1.9, since } \pi_{I_\phi} \sim I_\phi) \\ &\Leftrightarrow d_t(\pi_{I_\phi}) \leq k \text{ (by Property 1.4).} \end{aligned}$$

□

Corollary 1.22. *The following decision problem from [55] is* NP*-hard: given a permutation π of $[\![0\,;\,n]\!]$, is the equality $d_t(\pi) = d_b(\pi)/3$ satisfied?*

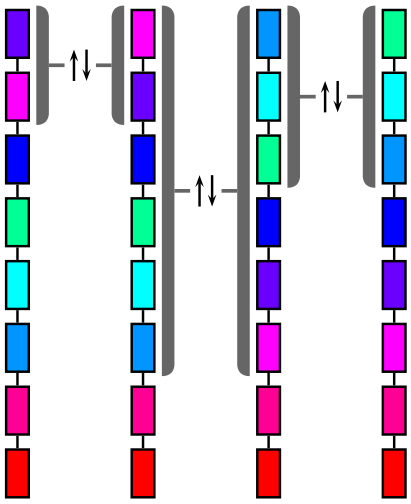
Conclusion

In this chapter, we have addressed a 15-year open question as we proved that the Sorting by Transpositions problem is NP-hard. However, a number of questions remain open. For instance, does this problem admit a polynomial-time approximation scheme? We note that the reduction we have provided does not answer this question, since it is not a gap-preserving reduction. Indeed, in our reduction, if a formula ϕ is not satisfiable, it can be seen that we have $d_t(\pi_{I_\phi}) = d_b(\pi_{I_\phi})/3 + 1$.

Also, do there exist some relevant parameters for which the problem is fixed parameter tractable? A parameter that comes to mind when dealing with the transposition distance is the length of the exchanged factors (i.e., the value $\max\{j-i, k-j\}$ for a transposition $\tau_{i,j,k}$). Does the problem become tractable if this parameter is bounded? In fact, the answer to this question is negative if only the length of the smallest factor, $\min\{j-i, k-j\}$, is bounded: in our reduction, this parameter is upper bounded by 6 for every transposition needed to sort π_{I_ϕ}, independently of the formula ϕ.

2

Sorting by Prefix Reversals



Sorting by Prefix Reversals (SBPR) is a problem better known under the name *Pancake Flipping*: rearrange a stack of pancakes of different sizes (that is, a permutation) into a pyramidal stack, when the only allowed operation is to insert a spatula anywhere in the stack and to flip the pancakes above it (that is, to perform a prefix reversal). Computing the optimal scenario has been an intriguing combinatorial problem for the last three decades.

In this chapter, we show that the Pancake Flipping problem is NP-hard.

The results in this chapter have been presented as an invited talk at the *Algorithms and Permutations* workshop (A&P 2012, Paris), and at the *37th International Symposium on Mathematical Foundations of Computer Science* (MFCS 2012, Bratislava [34]). They are currently submitted for publication in the *Journal of Computer and System Sciences* (JCSS [37]).

Introduction

The pancake problem was stated in [64] as follows:

> The chef in our place is sloppy, and when he prepares a stack of pancakes they come out all different sizes. Therefore, when I deliver them to a customer, on the way to the table I rearrange them (so that the smallest winds up on top, and so on, down to the largest at the bottom) by grabbing several from the top and flipping them over, repeating this (varying the number I flip) as many times as necessary. If there are n pancakes, what is the maximum number of flips (as a function of n) that I will ever have to use to rearrange them?

Stacks of pancakes are represented by permutations, and a flip consists in reversing a prefix of any length. The previous puzzle yields two entangled problems:

- Designing an algorithm that sorts any permutation with a minimum number of flips (this optimization problem is called SBPR, for Sorting By Prefix Reversals). See Figure 2.1 for an example.
- Computing $f(n)$, the maximum number of flips required to sort a permutation of size n (the diameter of the so-called *pancake network*).

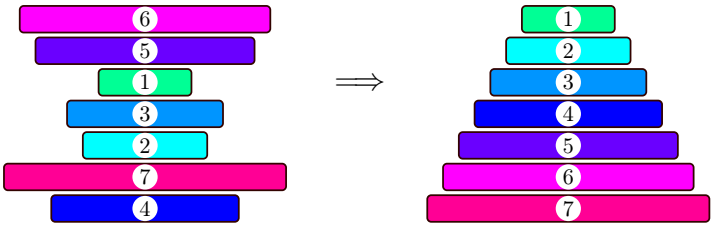
Gates and Papadimitriou [76] introduced the *burnt* variant of the problem: the pancakes are two-sided, and an additional constraint requires the pancakes to end with the unburnt side up. The diameter of the corresponding *burnt pancake network* is denoted $g(n)$. A number of studies [50, 57, 59, 76, 88, 87, 100] have aimed at determining more precisely the values of $f(n)$ and $g(n)$, with the following results:

- $f(n)$ and $g(n)$ are known exactly for $n \leq 19$ and $n \leq 17$, respectively [57].
- $15n/14 \leq f(n) \leq 18n/11 + O(1)$ [87, 50].
- $\lfloor (3n+3)/2 \rfloor \leq g(n) \leq 2n - 6$ [57] (upper bound for $n \geq 16$).

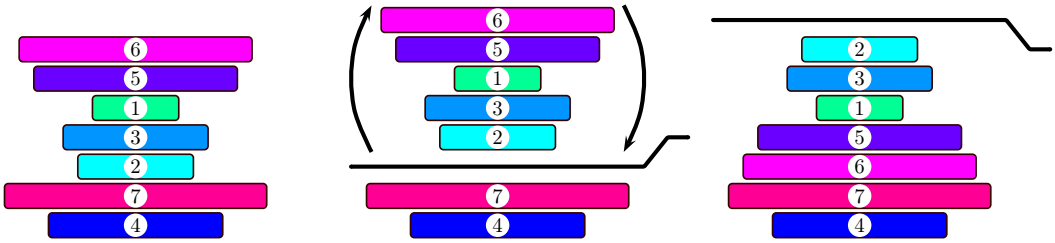
Considering SBPR, 2-approximation algorithms have been designed both for the burnt and unburnt variants [59, 70]. Moreover, Labarre and Cibulka [100] have characterized a subclass of signed permutations, called *simple permutations*, that can be sorted in polynomial time.

The pancake problems have various applications. For instance, the pancake network, having both a small degree and diameter, is of interest in parallel computing [1, 112]. The algorithmic aspect, i.e. the sorting problem, corresponds exactly to the rearrangement distance where allowed operations are prefix reversals, and burnt and unburnt variants correspond respectively to signed and unsigned genome models. This problem thus attracted attention from the comparative genomics community, especially for its similarities with the now well-known Sorting By Reversals [11] problem, which admits a polynomial-time exact algorithm [84] for the signed case, and a 1.375-approximation [19] for the APX-hard unsigned case [20].

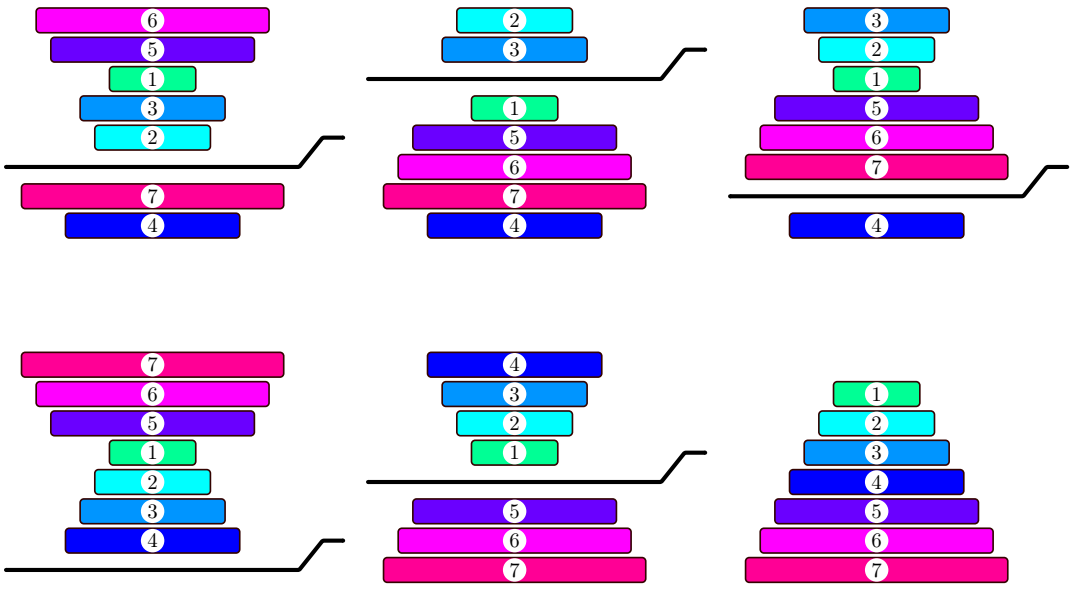
In this chapter, we prove that the SBPR problem is NP-hard (in its unburnt, or unsigned, variant), thus answering a question which has remained open for several decades. We in fact prove a stronger result: it is known that the number of breakpoints of a permutation (that is, the number of pairs of consecutive elements that are not consecutive in the identity permutation) is a lower bound on the number of flips necessary to sort a permutation. We show that deciding whether this bound is tight is already NP-hard.



(a) Goal: transform a permutation (e.g., $\langle 6, 5, 1, 3, 2, 7, 4 \rangle$) into the Identity $\mathcal{I}_7^1$.



(b) A 5-element *flip*: $\langle 6, 5, 1, 3, 2, 7, 4 \rangle$ becomes $\langle 2, 3, 1, 5, 6, 7, 4 \rangle$. The flip limit is marked with a spatula symbol.



(c) The sequence of 5 flips required to sort $\langle 6, 5, 1, 3, 2, 7, 4 \rangle$. It is optimal since the breakpoint distance is 5. It is also, in fact, the only optimal sequence for this permutation.

Figure 2.1: Illustration of the Sorting by Prefix Inversions problem.

2.1 Notations

Let n be an integer. The input of SBPR consists of permutations of $[\![1\,;\,n]\!]$, we construct such a permutation by concatenating duplication-free, unsigned sequences. When there is no ambiguity, we use the same notation for a sequence and the set of elements it contains. We use upper case for sets and sequences, and lower case for elements.

Consider a sequence S of length n, $S = \langle x_1, x_2, \ldots, x_n \rangle$. Element x_1 is said to be the *head element* of S. Sequence S has a *breakpoint* at position r, $1 \leq r < n$ if $x_r \notin \{x_{r+1} - 1, x_{r+1} + 1\}$, and a *breakpoint* at position n if $x_n \neq n$. We write $d_b(S)$ for the number of breakpoints of S. Note that having $x_1 \neq 1$ does not directly count as a breakpoint, and that $d_b(S) \leq n$ for any sequence of length n. For any $p \leq q \in \mathbb{N}$, we write $\mathcal{I}_q^p$ for the sequence $\langle p, p+1, p+2, \ldots, q \rangle$; $\mathcal{I}_n^1$ is the *identity*. Given a sequence of any length $S = \langle x_1, x_2, \ldots, x_k \rangle$, we write ${}^\star S$ for the sequence obtained by reversing S: ${}^\star S = \langle x_k, x_{k-1}, \ldots, x_1 \rangle$. Given an integer p, we write $p + S = \langle p + x_1, p + x_2, \ldots, p + x_k \rangle$.

The *flip* of length r is the operation that consists in reversing the r first elements of the sequence. It transforms

$$\begin{aligned} & S = \langle x_1, x_2, \ldots, x_r, \quad x_{r+1}, \ldots, x_n \rangle \\ \text{into} \quad & S' = \langle x_r, x_{r-1}, \ldots, x_1, \quad x_{r+1}, \ldots, x_n \rangle. \end{aligned}$$

Note that the flip of length 1 does not modify S, and the flip of length n transforms S into ${}^\star S$. Moreover, since a flip of length r cannot add or remove breakpoints other than in position r, we have the following easy property.

Property 2.1. *Given a sequence S' obtained from a sequence S by performing one flip, we have $d_b(S') - d_b(S) \in \{-1, 0, 1\}$.*

A flip from S to S' is said to be *efficient* if $d_b(S') = d_b(S) - 1$, and we reserve the notation $S \to S'$ for such flips. A sequence of size n, different from the identity, is a *deadlock* if it yields no efficient flip, and we write $S \to \bot$. As a convention, we place a specific separator $\mid$ in a sequence at the positions corresponding to possible efficient flips: there are at most two of them, and at least one if the sequence is neither a deadlock nor the identity. A *path* is a series of flips, it is *efficient* if each flip it contains is efficient. A sequence S is *efficiently sortable* if there exists an efficient path from S to the identity (equivalently, if it can be sorted in $d_b(S)$ flips). See for example Figure 2.2.

Let S be a sequence different from the identity, and $\mathbb{T}$ be a set of sequences. We write $S \Longrightarrow \mathbb{T}$ if both following conditions are satisfied:

1. for each $T \in \mathbb{T}$, there exists an efficient path from S to T.
2. for each efficient path from S to the identity, there exists a sequence $T \in \mathbb{T}$ such that the path goes through T.

If $\mathbb{T}$ consists of a single element ($\mathbb{T} = \{T\}$), we may write $S \Longrightarrow T$ instead of $S \Longrightarrow \{T\}$. Note that Condition 1 is trivial if $\mathbb{T} = \emptyset$, and Condition 2 is trivial if there is no efficient path from S to $\mathcal{I}_n^1$. Given a sequence S, there can be several different sets $\mathbb{T}$ such that $S \Longrightarrow \mathbb{T}$. However, two are especially relevant:

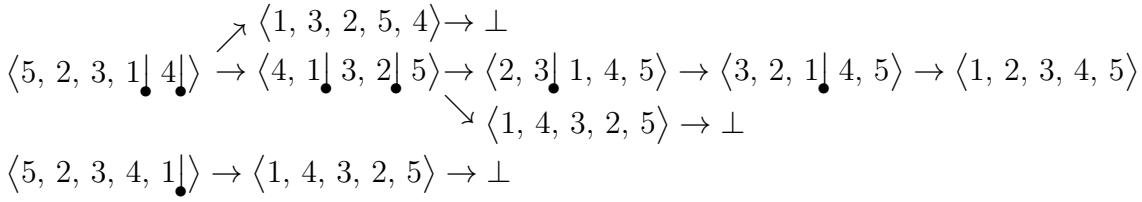


Figure 2.2: Examples of efficient flips. Sequence $\langle 5, 2, 3, 1, 4\rangle$ is efficiently sortable (in four flips), but $\langle 5, 2, 3, 4, 1\rangle$ is not.

Property 2.2. *Given any sequence $S \neq \mathcal{I}_n^1$,*

$$\begin{aligned} S \Longrightarrow \mathcal{I}_n^1 \quad &\Leftrightarrow \quad S \textit{ is efficiently sortable.} \\ S \Longrightarrow \emptyset \quad &\Leftrightarrow \quad S \textit{ is not efficiently sortable.} \end{aligned}$$

Proof. For $S \Longrightarrow \mathcal{I}_n^1$: Condition 1 is true iff there exists an efficient path from S to the identity, that is S is efficiently sortable. Condition 2 is always true.

For $S \Longrightarrow \emptyset$: Condition 1 is always true. If there exists at least one efficient path from S to $\mathcal{I}_n^1$, then, since there exists no sequence $T \in \emptyset$, Condition 2 cannot be true. Hence Condition 2. is false when there exists an efficient path from S to the identity and true otherwise, so it is equivalent to the fact that S is not efficiently sortable. □

The following property, yielding a certain transitivity of the $\Longrightarrow$ relation, is easily deduced from the definition.

Property 2.3. *If $S \Longrightarrow \{S_1, S_2, \ldots, S_k\}$, $S_1 \Longrightarrow \mathbb{T}_1$, $S_2 \Longrightarrow \mathbb{T}_2$, ... and $S_k \Longrightarrow \mathbb{T}_k$, then $S \Longrightarrow \mathbb{T}_1 \cup \mathbb{T}_2 \cup \ldots \cup \mathbb{T}_k$.*

In particular for $k = 1$, if $S \Longrightarrow S_1$ and $S_1 \Longrightarrow \mathbb{T}_1$, then $S \Longrightarrow \mathbb{T}_1$.

2.2 Low-level Gadgets

The reduction from the satisfiability problem 3-SAT (Section 2.4) uses a number of gadget sequences in order to simulate boolean variables and clauses with subsequences. They are organized in two levels (where low-level gadgets are directly defined by sequences of integers, and high-level gadgets are defined using a pattern of low-level gadgets). For each gadget we define, we derive a property characterizing the efficient paths that can be followed if some part of the gadget appears at the head of a sequence.

We have not aimed at providing the smallest possible gadgets (the overall reduction for a formula containing l variables and k clauses creates a stack of $31l + 98k$ elements with $16l + 50k$ breakpoints), and we preferred straightforward proofs and easy-to-combine gadgets over short sequences. A rough analysis shows that the final stack size could easily be reduced to $22l+71k$, with the same number of breakpoints.

2.2.1 Dock

The dock gadget is the simplest gadget of our construction. Its only goal is to store sequences of the kind ${}^\star\mathcal{I}_q^{p+1}$ (with $p < q$) taken from the head of the sequence, without "disturbing" any other part.

Definition 2.1 (Dock Gadget)**.** *Given two integers p and q with $p < q$, the* dock *for ${}^\star\mathcal{I}_q^{p+1}$ is the sequence $\mathsf{Dock}(p, q) = D$, where*

$$D = \langle p-1,\ p,\ q+1,\ q+2 \rangle.$$

It has the following property:

Property 2.4. *Let p and q be any integers with $p < q$, $D = \mathsf{Dock}(p, q)$, and X and Y be any sequences. We have*

$$\langle {}^\star\mathcal{I}_q^{p+1},\ X,\ D,\ Y \rangle \Longrightarrow \langle X,\ \mathcal{I}_{q+2}^{p-1},\ Y \rangle$$

Proof. An efficient path from $\langle {}^\star\mathcal{I}_q^{p+1},\ X,\ D,\ Y \rangle$ to $\langle X,\ \mathcal{I}_{q+2}^{p-1},\ Y \rangle$ is given below.

$$\begin{aligned}
\langle {}^\star\mathcal{I}_q^{p+1},\ X,\ D,\ Y \rangle &= \langle q,\ q-1,\ \ldots,\ p+2,\ p+1,\ X,\ p-1,\ p \rfloor\ q+1,\ q+2,\ Y \rangle \\
&\to \langle p,\ p-1,\ {}^\star X \rfloor\ p+1,\ p+2,\ \ldots,\ q-1,\ q,\ q+1,\ q+2,\ Y \rangle \\
&\to \langle X,\ p-1,\ p,\ p+1,\ p+2,\ \ldots,\ q-1,\ q,\ q+1,\ q+2,\ Y \rangle \\
&= \langle X,\ \mathcal{I}_{q+2}^{p-1},\ Y \rangle
\end{aligned}$$

For each sequence in the path, we apply the only possible efficient flip (there is no breakpoint before $q-1$ in the first sequence, and no breakpoint before $p-1$ in the second). Hence, every efficient path between $\langle {}^\star\mathcal{I}_q^{p+1},\ X,\ D,\ Y \rangle$ and $\mathcal{I}_n^1$ (if such a path exists) begins with these two flips, and goes through $\langle X,\ \mathcal{I}_{q+2}^{p-1},\ Y \rangle$. □

2.2.2 Lock

A lock gadget contains three parts: a sequence which is the lock itself, a key element that "opens" the lock, and a test element that checks whether the lock is open.

Definition 2.2 (Lock Gadget)**.** *For any integer p, $\mathsf{Lock}(p)$ is defined by $\mathsf{Lock}(p) = (key, test, L)$, where*

$$\begin{aligned}
key &= p + 10 \qquad & test &= p + 7 \\
L &= p + \langle 1,\ 2,\ 9,\ 8,\ 5,\ 6,\ 4,\ 3,\ 11,\ 12 \rangle
\end{aligned}$$

Given a lock $(key, test, L) = \mathsf{Lock}(p)$, we write

$$L^o = p + \langle 1,\ 2,\ 3,\ 4,\ 6,\ 5,\ 8,\ 9,\ 10,\ 11,\ 12 \rangle.$$

Sequences L and L^o represent the lock when it is closed and open, respectively. If a sequence containing a closed lock has *key* for head element, then efficient flips put the lock in open position. If it has *test* for head element, then it is a deadlock iff the lock is closed.

Property 2.5. *Let p be any integer, $(key, test, L) = \mathsf{Lock}(p)$, and X and Y be any sequences. We have*

$$\textbf{a.} \quad \langle key,\, X,\, L,\, Y\rangle \Longrightarrow \langle X,\, L^o,\, Y\rangle$$
$$\textbf{b.} \quad \langle test,\, X,\, L^o,\, Y\rangle \Longrightarrow \langle X,\, \mathcal{I}^{p+1}_{p+12},\, Y\rangle$$
$$\textbf{c.} \quad \langle test,\, X,\, L,\, Y\rangle \to \bot$$

Proof. The possible efficient paths from **(a.)** $\langle key, X, L, Y\rangle$, **(b.)** $\langle test, X, L^o, Y\rangle$ and **(c.)** $\langle test, X, L, Y\rangle$ are the following. Note that for readability reasons, the proof is given for $p = 0$; it can obviously be extended to any value of p (each element would then be increased by p).

a. $\langle key, X, L, Y\rangle = \langle 10, X, 1, 2\bullet 9, 8, 5, 6, 4, 3\bullet 11, 12, Y\rangle$

$S_1 \swarrow \quad \searrow S_2$ (where sequences S_1 and S_2 are described below)

$S_1 = \langle 2, 1, {}^\star X, 10, 9, 8, 5, 6, 4, 3, 11, 12, Y\rangle$

$\to \bot$

$S_2 = \langle 3, 4, 6, 5, 8, 9\bullet 2, 1, {}^\star X, 10, 11, 12, Y\rangle$

$\to \langle 9, 8, 5, 6, 4, 3, 2, 1, {}^\star X\bullet 10, 11, 12, Y\rangle$

$\to \langle X, 1, 2, 3, 4, 6, 5, 8, 9, 10, 11, 12, Y\rangle$

$= \langle X, L^o, Y\rangle$

b. $\langle test, X, L^o, Y\rangle = \langle 7, X, 1, 2, 3, 4\bullet 6, 5\bullet 8, 9, 10, 11, 12, Y\rangle$

$S_3 \swarrow \quad \searrow S_4$

$S_3 = \langle 4, 3, 2, 1, {}^\star X, 7, 6, 5, 8, 9, 10, 11, 12, Y\rangle$

$\to \bot$

$S_4 = \langle 5, 6\bullet 4, 3, 2, 1, {}^\star X, 7, 8, 9, 10, 11, 12, Y\rangle$

$\to \langle 6, 5, 4, 3, 2, 1, {}^\star X\bullet 7, 8, 9, 10, 11, 12, Y\rangle$

$\to \langle X, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, Y\rangle$

$= \langle X, \mathcal{I}^1_{12}, Y\rangle$

c. $\langle test, X, L, Y\rangle = \langle 7, X, 1, 2, 9, 8, 5, 6, 4, 3, 11, 12, Y\rangle$

$\to \bot$

□

Overview of the reduction

With the lock and dock gadget defined, we can give an overall view of how the different gadgets are used in the constructed sequence.

We use locks to emulate literals of a boolean formula: one lock is created for each literal; each boolean variable is "given the keys" of the corresponding literals, and each clause holds three test elements. In a first step, variable gadgets open the locks corresponding to literals which are assigned "true". In a second step, each clause gadget tests one of its three locks, thus verifying that one literal in the clause is true. The formula is true if and only if there is an efficient path completing these two steps.

In order to ensure that such an efficient path can continue all the way to the identity permutation, two additional steps are required. The variable gadgets are again put forward and open the closed locks, then, with the clause gadgets, all remaining locks are tested. Docks ensure that once a variable or clause gadget is "done", it is stored at its final location. Thus, an efficient path completing all four steps necessarily leads to the identity permutation.

The next two low-level gadgets, hooks and forks, are the building blocks of high-level variable and clause gadgets.

2.2.3 Hook

A hook gadget contains four parts: two sequences used as delimiters, a *take* element that takes the interval between the delimiters and places it in head, and a *put* element that does the reverse operation. Thus, the sequence between the delimiters can be stored anywhere until it is called to the head by *take*, and then can be stored back using *put*.

Definition 2.3 (Hook Gadget)**.** *For any integer p,* $\mathsf{Hook}(p)$ *is defined by* $\mathsf{Hook}(p) = (take, put, G, H)$*, where*

$$take = p + 10 \qquad put = p + 7$$
$$G = p + \langle 3, 4\rangle \qquad H = p + \langle 12, 11, 6, 5, 9, 8, 2, 1\rangle.$$

Given a hook $(take, put, G, H) = \mathsf{Hook}(p)$*, we write*

$$G' = p + \langle 12, 11, 6, 5, 4, 3\rangle \qquad H' = p + \langle 10, 9, 8, 2, 1\rangle$$
$$G'' = p + \langle 3, 4, 5, 6, 7\rangle \qquad H'' = p + \langle 12, 11, 10, 9, 8, 2, 1\rangle.$$

Property 2.6. *Let p be an integer,* $(take, put, G, H) = \mathsf{Hook}(p)$*, and X, Y and Z be any sequences. We have*

a. $\langle take, X, G, Y, H, Z\rangle \Longrightarrow \langle Y, G', {}^\star X, H', Z\rangle$

b. $\langle put, X, G', {}^\star Y, H', Z\rangle \Longrightarrow \langle Y, G'', X, H'', Z\rangle$

c. $\langle G'', X, H'', Y\rangle \Longrightarrow \langle X, {}^\star\mathcal{I}^{p+1}_{p+12}, Y\rangle$

Proof. The possible efficient paths from each sequence are the following (for $p = 0$).

a.
$$\begin{aligned}\langle take, X, G, Y, H, Z\rangle &= \langle 10, X, 3, 4, Y, 12, 11, 6, 5\vert\, 9, 8, 2, 1, Z\rangle\\ &\to \langle 5, 6, 11, 12, {}^\star Y\vert\, 4, 3, {}^\star X, 10, 9, 8, 2, 1, Z\rangle\\ &\to \langle Y, 12, 11, 6, 5, 4, 3, {}^\star X, 10, 9, 8, 2, 1, Z\rangle\\ &= \langle Y, G', {}^\star X, H', Z\rangle\end{aligned}$$

b.
$$\begin{aligned}\langle put, X, G', {}^\star Y, H', Z\rangle &= \langle 7, X, 12, 11\vert\, 6, 5, 4, 3, {}^\star Y, 10, 9, 8, 2, 1, Z\rangle\\ &\to \langle 11, 12, {}^\star X, 7, 6, 5, 4, 3, {}^\star Y\vert\, 10, 9, 8, 2, 1, Z\rangle\\ &\to \langle Y, 3, 4, 5, 6, 7, X, 12, 11, 10, 9, 8, 2, 1, Z\rangle\\ &= \langle Y, G'', X, H'', Z\rangle\end{aligned}$$

c.
$$\begin{aligned}\langle G'', X, H'', Y\rangle &= \langle 3, 4, 5, 6, 7, X, 12, 11, 10, 9, 8\vert\, 2, 1, Y\rangle\\ &\to \langle 8, 9, 10, 11, 12, {}^\star X\vert\, 7, 6, 5, 4, 3, 2, 1, Y\rangle\\ &\to \langle X, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, Y\rangle\\ &= \langle X, {}^\star\mathcal{I}^1_{12}, Y\rangle\end{aligned}$$

□

2.2.4 Fork

A fork gadget implements choices. It contains two parts delimiting a sequence X. Any efficient path encountering a fork gadget follows one of two tracks, where either X or $^\star X$ appears at the head of the sequence at some point. Sequence X would typically contain a series of triggers for various gadgets (*key*, *take*, etc.), so that X and $^\star X$ differ in the order in which the gadgets are triggered.

Definition 2.4 (Fork Gadget)**.** *For any integer p, $\mathsf{Fork}(p)$ is defined by $\mathsf{Fork}(p) = (E, F)$, where*

$$E = p + \langle 11, 8, 7, 3\rangle \quad F = p + \langle 10, 9, 6, 12, 13, 4, 5, 15, 14, 2, 1\rangle.$$

Given a fork $(E, F) = \mathsf{Fork}(p)$, we write

$$\begin{aligned} F^1 &= p + \langle 10, 9, 6, 7, 8, 11, 12, 13, 14, 15, 5, 4, 3, 2, 1\rangle \\ F^2 &= p + \langle 3, 7, 8, 11, 10, 9, 6, 12, 13, 4, 5, 15, 14, 2, 1\rangle \end{aligned}$$

Property 2.7. *Let p be an integer, $(E, F) = \mathsf{Fork}(p)$, and X, Y be any sequences. We have*

a. $\langle E, X, F, Y\rangle \Longrightarrow \{\langle X, F^1, Y\rangle, \langle {}^\star X, F^2, Y\rangle\}$
b. $\langle F^1, Y\rangle \Longrightarrow \langle {}^\star\mathcal{I}^{p+1}_{p+15}, Y\rangle$
c. $\langle F^2, Y\rangle \Longrightarrow \langle {}^\star\mathcal{I}^{p+1}_{p+15}, Y\rangle$

Proof. The possible efficient paths from **(a.)** $\langle E, X, F, Y\rangle$, **(b.)** $\langle F^1, Y\rangle$ and **(c.)** $\langle F^2, Y\rangle$ are the following (for $p = 0$).

a. $\langle E, X, F, Y\rangle = \langle 11, 8, 7, 3, X\rangle\!\!\bullet\, 10, 9, 6\rangle\!\!\bullet\, 12, 13, 4, 5, 15, 14, 2, 1, Y\rangle$

$S_1 \swarrow \quad \searrow S_2$

$S_1 = \langle {}^\star X, 3, 7, 8, 11, 10, 9, 6, 12, 13, 4, 5, 15, 14, 2, 1, Y\rangle$
$= \langle {}^\star X, F^2, Y\rangle$

$S_2 = \langle 6, 9, 10, {}^\star X, 3\rangle\!\!\bullet\, 7, 8, 11, 12, 13, 4, 5, 15, 14, 2, 1, Y\rangle$
$\to \langle 3, X, 10, 9, 6, 7, 8, 11, 12, 13\rangle\!\!\bullet\, 4, 5, 15, 14\rangle\!\!\bullet\, 2, 1, Y\rangle$

$S_3 \swarrow \quad \searrow S_4$

$S_3 = \langle 13, 12, 11, 8, 7, 6, 9, 10, {}^\star X, 3, 4, 5, 15, 14, 2, 1, Y\rangle$
$\to \bot$

$S_4 = \langle 14, 15, 5, 4\rangle\!\!\bullet\, 13, 12, 11, 8, 7, 6, 9, 10, {}^\star X, 3, 2, 1, Y\rangle$
$\to \langle 4, 5, 15, 14, 13, 12, 11, 8, 7, 6, 9, 10, {}^\star X\rangle\!\!\bullet\, 3, 2, 1, Y\rangle$
$\to \langle X, 10, 9, 6, 7, 8, 11, 12, 13, 14, 15, 5, 4, 3, 2, 1, Y\rangle$
$= \langle X, F^1, Y\rangle$

b. $\langle F^1, Y\rangle = \langle 10, 9, 6, 7, 8| \ 11, 12, 13, 14, 15, 5, 4, 3, 2, 1, Y\rangle$
$\to \langle 8, 7, 6| \ 9, 10, 11, 12, 13, 14, 15, 5, 4, 3, 2, 1, Y\rangle$
$\to \langle 6, 7, 8, 9, 10, 11, 12, 13, 14, 15| \ 5, 4, 3, 2, 1, Y\rangle$
$\to \langle 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, Y\rangle$
$= \langle {}^\star\mathcal{I}^1_{15}, Y\rangle$

c. $\langle F^2, Y\rangle = \langle 3, 7, 8, 11, 10, 9, 6, 12, 13| \ 4, 5, 15, 14| \ 2, 1, Y\rangle$
$S_5 \swarrow \quad \searrow S_6$
$S_5 = \langle 13, 12, 6, 9, 10, 11, 8, 7, 3, 4, 5, 15, 14, 2, 1, Y\rangle$
$\to \bot$
$S_6 = \langle 14, 15, 5, 4| \ 13, 12, 6, 9, 10, 11, 8, 7, 3, 2, 1, Y\rangle$
$\to \langle 4, 5, 15, 14, 13, 12, 6, 9, 10, 11, 8, 7| \ 3, 2, 1, Y\rangle$
$\to \langle 7, 8, 11, 10, 9| \ 6, 12, 13, 14, 15, 5, 4, 3, 2, 1, Y\rangle$
$\to \langle 9, 10, 11| \ 8, 7, 6, 12, 13, 14, 15, 5, 4, 3, 2, 1, Y\rangle$
$\to \langle 11, 10, 9, 8, 7, 6| \ 12, 13, 14, 15, 5, 4, 3, 2, 1, Y\rangle$
$\to \langle 6, 7, 8, 9, 10, 11, 12, 13, 14, 15| \ 5, 4, 3, 2, 1, Y\rangle$
$\to \langle 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, Y\rangle$
$= \langle {}^\star\mathcal{I}^1_{15}, Y\rangle$

□

2.3 High-level Gadgets

In this section, we define new gadgets based on the four level-1 gadgets. From now on, each property proof uses exclusively properties from smaller gadgets. In order to help the reader follow the ever-present references, we use the following notations. Bold font is used to emphasize the "active" parts of the gadget currently having an element at the head of the sequence. For each relation $S \Longrightarrow T$, we give the relevant reference below (e.g. $S \underset{2.4.}{\Longrightarrow} T$ if it is obtained from Property 2.4). Finally, a summary of all gadget properties (either level-1 or -2) is given in Figure 2.3.

2.3.1 Literals

The following gadget is used only once in the reduction. It contains the locks corresponding to all literals of the formula.

Definition 2.5 (Literals Gadget)**.** *Let p and m be two integers,* $\mathsf{Literals}(p, m)$ *is defined by*

$$\begin{array}{rrcl} & \mathsf{Literals}(p,m) & = & (key_1, \ldots, key_m, test_1, \ldots, test_m, \Lambda) \\ \textit{where} \quad \forall i \in [\![1\,;\,m]\!]\,, & (key_i, test_i, L_i) & = & \mathsf{Lock}(p + 12(i-1)) \\ & \Lambda & = & \langle L_1, L_2, \ldots, L_m\rangle \end{array}$$

Dock gadget

$$\langle {}^{\star}\boldsymbol{\mathcal{I}}_{\boldsymbol{q}}^{\boldsymbol{p+1}},\, X,\, \boldsymbol{D},\, Y\rangle \underset{2.4.}{\Longrightarrow} \langle X,\, \mathcal{I}_{q+2}^{p-1},\, Y\rangle$$

Lock gadget

$$\langle \boldsymbol{key},\, X,\, \boldsymbol{L},\, Y\rangle \underset{2.5.\mathrm{a}}{\Longrightarrow} \langle X,\, L^{o},\, Y\rangle$$
$$\langle \boldsymbol{test},\, X,\, \boldsymbol{L^{o}},\, Y\rangle \underset{2.5.\mathrm{b}}{\Longrightarrow} \langle X,\, \mathcal{I}_{p+12}^{p+1},\, Y\rangle$$
$$\langle \boldsymbol{test},\, X,\, \boldsymbol{L},\, Y\rangle \underset{2.5.\mathrm{c}}{\rightarrow} \bot$$

Hook gadget

$$\langle \boldsymbol{take},\, X,\, \boldsymbol{G},\, Y,\, \boldsymbol{H},\, Z\rangle \underset{2.6.\mathrm{a}}{\Longrightarrow} \langle Y,\, G',\, {}^{\star}X,\, H',\, Z\rangle$$
$$\langle \boldsymbol{put},\, X,\, \boldsymbol{G'},\, {}^{\star}Y,\, \boldsymbol{H'},\, Z\rangle \underset{2.6.\mathrm{b}}{\Longrightarrow} \langle Y,\, G'',\, X,\, H'',\, Z\rangle$$
$$\langle \boldsymbol{G''},\, X,\, \boldsymbol{H''},\, Y\rangle \underset{2.6.\mathrm{c}}{\Longrightarrow} \langle X,\, {}^{\star}\mathcal{I}_{p+12}^{p+1},\, Y\rangle$$

Fork gadget

$$\langle \boldsymbol{E},\, X,\, \boldsymbol{F},\, Y\rangle \underset{2.7.\mathrm{a}}{\Longrightarrow} \left\{ \begin{matrix} \langle X,\, F^{1},\, Y\rangle \\ \langle {}^{\star}X,\, F^{2},\, Y\rangle \end{matrix} \right\}$$
$$\langle \boldsymbol{F^{1}},\, Y\rangle \underset{2.7.\mathrm{b}}{\Longrightarrow} \langle {}^{\star}\mathcal{I}_{p+15}^{p+1},\, Y\rangle$$
$$\langle \boldsymbol{F^{2}},\, Y\rangle \underset{2.7.\mathrm{c}}{\Longrightarrow} \langle {}^{\star}\mathcal{I}_{p+15}^{p+1},\, Y\rangle$$

Literals gadget

$$\forall i \notin O \cup I,\ \langle \boldsymbol{key_{i}},\, X,\, \boldsymbol{\Lambda_{I}^{O}}\rangle \underset{2.8.\mathrm{a}}{\Longrightarrow} \langle X,\, \Lambda_{I}^{O\cup\{i\}}\rangle$$
$$\forall i \in O,\ \langle \boldsymbol{test_{i}},\, X,\, \boldsymbol{\Lambda_{I}^{O}}\rangle \underset{2.8.\mathrm{b}}{\Longrightarrow} \langle X,\, \Lambda_{I\cup\{i\}}^{O-\{i\}}\rangle$$
$$\forall i \notin O,\ \langle \boldsymbol{test_{i}},\, X,\, \boldsymbol{\Lambda_{I}^{O}}\rangle \underset{2.8.\mathrm{c}}{\rightarrow} \bot$$

Variable gadget

$$\langle \boldsymbol{\nu},\, X,\, \boldsymbol{V},\, Y,\, \boldsymbol{\Lambda_{I}^{O}}\rangle \underset{2.9.\mathrm{a}}{\Longrightarrow} \left\{ \begin{matrix} \langle X,\, V^{1},\, Y,\, \Lambda_{I}^{O\cup P}\rangle \\ \langle X,\, V^{2},\, Y,\, \Lambda_{I}^{O\cup N}\rangle \end{matrix} \right\}$$
$$\langle \boldsymbol{V^{1}},\, X,\, \boldsymbol{D},\, Y,\, \boldsymbol{\Lambda_{I}^{O}}\rangle \underset{2.9.\mathrm{b}}{\Longrightarrow} \langle X,\, \mathcal{I}_{p+31}^{p+1},\, Y,\, \Lambda_{I}^{O\cup N}\rangle$$
$$\langle \boldsymbol{V^{2}},\, X,\, \boldsymbol{D},\, Y,\, \boldsymbol{\Lambda_{I}^{O}}\rangle \underset{2.9.\mathrm{c}}{\Longrightarrow} \langle X,\, \mathcal{I}_{p+31}^{p+1},\, Y,\, \Lambda_{I}^{O\cup P}\rangle$$

Clause gadget

$$\langle \boldsymbol{\gamma},\, X,\, \boldsymbol{\Gamma},\, Y,\, \boldsymbol{\Lambda_{I}^{O}}\rangle \underset{2.10.}{\Longrightarrow} \left\{ \begin{matrix} \langle X,\, \Gamma^{1},\, Y,\, \Lambda_{I\cup\{a\}}^{O-\{a\}}\rangle \text{ iff } a \in O \\ \langle X,\, \Gamma^{2},\, Y,\, \Lambda_{I\cup\{b\}}^{O-\{b\}}\rangle \text{ iff } b \in O \\ \langle X,\, \Gamma^{3},\, Y,\, \Lambda_{I\cup\{c\}}^{O-\{c\}}\rangle \text{ iff } c \in O \end{matrix} \right\}$$
$$\langle \boldsymbol{\Gamma^{1}},\, Y,\, \Delta,\, Z,\, \boldsymbol{\Lambda_{I}^{O}}\rangle \underset{2.11.\mathrm{a}}{\Longrightarrow} \langle Y,\, \mathcal{I}_{p+62}^{p+1},\, Z,\, \Lambda_{I\cup\{b,c\}}^{O-\{b,c\}}\rangle$$
$$\langle \boldsymbol{\Gamma^{2}},\, Y,\, \Delta,\, Z,\, \boldsymbol{\Lambda_{I}^{O}}\rangle \underset{2.11.\mathrm{b}}{\Longrightarrow} \langle Y,\, \mathcal{I}_{p+62}^{p+1},\, Z,\, \Lambda_{I\cup\{a,c\}}^{O-\{a,c\}}\rangle$$
$$\langle \boldsymbol{\Gamma^{3}},\, Y,\, \Delta,\, Z,\, \boldsymbol{\Lambda_{I}^{O}}\rangle \underset{2.11.\mathrm{c}}{\Longrightarrow} \langle Y,\, \mathcal{I}_{p+62}^{p+1},\, Z,\, \Lambda_{I\cup\{a,b\}}^{O-\{a,b\}}\rangle$$

Figure 2.3: Compilation of all gadget properties. As a general rule, X, Y, Z can be any sequences, O and I any disjoint subsets of $[\![1\,;\,m]\!]$. See respective definitions and properties for specific constraints and notations.

Let O and I be two disjoint subsets of $[\![1\,;\,m]\!]$. We use Λ_I^O for the sequence obtained from Λ by replacing L_i by L_i^o for all $i \in O$ and by $\mathcal{I}_{p+12i}^{p+12i-11}$ for all $i \in I$.

Elements of O correspond to open locks in Λ_I^O, while elements of I correspond to open locks which have moreover been tested. Note that $\Lambda_\emptyset^\emptyset = \Lambda$, and that $\Lambda_{[\![1\,;\,m]\!]}^\emptyset = \mathcal{I}_{p+12m}^{p+1}$.

Property 2.8. *Let p and m be two integers, $(key_1, \ldots, key_m, test_1, \ldots, test_m, \Lambda) = \mathsf{Literals}(p, m)$, O and I be two disjoint subsets of $[\![1\,;\,m]\!]$, and X be any sequence. We have*

a. $\forall i \in [\![1\,;\,m]\!] - O - I, \quad \langle key_i,\, X,\, \Lambda_I^O \rangle \Longrightarrow \langle X,\, \Lambda_I^{O\cup\{i\}} \rangle$

b. $\forall i \in O, \quad \langle test_i,\, X,\, \Lambda_I^O \rangle \Longrightarrow \langle X,\, \Lambda_{I\cup\{i\}}^{O-\{i\}} \rangle$

c. $\forall i \in [\![1\,;\,m]\!] - O, \quad \langle test_i,\, X,\, \Lambda_I^O \rangle \to \bot$

Proof. The proof follows from Property 2.5.

a. Let $i \in [\![1\,;\,m]\!] - O - I$. Then Λ_I^O can be written $\Lambda_I^O = \langle A,\, L_i,\, B \rangle$. Hence

$$\begin{aligned} \langle key_i,\, X,\, \Lambda_I^O \rangle &= \langle \boldsymbol{key_i},\, X,\, A,\, \boldsymbol{L_i},\, B \rangle \\ &\underset{2.5.\mathrm{a}}{\Longrightarrow} \langle X,\, A,\, L_i^o,\, B \rangle \\ &= \langle X,\, \Lambda_I^{O\cup\{i\}} \rangle \end{aligned}$$

b. Let $i \in O$. Then Λ_I^O can be written $\Lambda_I^O = \langle A,\, L_i^o,\, B \rangle$. Hence

$$\begin{aligned} \langle test_i,\, X,\, \Lambda_I^O \rangle &= \langle \boldsymbol{test_i},\, X,\, A,\, \boldsymbol{L_i^o},\, B \rangle \\ &\underset{2.5.\mathrm{b}}{\Longrightarrow} \langle X,\, A,\, \mathcal{I}_{p+12i}^{p+12i-11},\, B \rangle \\ &= \langle X,\, \Lambda_{I\cup\{i\}}^{O-\{i\}} \rangle \end{aligned}$$

c. Let $i \in [\![1\,;\,m]\!] - O$. If $i \in I$, then $test_i \in \mathcal{I}_{p+12i}^{p+12i-11} \subset \Lambda_I^O$, and $\langle test_i,\, X,\, \Lambda_I^O \rangle$ is not a valid sequence (it contains a duplicate). Otherwise, $i \in [\![1\,;\,m]\!] - O - I$, and Λ_I^O can be written $\Lambda_I^O = \langle A,\, L_i,\, B \rangle$. Hence

$$\langle test_i,\, X,\, \Lambda_I^O \rangle = \langle \boldsymbol{test_i},\, X,\, A,\, \boldsymbol{L_i},\, B \rangle \underset{2.5.\mathrm{c}}{\to} \bot$$

□

2.3.2 Variable

In the following two sections, we assume that p_Λ and m are two fixed integers, and we define the gadget $(key_1, \ldots, key_m, test_1, \ldots, test_m, \Lambda) = \mathsf{Literals}(p_\Lambda, m)$. Thus, we can use elements key_i and $test_i$ for $i \in [\![1\,;\,m]\!]$, and sequences Λ_I^O for any disjoint subsets O and I of $[\![1\,;\,m]\!]$.

We now define a gadget simulating a boolean variable x_i. It holds two series of *key* elements: the ones with indices in P (resp. N) open the locks corresponding to literals of the form x_i (resp. $\neg x_i$). When the triggering element, ν, is brought to the head, a choice has to be made between P and N, and the locks associated with the chosen set (and only these) are opened.

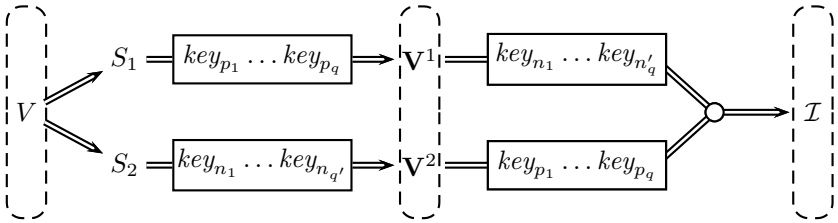


Figure 2.4: Initially, a variable gadget contains mainly the sequence V. Property 2.9a proves that two paths are possible, leading to sequences containing either V^1 or V^2. Along the first (resp. second) path, the locks with indices in P (resp. N) are opened. By Property 2.9b (resp. c), there exists a path transforming V^1 (resp. V^2) into the identity over $[\![p+1\,;\,p+31]\!]$, which opens the remaining locks.

Definition 2.6 (Variable Gadget)**.** *Let P, N be two disjoint subsets of $[\![1\,;\,m]\!]$ ($P = \{p_1, p_2, \ldots, p_q\}$, $N = \{n_1, n_2, \ldots, n_{q'}\}$) and p be an integer,* $\mathsf{Variable}(P, N, p)$ *is defined by*

$$
\begin{array}{l}
\mathsf{Variable}(P, N, p) = (\nu, V, D) \\
\textit{where} \quad (\mathit{take}, \mathit{put}, G, H) = \mathsf{Hook}(p+2), \quad (E, F) = \mathsf{Fork}(p+14), \\
\textit{in} \quad \begin{array}{rl}
\nu & = \mathit{take} \\
V & = \langle G,\ E,\ \mathit{key}_{p_1},\ \ldots,\ \mathit{key}_{p_q},\ \mathit{put},\ \mathit{key}_{n_1},\ \ldots,\ \mathit{key}_{n_{q'}},\ F,\ H \rangle \\
D & = \mathsf{Dock}(p+2, p+29)
\end{array}
\end{array}
$$

Given a variable gadget $(\nu, V, D) = \mathsf{Variable}(P, N, p)$, we write

$$
\begin{array}{rcl}
V^1 & = & \langle G'',\ \mathit{key}_{n_1},\ \ldots,\ \mathit{key}_{n_{q'}},\ F^1,\ H'' \rangle \\
V^2 & = & \langle G'',\ \mathit{key}_{p_q},\ \ldots,\ \mathit{key}_{p_1},\ F^2,\ H'' \rangle
\end{array}
$$

where G'', H'', F^1, F^2, come from the definitions of Hook (Definition 2.3) and Fork (Definition 2.4).

The following property determines the possible behavior of a variable gadget. It is illustrated in Figure 2.4.

Property 2.9. *Let P, N be two disjoint subsets of $[\![1\,;\,m]\!]$, p be an integer, X and Y be two sequences, O, I be two disjoint subsets of $[\![1\,;\,m]\!]$, and $(\nu, V, D) =$ $\mathsf{Variable}(P, N, p)$. For sub-property (a.) we require that $(P \cup N) \cap (O \cup I) = \emptyset$, for (b.) that $N \cap (O \cup I) = \emptyset$, and for (c.) that $P \cap (O \cup I) = \emptyset$ (these conditions are in fact necessarily satisfied by construction since all sequences considered are permutations). We have*

$$
\begin{array}{ll}
\textbf{a.} & \langle \nu,\ X,\ V,\ Y,\ \Lambda_I^O \rangle \Longrightarrow \left\{ \begin{array}{l} \langle X,\ V^1,\ Y,\ \Lambda_I^{O \cup P} \rangle, \\ \langle X,\ V^2,\ Y,\ \Lambda_I^{O \cup N} \rangle \end{array} \right\} \\
\textbf{b.} & \langle V^1,\ X,\ D,\ Y,\ \Lambda_I^O \rangle \Longrightarrow \langle X,\ \mathcal{I}_{p+31}^{p+1},\ Y,\ \Lambda_I^{O \cup N} \rangle \\
\textbf{c.} & \langle V^2,\ X,\ D,\ Y,\ \Lambda_I^O \rangle \Longrightarrow \langle X,\ \mathcal{I}_{p+31}^{p+1},\ Y,\ \Lambda_I^{O \cup P} \rangle
\end{array}
$$

Proof.

a. $\langle \nu, X, V, Y, \Lambda_I^O \rangle$
$= \langle \boldsymbol{take}, X, \boldsymbol{G}, E, key_{p_1}, \ldots, key_{p_q}, put, key_{n_1}, \ldots, key_{n_{q'}}, F, \boldsymbol{H}, Y, \Lambda_I^O \rangle$
$\underset{2.6.\text{a}}{\Longrightarrow} \langle \boldsymbol{E}, key_{p_1}, \ldots, key_{p_q}, put, key_{n_1}, \ldots, key_{n_{q'}}, \boldsymbol{F}, G', {}^\star X, H', Y, \Lambda_I^O \rangle$
$\underset{2.7.\text{a}}{\Longrightarrow} \{S_1, S_2\}$ (see below.)

$S_1 = \langle \boldsymbol{key_{p_1}}, key_{p_2}, \ldots, key_{p_q}, put, key_{n_1}, \ldots, key_{n_{q'}}, F^1, G', {}^\star X, H', Y, \boldsymbol{\Lambda_I^O} \rangle$
$\underset{2.8.\text{a}}{\Longrightarrow} \langle \boldsymbol{key_{p_2}}, \ldots, key_{p_q}, put, key_{n_1}, \ldots, key_{n_{q'}}, F^1, G', {}^\star X, H', Y, \boldsymbol{\Lambda_I^{O \cup \{p_1\}}} \rangle$
$\vdots$
$\underset{2.8.\text{a}}{\Longrightarrow} \langle \boldsymbol{put}, key_{n_1}, \ldots, key_{n_{q'}}, F^1, \boldsymbol{G'}, {}^\star X, \boldsymbol{H'}, Y, \Lambda_I^{O \cup P} \rangle$
$\underset{2.6.\text{b}}{\Longrightarrow} \langle X, G'', key_{n_1}, \ldots, key_{n_{q'}}, F^1, H'', Y, \Lambda_I^{O \cup P} \rangle$
$= \langle X, V^1, Y, \Lambda_I^{O \cup P} \rangle$

$S_2 = \langle \boldsymbol{key_{n_{q'}}}, key_{n_{q'-1}}, \ldots, key_{n_1}, put, key_{p_q}, \ldots, key_{p_1}, F^2, G', {}^\star X, H', Y, \boldsymbol{\Lambda_I^O} \rangle$
$\underset{2.8.\text{a}}{\Longrightarrow} \langle \boldsymbol{key_{n_{q'-1}}}, \ldots, key_{n_1}, put, key_{p_q}, \ldots, key_{p_1}, F^2, G', {}^\star X, H', Y, \boldsymbol{\Lambda_I^{O \cup \{n_{q'}\}}} \rangle$
$\vdots$
$\underset{2.8.\text{a}}{\Longrightarrow} \langle \boldsymbol{put}, key_{p_q}, \ldots, key_{p_1}, F^2, \boldsymbol{G'}, {}^\star X, \boldsymbol{H'}, Y, \Lambda_I^{O \cup N} \rangle$
$\underset{2.6.\text{b}}{\Longrightarrow} \langle X, G'', key_{p_q}, \ldots, key_{p_1}, F^2, H'', Y, \Lambda_I^{O \cup N} \rangle$
$= \langle X, V^2, Y, \Lambda_I^{O \cup N} \rangle$

b. $\langle V^1, X, D, Y, \Lambda_I^O \rangle = \langle \boldsymbol{G''}, key_{n_1}, \ldots, key_{n_{q'}}, F^1, \boldsymbol{H''}, X, D, Y, \Lambda_I^O \rangle$
$\underset{2.6.\text{c}}{\Longrightarrow} \langle \boldsymbol{key_{n_1}}, key_{n_2}, \ldots, key_{n_{q'}}, F^1, {}^\star \mathcal{I}_{p+14}^{p+3}, X, D, Y, \boldsymbol{\Lambda_I^O} \rangle$
$\underset{2.8.\text{a}}{\Longrightarrow} \langle \boldsymbol{key_{n_2}}, \ldots, key_{n_{q'}}, F^1, {}^\star \mathcal{I}_{p+14}^{p+3}, X, D, Y, \boldsymbol{\Lambda_I^{O \cup \{n_1\}}} \rangle$
$\vdots$
$\underset{2.8.\text{a}}{\Longrightarrow} \langle \boldsymbol{F^1}, {}^\star \mathcal{I}_{p+14}^{p+3}, X, D, Y, \Lambda_I^{O \cup N} \rangle$
$\underset{2.7.\text{b}}{\Longrightarrow} \langle {}^\star \boldsymbol{\mathcal{I}_{p+29}^{p+15}}, {}^\star \boldsymbol{\mathcal{I}_{p+14}^{p+3}}, X, \boldsymbol{D}, Y, \Lambda_I^{O \cup N} \rangle$
$\underset{2.4.}{\Longrightarrow} \langle X, \mathcal{I}_{p+31}^{p+1}, Y, \Lambda_I^{O \cup N} \rangle$

c. $\langle V^2, X, D, Y, \Lambda_I^O \rangle = \langle \boldsymbol{G''}, key_{p_q}, \ldots, key_{p_1}, F^2, \boldsymbol{H''}, X, D, Y, \Lambda_I^O \rangle$
$\underset{2.6.\text{c}}{\Longrightarrow} \langle \boldsymbol{key_{p_q}}, key_{p_{q-1}}, \ldots, key_{p_1}, F^2, {}^\star \mathcal{I}_{p+14}^{p+3}, X, D, Y, \boldsymbol{\Lambda_I^O} \rangle$
$\underset{2.8.\text{a}}{\Longrightarrow} \langle \boldsymbol{key_{p_{q-1}}}, \ldots, key_{p_1}, F^2, {}^\star \mathcal{I}_{p+14}^{p+3}, X, D, Y, \boldsymbol{\Lambda_I^{O \cup \{p_q\}}} \rangle$
$\vdots$
$\underset{2.8.\text{a}}{\Longrightarrow} \langle \boldsymbol{F^2}, {}^\star \mathcal{I}_{p+14}^{p+3}, X, D, Y, \Lambda_I^{O \cup P} \rangle$
$\underset{2.7.\text{c}}{\Longrightarrow} \langle {}^\star \boldsymbol{\mathcal{I}_{p+29}^{p+15}}, {}^\star \boldsymbol{\mathcal{I}_{p+14}^{p+3}}, X, \boldsymbol{D}, Y, \Lambda_I^{O \cup P} \rangle$
$\underset{2.4.}{\Longrightarrow} \langle X, \mathcal{I}_{p+31}^{p+1}, Y, \Lambda_I^{O \cup P} \rangle$

□

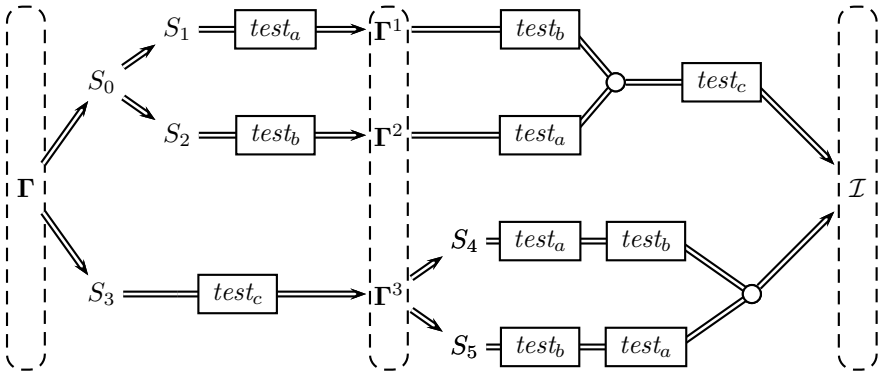


Figure 2.5: Initially, a clause gadget contains mainly the sequence Γ. Property 2.10 proves that three paths may be possible, leading to sequences containing either Γ^1, Γ^2 or Γ^3. Because of the *test* elements, each path requires one lock to be open (either a, b or c). By Property 2.11a (resp. b, c), there exists a path transforming Γ^1 (resp. Γ^2, Γ^3) into the identity over $[\![p+1\,;\,p+62]\!]$, provided the remaining locks are open.

2.3.3 Clause

The following gadget simulates a 3-clause in a boolean formula. It holds the *test* elements for three locks, corresponding to three literals. When the triggering element, γ, is at the head of a sequence, three distinct efficient paths may be followed. In each such path, one of the three locks is tested: in other words, any efficient path leading to the identity requires one of the locks to be open.

Definition 2.7 (Clause Gadget)**.** *Let $a, b, c \in [\![1\,;\,m]\!]$ be pairwise distinct integers and p be an integer,* $\mathsf{Clause}(a,b,c,p)$ *is defined by*

$$\begin{array}{lrlrl}
\mathsf{Clause}(a,b,c,p) = (\gamma, \Gamma, \Delta) & & & & \\
\quad \textit{where} & (E_1, F_1) & = \mathsf{Fork}(p+2), & (take_1, put_1, G_1, H_1) & = \mathsf{Hook}(p+21), \\
& (E_2, F_2) & = \mathsf{Fork}(p+45), & (take_2, put_2, G_2, H_2) & = \mathsf{Hook}(p+33), \\
\quad \textit{in} \quad \gamma & = take_1 & & & \\
\Gamma & = \langle G_1,\ E_1,\ take_2,\ put_1,\ test_c,\ F_1,\ G_2,\ E_2,\ test_a,\ put_2,\ test_b,\ F_2,\ H_2,\ H_1 \rangle & & & \\
\Delta & = \langle \mathsf{Dock}(p+2, p+17),\ \mathsf{Dock}(p+21, p+60) \rangle & & &
\end{array}$$

Given a clause gadget $(\gamma, \Gamma, \Delta) = \mathsf{Clause}(a,b,c,p)$, we write

$$\begin{array}{rcl}
\Gamma^1 & = & \langle G''_1,\ test_c,\ F^1_1,\ G''_2,\ test_b,\ F^1_2,\ H''_2,\ H''_1 \rangle \\
\Gamma^2 & = & \langle G''_1,\ test_c,\ F^1_1,\ G''_2,\ test_a,\ F^2_2,\ H''_2,\ H''_1 \rangle \\
\Gamma^3 & = & \langle G''_1,\ take_2,\ F^2_1,\ G_2,\ E_2,\ test_a,\ put_2,\ test_b,\ F_2,\ H_2,\ H''_1 \rangle
\end{array}$$

The following two properties determine the possible behavior of a clause gadget. They are illustrated in Figure 2.5. The main point is that, starting from a sequence $\langle \gamma,\ X,\ \Gamma,\ Y,\ \Lambda^O_I \rangle$, there is one efficient path for each true literal in the clause (ie. each literal with index in O).

Property 2.10. *Let $a, b, c \in [\![1\,;\,m]\!]$ be pairwise distinct integers, p be an integer, $(\gamma, \Gamma, \Delta) = \mathsf{Clause}(a, b, c, p)$, X and Y be any sequences, and O, I be two disjoint subsets of $[\![1\,;\,m]\!]$. We have*

$$\langle \gamma,\, X,\, \Gamma,\, Y,\, \Lambda_I^O \rangle \Longrightarrow \mathbb{T},$$

where $\mathbb{T}$ is the set containing from 0 to 3 sequences, defined by:

$$\begin{aligned}
\langle X,\, \Gamma^1,\, Y,\, \Lambda_{I\cup\{a\}}^{O-\{a\}} \rangle \in \mathbb{T} \quad &\textit{iff} \quad a \in O \\
\langle X,\, \Gamma^2,\, Y,\, \Lambda_{I\cup\{b\}}^{O-\{b\}} \rangle \in \mathbb{T} \quad &\textit{iff} \quad b \in O \\
\langle X,\, \Gamma^3,\, Y,\, \Lambda_{I\cup\{c\}}^{O-\{c\}} \rangle \in \mathbb{T} \quad &\textit{iff} \quad c \in O
\end{aligned}$$

Proof.

$$\begin{aligned}
&\langle \gamma,\, X,\, \Gamma,\, Y,\, \Lambda_I^O \rangle \\
&= \langle \boldsymbol{take_1},\, X,\, \boldsymbol{G_1},\, E_1,\, take_2,\, put_1,\, test_c,\, F_1,\, G_2,\, E_2,\, test_a,\, put_2,\, test_b,\, F_2,\, H_2,\, \boldsymbol{H_1},\, Y,\, \Lambda_I^O \rangle \\
&\underset{2.6.\mathrm{a}}{\Longrightarrow} \langle \boldsymbol{E_1},\, take_2,\, put_1,\, test_c,\, \boldsymbol{F_1},\, G_2,\, E_2,\, test_a,\, put_2,\, test_b,\, F_2,\, H_2,\, G_1',\, {}^\star X,\, H_1',\, Y,\, \Lambda_I^O \rangle \\
&\underset{2.7.\mathrm{a}}{\Longrightarrow} \{S_0, S_3\}
\end{aligned}$$

$$\begin{aligned}
S_0 &= \langle \boldsymbol{take_2},\, put_1,\, test_c,\, F_1^1,\, \boldsymbol{G_2},\, E_2,\, test_a,\, put_2,\, test_b,\, F_2,\, \boldsymbol{H_2},\, G_1',\, {}^\star X,\, H_1',\, Y,\, \Lambda_I^O \rangle \\
&\underset{2.6.\mathrm{a}}{\Longrightarrow} \langle \boldsymbol{E_2},\, test_a,\, put_2,\, test_b,\, \boldsymbol{F_2},\, G_2',\, {}^\star F_1^1,\, test_c,\, put_1,\, H_2',\, G_1',\, {}^\star X,\, H_1',\, Y,\, \Lambda_I^O \rangle \\
&\underset{2.7.\mathrm{a}}{\Longrightarrow} \{S_1, S_2\}
\end{aligned}$$

$$\begin{aligned}
S_1 &= \langle \boldsymbol{test_a},\, put_2,\, test_b,\, F_2^1,\, G_2',\, {}^\star F_1^1,\, test_c,\, put_1,\, H_2',\, G_1',\, {}^\star X,\, H_1',\, Y,\, \boldsymbol{\Lambda_I^O} \rangle \\
&\text{if } a \notin O \text{ then } S_1 \underset{2.8.\mathrm{c}}{\to} \bot \\
&\text{if } a \in O \text{ then} \\
S_1 &\underset{2.8.\mathrm{b}}{\Longrightarrow} \langle \boldsymbol{put_2},\, test_b,\, F_2^1,\, \boldsymbol{G_2'},\, {}^\star F_1^1,\, test_c,\, put_1,\, \boldsymbol{H_2'},\, G_1',\, {}^\star X,\, H_1',\, Y,\, \Lambda_{I\cup\{a\}}^{O-\{a\}} \rangle \\
&\underset{2.6.\mathrm{b}}{\Longrightarrow} \langle \boldsymbol{put_1},\, test_c,\, F_1^1,\, G_2'',\, test_b,\, F_2^1,\, H_2'',\, \boldsymbol{G_1'},\, {}^\star X,\, \boldsymbol{H_1'},\, Y,\, \Lambda_{I\cup\{a\}}^{O-\{a\}} \rangle \\
&\underset{2.6.\mathrm{b}}{\Longrightarrow} \langle X,\, G_1'',\, test_c,\, F_1^1,\, G_2'',\, test_b,\, F_2^1,\, H_2'',\, H_1'',\, Y,\, \Lambda_{I\cup\{a\}}^{O-\{a\}} \rangle \\
&= \langle X,\, \Gamma^1,\, Y,\, \Lambda_{I\cup\{a\}}^{O-\{a\}} \rangle
\end{aligned}$$

$$\begin{aligned}
S_2 &= \langle \boldsymbol{test_b},\, put_2,\, test_a,\, F_2^2,\, G_2',\, {}^\star F_1^1,\, test_c,\, put_1,\, H_2',\, G_1',\, {}^\star X,\, H_1',\, Y,\, \boldsymbol{\Lambda_I^O} \rangle \\
&\text{if } b \notin O \text{ then } S_2 \underset{2.8.\mathrm{c}}{\to} \bot \\
&\text{if } b \in O \text{ then} \\
S_2 &\underset{2.8.\mathrm{b}}{\Longrightarrow} \langle \boldsymbol{put_2},\, test_a,\, F_2^2,\, \boldsymbol{G_2'},\, {}^\star F_1^1,\, test_c,\, put_1,\, \boldsymbol{H_2'},\, G_1',\, {}^\star X,\, H_1',\, Y,\, \Lambda_{I\cup\{b\}}^{O-\{b\}} \rangle \\
&\underset{2.6.\mathrm{b}}{\Longrightarrow} \langle \boldsymbol{put_1},\, test_c,\, F_1^1,\, G_2'',\, test_a,\, F_2^2,\, H_2'',\, \boldsymbol{G_1'},\, {}^\star X,\, \boldsymbol{H_1'},\, Y,\, \Lambda_{I\cup\{b\}}^{O-\{b\}} \rangle \\
&\underset{2.6.\mathrm{b}}{\Longrightarrow} \langle X,\, G_1'',\, test_c,\, F_1^1,\, G_2'',\, test_a,\, F_2^2,\, H_2'',\, H_1'',\, Y,\, \Lambda_{I\cup\{b\}}^{O-\{b\}} \rangle \\
&= \langle X,\, \Gamma^2,\, Y,\, \Lambda_{I\cup\{b\}}^{O-\{b\}} \rangle
\end{aligned}$$

$S_3 = \langle \boldsymbol{test_c}, put_1, take_2, F_1^2, G_2, E_2, test_a, put_2, test_b, F_2, H_2, G_1', {}^\star X, H_1', Y, \boldsymbol{\Lambda_I^O} \rangle$
if $c \notin O$ then $S_3 \underset{2.8.\text{c}}{\to} \bot$
if $c \in O$ then
$S_3 \underset{2.8.\text{b}}{\Longrightarrow} \langle \boldsymbol{put_1}, take_2, F_1^2, G_2, E_2, test_a, put_2, test_b, F_2, H_2, \boldsymbol{G_1'}, {}^\star X, \boldsymbol{H_1'}, Y, \Lambda_{I\cup\{c\}}^{O-\{c\}} \rangle$
$\underset{2.6.\text{b}}{\Longrightarrow} \langle X, G_1'', take_2, F_1^2, G_2, E_2, test_a, put_2, test_b, F_2, H_2, H_1'', Y, \Lambda_{I\cup\{c\}}^{O-\{c\}} \rangle$
$= \langle X, \Gamma^3, Y, \Lambda_{I\cup\{c\}}^{O-\{c\}} \rangle$

□

With the previous property we have shown that with γ as the head element, any of three efficient paths can be followed, leading to Γ being transformed into respectively Γ_1, Γ_2 and Γ_3. In each path, exactly one lock is tested, hence it needs to have been opened beforehand.

We now prove that, in a second time, efficient paths exist to sort Γ_1, Γ_2 and Γ_3 into the same identity sequence. Moreover, remaining locks are tested.

Property 2.11. *Let $a, b, c \in [\![1\,;\,m]\!]$ be pairwise distinct integers, p be an integer, $(\gamma, \Gamma, \Delta) = \mathsf{Clause}(a, b, c, p)$, Y and Z be any sequences, and O, I be two disjoint subsets of $[\![1\,;\,m]\!]$. We have*

a. *If $b, c \in O$, then* $\langle \Gamma^1, Y, \Delta, Z, \Lambda_I^O \rangle \implies \langle Y, \mathcal{I}_{p+62}^{p+1}, Z, \Lambda_{I\cup\{b,c\}}^{O-\{b,c\}} \rangle$

b. *If $a, c \in O$, then* $\langle \Gamma^2, Y, \Delta, Z, \Lambda_I^O \rangle \implies \langle Y, \mathcal{I}_{p+62}^{p+1}, Z, \Lambda_{I\cup\{a,c\}}^{O-\{a,c\}} \rangle$

c. *If $a, b \in O$, then* $\langle \Gamma^3, Y, \Delta, Z, \Lambda_I^O \rangle \implies \langle Y, \mathcal{I}_{p+62}^{p+1}, Z, \Lambda_{I\cup\{a,b\}}^{O-\{a,b\}} \rangle$

Proof.

a.
$\langle \Gamma^1, Y, \Delta, Z, \Lambda_I^O \rangle$
$= \langle \boldsymbol{G_1''}, test_c, F_1^1, G_2'', test_b, F_2^1, H_2'', \boldsymbol{H_1''}, Y, D_1, D_2, Z, \Lambda_I^O \rangle$
$\underset{2.6.\text{c}}{\Longrightarrow} \langle \boldsymbol{test_c}, F_1^1, G_2'', test_b, F_2^1, H_2'', {}^\star\mathcal{I}_{p+33}^{p+22}, Y, D_1, D_2, Z, \boldsymbol{\Lambda_I^O} \rangle$
$\underset{2.8.\text{b}}{\Longrightarrow} \langle \boldsymbol{F_1^1}, G_2'', test_b, F_2^1, H_2'', {}^\star\mathcal{I}_{p+33}^{p+22}, Y, D_1, D_2, Z, \Lambda_{I\cup\{c\}}^{O-\{c\}} \rangle$
$\underset{2.7.\text{b}}{\Longrightarrow} \langle {}^\star\boldsymbol{\mathcal{I}_{p+17}^{p+3}}, G_2'', test_b, F_2^1, H_2'', {}^\star\mathcal{I}_{p+33}^{p+22}, Y, \boldsymbol{D_1}, D_2, Z, \Lambda_{I\cup\{c\}}^{O-\{c\}} \rangle$
$\underset{2.4.}{\Longrightarrow} \langle \boldsymbol{G_2''}, test_b, F_2^1, \boldsymbol{H_2''}, {}^\star\mathcal{I}_{p+33}^{p+22}, Y, \mathcal{I}_{p+19}^{p+1}, D_2, Z, \Lambda_{I\cup\{c\}}^{O-\{c\}} \rangle$
$\underset{2.6.\text{c}}{\Longrightarrow} \langle \boldsymbol{test_b}, F_2^1, {}^\star\mathcal{I}_{p+45}^{p+34}, {}^\star\mathcal{I}_{p+33}^{p+22}, Y, \mathcal{I}_{p+19}^{p+1}, D_2, Z, \boldsymbol{\Lambda_{I\cup\{c\}}^{O-\{c\}}} \rangle$
$\underset{2.8.\text{b}}{\Longrightarrow} \langle \boldsymbol{F_2^1}, {}^\star\mathcal{I}_{p+45}^{p+34}, {}^\star\mathcal{I}_{p+33}^{p+22}, Y, \mathcal{I}_{p+19}^{p+1}, D_2, Z, \Lambda_{I\cup\{b,c\}}^{O-\{b,c\}} \rangle$
$\underset{2.7.\text{b}}{\Longrightarrow} \langle {}^\star\boldsymbol{\mathcal{I}_{p+60}^{p+46}}, {}^\star\boldsymbol{\mathcal{I}_{p+45}^{p+34}}, {}^\star\boldsymbol{\mathcal{I}_{p+33}^{p+22}}, Y, \mathcal{I}_{p+19}^{p+1}, \boldsymbol{D_2}, Z, \Lambda_{I\cup\{b,c\}}^{O-\{b,c\}} \rangle$
$\underset{2.4.}{\Longrightarrow} \langle Y, \mathcal{I}_{p+19}^{p+1}, \mathcal{I}_{p+62}^{p+20}, Z, \Lambda_{I\cup\{b,c\}}^{O-\{b,c\}} \rangle$
$= \langle Y, \mathcal{I}_{p+62}^{p+1}, Z, \Lambda_{I\cup\{b,c\}}^{O-\{b,c\}} \rangle$

b. $\langle \Gamma^2, Y, \Delta, Z, \Lambda_I^O \rangle$

$= \langle \boldsymbol{G''_1}, test_c, F_1^1, G''_2, test_a, F_2^2, H''_2, \boldsymbol{H''_1}, Y, D_1, D_2, Z, \Lambda_I^O \rangle$

$\underset{\text{2.6.c}}{\Longrightarrow} \langle \boldsymbol{test_c}, F_1^1, G''_2, test_a, F_2^2, H''_2, {}^\star\mathcal{I}_{p+33}^{p+22}, Y, D_1, D_2, Z, \boldsymbol{\Lambda_I^O} \rangle$

$\underset{\text{2.8.b}}{\Longrightarrow} \langle \boldsymbol{F_1^1}, G''_2, test_a, F_2^2, H''_2, {}^\star\mathcal{I}_{p+33}^{p+22}, Y, D_1, D_2, Z, \Lambda_{I\cup\{c\}}^{O-\{c\}} \rangle$

$\underset{\text{2.7.b}}{\Longrightarrow} \langle \boldsymbol{{}^\star\mathcal{I}_{p+17}^{p+3}}, G''_2, test_a, F_2^2, H''_2, {}^\star\mathcal{I}_{p+33}^{p+22}, Y, \boldsymbol{D_1}, D_2, Z, \Lambda_{I\cup\{c\}}^{O-\{c\}} \rangle$

$\underset{\text{2.4.}}{\Longrightarrow} \langle \boldsymbol{G''_2}, test_a, F_2^2, \boldsymbol{H''_2}, {}^\star\mathcal{I}_{p+33}^{p+22}, Y, \mathcal{I}_{p+19}^{p+1}, D_2, Z, \Lambda_{I\cup\{c\}}^{O-\{c\}} \rangle$

$\underset{\text{2.6.c}}{\Longrightarrow} \langle \boldsymbol{test_a}, F_2^2, {}^\star\mathcal{I}_{p+45}^{p+34}, {}^\star\mathcal{I}_{p+33}^{p+22}, Y, \mathcal{I}_{p+19}^{p+1}, D_2, Z, \boldsymbol{\Lambda_{I\cup\{c\}}^{O-\{c\}}} \rangle$

$\underset{\text{2.8.b}}{\Longrightarrow} \langle \boldsymbol{F_2^2}, {}^\star\mathcal{I}_{p+45}^{p+34}, {}^\star\mathcal{I}_{p+33}^{p+22}, Y, \mathcal{I}_{p+19}^{p+1}, D_2, Z, \Lambda_{I\cup\{a,c\}}^{O-\{a,c\}} \rangle$

$\underset{\text{2.7.c}}{\Longrightarrow} \langle \boldsymbol{{}^\star\mathcal{I}_{p+60}^{p+46}}, \boldsymbol{{}^\star\mathcal{I}_{p+45}^{p+34}}, \boldsymbol{{}^\star\mathcal{I}_{p+33}^{p+22}}, Y, \mathcal{I}_{p+19}^{p+1}, \boldsymbol{D_2}, Z, \Lambda_{I\cup\{a,c\}}^{O-\{a,c\}} \rangle$

$\underset{\text{2.4.}}{\Longrightarrow} \langle Y, \mathcal{I}_{p+19}^{p+1}, \mathcal{I}_{p+62}^{p+20}, Z, \Lambda_{I\cup\{a,c\}}^{O-\{a,c\}} \rangle$

$= \langle Y, \mathcal{I}_{p+62}^{p+1}, Z, \Lambda_{I\cup\{a,c\}}^{O-\{a,c\}} \rangle$

c. $\langle \Gamma^3, Y, \Delta, Z, \Lambda_I^O \rangle$

$= \langle \boldsymbol{G''_1}, take_2, F_1^2, G_2, E_2, test_a, put_2, test_b, F_2, H_2, \boldsymbol{H''_1}, Y, D_1, D_2, Z, \Lambda_I^O \rangle$

$\underset{\text{2.6.c}}{\Longrightarrow} \langle \boldsymbol{take_2}, F_1^2, \boldsymbol{G_2}, E_2, test_a, put_2, test_b, F_2, \boldsymbol{H_2}, {}^\star\mathcal{I}_{p+33}^{p+22}, Y, D_1, D_2, Z, \Lambda_I^O \rangle$

$\underset{\text{2.6.a}}{\Longrightarrow} \langle \boldsymbol{E_2}, test_a, put_2, test_b, \boldsymbol{F_2}, G'_2, {}^\star F_1^2, H'_2, {}^\star\mathcal{I}_{p+33}^{p+22}, Y, D_1, D_2, Z, \Lambda_I^O \rangle$

$\underset{\text{2.7.a}}{\Longrightarrow} \{S_4, S_5\}$

$S_4 = \langle \boldsymbol{test_a}, put_2, test_b, F_2^1, G'_2, {}^\star F_1^2, H'_2, {}^\star\mathcal{I}_{p+33}^{p+22}, Y, D_1, D_2, Z, \boldsymbol{\Lambda_I^O} \rangle$

$\underset{\text{2.8.b}}{\Longrightarrow} \langle \boldsymbol{put_2}, test_b, F_2^1, \boldsymbol{G'_2}, {}^\star F_1^2, \boldsymbol{H'_2}, {}^\star\mathcal{I}_{p+33}^{p+22}, Y, D_1, D_2, Z, \Lambda_{I\cup\{a\}}^{O-\{a\}} \rangle$

$\underset{\text{2.6.b}}{\Longrightarrow} \langle \boldsymbol{F_1^2}, G''_2, test_b, F_2^1, H''_2, {}^\star\mathcal{I}_{p+33}^{p+22}, Y, D_1, D_2, Z, \Lambda_{I\cup\{a\}}^{O-\{a\}} \rangle$

$\underset{\text{2.7.c}}{\Longrightarrow} \langle \boldsymbol{{}^\star\mathcal{I}_{p+17}^{p+3}}, G''_2, test_b, F_2^1, H''_2, {}^\star\mathcal{I}_{p+33}^{p+22}, Y, \boldsymbol{D_1}, D_2, Z, \Lambda_{I\cup\{a\}}^{O-\{a\}} \rangle$

$\underset{\text{2.4.}}{\Longrightarrow} \langle \boldsymbol{G''_2}, test_b, F_2^1, \boldsymbol{H''_2}, {}^\star\mathcal{I}_{p+33}^{p+22}, Y, \mathcal{I}_{p+19}^{p+1}, D_2, Z, \Lambda_{I\cup\{a\}}^{O-\{a\}} \rangle$

$\underset{\text{2.6.c}}{\Longrightarrow} \langle \boldsymbol{test_b}, F_2^1, {}^\star\mathcal{I}_{p+45}^{p+34}, {}^\star\mathcal{I}_{p+33}^{p+22}, Y, \mathcal{I}_{p+19}^{p+1}, D_2, Z, \boldsymbol{\Lambda_{I\cup\{a\}}^{O-\{a\}}} \rangle$

$\underset{\text{2.8.b}}{\Longrightarrow} \langle \boldsymbol{F_2^1}, {}^\star\mathcal{I}_{p+45}^{p+34}, {}^\star\mathcal{I}_{p+33}^{p+22}, Y, \mathcal{I}_{p+19}^{p+1}, D_2, Z, \Lambda_{I\cup\{a,b\}}^{O-\{a,b\}} \rangle$

$\underset{\text{2.7.b}}{\Longrightarrow} \langle \boldsymbol{{}^\star\mathcal{I}_{p+60}^{p+46}}, \boldsymbol{{}^\star\mathcal{I}_{p+45}^{p+34}}, \boldsymbol{{}^\star\mathcal{I}_{p+33}^{p+22}}, Y, \mathcal{I}_{p+19}^{p+1}, \boldsymbol{D_2}, Z, \Lambda_{I\cup\{a,b\}}^{O-\{a,b\}} \rangle$

$\underset{\text{2.4.}}{\Longrightarrow} \langle Y, \mathcal{I}_{p+19}^{p+1}, \mathcal{I}_{p+62}^{p+20}, Z, \Lambda_{I\cup\{a,b\}}^{O-\{a,b\}} \rangle$

$= \langle Y, \mathcal{I}_{p+62}^{p+1}, Z, \Lambda_{I\cup\{a,b\}}^{O-\{a,b\}} \rangle$

$$
\begin{aligned}
S_5 &= \langle \boldsymbol{test_b},\ put_2,\ test_a,\ F_2^2,\ G'_2,\ {}^\star F_1^2,\ H'_2,\ {}^\star\mathcal{I}_{p+33}^{p+22},\ Y,\ D_1,\ D_2,\ Z,\ \boldsymbol{\Lambda_I^O} \rangle \\
&\underset{\text{2.8.b}}{\Longrightarrow} \langle \boldsymbol{put_2},\ test_a,\ F_2^2,\ \boldsymbol{G'_2},\ {}^\star F_1^2,\ \boldsymbol{H'_2},\ {}^\star\mathcal{I}_{p+33}^{p+22},\ Y,\ D_1,\ D_2,\ Z,\ \Lambda_{I\cup\{a\}}^{O-\{a\}} \rangle \\
&\underset{\text{2.6.b}}{\Longrightarrow} \langle \boldsymbol{F_1^2},\ G''_2,\ test_a,\ F_2^2,\ H''_2,\ {}^\star\mathcal{I}_{p+33}^{p+22},\ Y,\ D_1,\ D_2,\ Z,\ \Lambda_{I\cup\{a\}}^{O-\{a\}} \rangle \\
&\underset{\text{2.7.c}}{\Longrightarrow} \langle {}^\star\boldsymbol{\mathcal{I}_{p+17}^{p+3}},\ G''_2,\ test_a,\ F_2^2,\ H''_2,\ {}^\star\mathcal{I}_{p+33}^{p+22},\ Y,\ \boldsymbol{D_1},\ D_2,\ Z,\ \Lambda_{I\cup\{a\}}^{O-\{a\}} \rangle \\
&\underset{\text{2.4.}}{\Longrightarrow} \langle \boldsymbol{G''_2},\ test_a,\ F_2^2,\ \boldsymbol{H''_2},\ {}^\star\mathcal{I}_{p+33}^{p+22},\ Y,\ \mathcal{I}_{p+19}^{p+1},\ D_2,\ Z,\ \Lambda_{I\cup\{a\}}^{O-\{a\}} \rangle \\
&\underset{\text{2.6.c}}{\Longrightarrow} \langle \boldsymbol{test_a},\ F_2^2,\ {}^\star\mathcal{I}_{p+45}^{p+34},\ {}^\star\mathcal{I}_{p+33}^{p+22},\ Y,\ \mathcal{I}_{p+19}^{p+1},\ D_2,\ Z,\ \boldsymbol{\Lambda_{I\cup\{a\}}^{O-\{a\}}} \rangle \\
&\underset{\text{2.8.b}}{\Longrightarrow} \langle \boldsymbol{F_2^2},\ {}^\star\mathcal{I}_{p+45}^{p+34},\ {}^\star\mathcal{I}_{p+33}^{p+22},\ Y,\ \mathcal{I}_{p+19}^{p+1},\ D_2,\ Z,\ \Lambda_{I\cup\{a,b\}}^{O-\{a,b\}} \rangle \\
&\underset{\text{2.7.c}}{\Longrightarrow} \langle {}^\star\boldsymbol{\mathcal{I}_{p+60}^{p+46}},\ {}^\star\boldsymbol{\mathcal{I}_{p+45}^{p+34}},\ {}^\star\boldsymbol{\mathcal{I}_{p+33}^{p+22}},\ Y,\ \mathcal{I}_{p+19}^{p+1},\ \boldsymbol{D_2},\ Z,\ \Lambda_{I\cup\{a,b\}}^{O-\{a,b\}} \rangle \\
&\underset{\text{2.4.}}{\Longrightarrow} \langle Y,\ \mathcal{I}_{p+19}^{p+1},\ \mathcal{I}_{p+62}^{p+20},\ Z,\ \Lambda_{I\cup\{a,b\}}^{O-\{a,b\}} \rangle \\
&= \langle Y,\ \mathcal{I}_{p+62}^{p+1},\ Z,\ \Lambda_{I\cup\{a,b\}}^{O-\{a,b\}} \rangle
\end{aligned}
$$

□

2.4 Reduction from 3-SAT

Let ϕ be a boolean formula over l variables in conjunctive normal form, such that each clause contains exactly three literals. We write k for the number of clauses, $m = 3k$ the total number of literals, and $\{\lambda_1, \dots, \lambda_m\}$ for the set of literals. Let $n = 31l + 62k + 12m$ (thus, $n = 31l + 98k$).

Definition 2.8 (Sequence S_ϕ)**.** *We define the sequence S_ϕ as the permutation of $[\![1\,;\,n]\!]$ obtained by:*

$$
\begin{aligned}
&(key_1, \dots, key_m, test_1, \dots, test_m, \Lambda) = \mathsf{Literals}(31l + 62k, m) \\
&\forall i \in [\![1\,;\,l]\!]\,, \quad P_i = \{j \in [\![1\,;\,m]\!] \mid \lambda_j = x_i\} \\
&\qquad\qquad\quad N_i = \{j \in [\![1\,;\,m]\!] \mid \lambda_j = \neg x_i\} \\
&\qquad\qquad\quad (\nu_i, V_i, D_i) = \mathsf{Variable}(P_i, N_i, 31(i-1)), \\
&\forall i \in [\![1\,;\,k]\!]\,, \quad (a_i, b_i, c_i) = \text{ indices such that the } i\text{-th clause of } \phi \text{ is } \lambda_{a_i} \vee \lambda_{b_i} \vee \lambda_{c_i} \\
&\qquad\qquad\quad (\gamma_i, \Gamma_i, \Delta_i) = \mathsf{Clause}(a_i, b_i, c_i, 31l + 62(i-1)) \\
&S_\phi = \langle \nu_1, \dots, \nu_l, \gamma_1, \dots, \gamma_k, V_1, \dots, V_l, \Gamma_1, \dots, \Gamma_k, D_1, \dots, D_l, \Delta_1, \dots, \Delta_k, \Lambda_\emptyset^\emptyset \rangle
\end{aligned}
$$

Two things should be noted in this definition. First, elements key_i and $test_i$ are used in the clause and variable gadgets, although they are not explicitly stated in the parameters (cf. Definitions 2.6 and 2.7). Second, one could assume that literals are sorted in the formula ($\phi = (\lambda_1 \vee \lambda_2 \vee \lambda_3) \wedge \dots$), so that $a_i = 3i - 2$, $b_i = 3i - 1$ and $c_i = 3i$, but it is not necessary since these values are not used in the following.

We now aim at proving Theorem 2.18 (p. 71), which states that S_ϕ is efficiently sortable if the formula ϕ is satisfiable. Several preliminary lemmas are necessary, and the overall process is illustrated in Figure 2.6.

$$S_\phi = \langle \nu_1, \ldots, \nu_l, \gamma_1, \ldots, \gamma_k, V_1, \ldots, V_l, \Gamma_1, \ldots, \Gamma_k, D_1, \ldots, D_l, \Delta_1, \ldots, \Delta_k, \Lambda^{\emptyset}_{\emptyset} \rangle$$

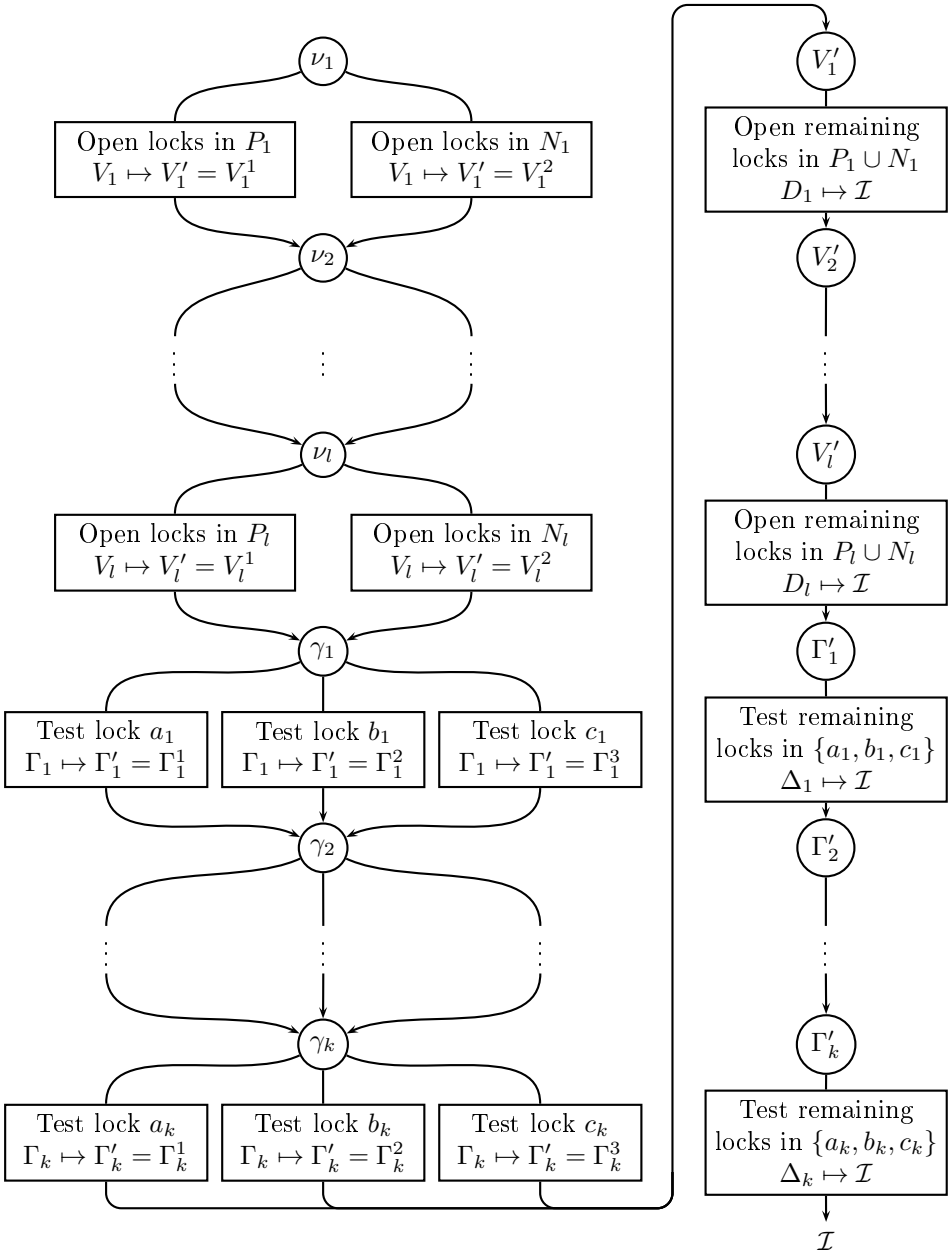


Figure 2.6: Description of an efficient sorting of S_ϕ (Definition 2.8). Circular nodes correspond to *landmarks*, that is, head elements or sequences especially relevant. We start with the head element of S_ϕ: ν_1. From each landmark, one, two or three paths are possible before reaching the next landmark, each path having its own effects, stated in rectangles, on the sequence. Possible effects are: transforming a subsequence of S_ϕ (symbol $\mapsto$), opening a lock, testing a lock (such a path requires the lock to be open); indices are removed from identity sequences ($\mathcal{I}$) for readability. The top-left quarter, from ν_1 to ν_l, is studied in Section 2.4.1; the bottom-left quarter, from γ_1 to γ_k, is studied in Section 2.4.2; and the right half, from V'_1 to Γ'_k, is studied in Section 2.4.3.

2.4.1 Variable Assignment

Definition 2.9 (Assignment)**.** *Let $r \in [\![0\,;\,l]\!]$. An* r-assignment *is a partition $\mathcal{P} = (T, F)$ of $[\![1\,;\,r]\!]$. An l-assignment is called a* full *assignment. Using notations from Definition 2.8, we define the sequence $S_\phi[\mathcal{P}]$ by:*

$$\text{For all } i \in [\![1\,;\,r]\!], \quad V'_i = \begin{cases} V_i^1 \text{ if } i \in T \\ V_i^2 \text{ if } i \in F \end{cases}$$

$$O = \bigcup_{i \in T} P_i \cup \bigcup_{i \in F} N_i$$

$$S_\phi[\mathcal{P}] = \big\langle \nu_{r+1}, \dots, \nu_l, \gamma_1, \dots, \gamma_k, V'_1, \dots, V'_r, V_{r+1}, \dots, V_l, \Gamma_1, \dots, \Gamma_k, D_1, \dots, D_l, \Delta_1, \dots, \Delta_k, \Lambda_\emptyset^O \big\rangle$$

Property 2.12. *Let $r \in [\![0\,;\,l]\!]$ with $r < l$, $\mathcal{P} = (T, F)$ be any r-assignment, $\mathcal{P}_1 = (T \cup \{r+1\}, F)$ and $\mathcal{P}_2 = (T, F \cup \{r+1\})$. Then*

$$S_\phi[\mathcal{P}] \Longrightarrow \{S_\phi[\mathcal{P}_1], S_\phi[\mathcal{P}_2]\}$$

Proof. This is a direct application of Property 2.9.a on variable $(\nu_{r+1}, V_{r+1}, D_{r+1})$, using sequences:

$$X = \big\langle \nu_{r+2}, \dots, \nu_l, \gamma_1, \dots, \gamma_k, V'_1, \dots, V'_r \big\rangle$$
$$Y = \big\langle V_{r+2}, \dots, V_l, \Gamma_1, \dots, \Gamma_k, D_1, \dots, D_l, \Delta_1, \dots, \Delta_k \big\rangle$$

□

With the following lemma, we ensure that any sequence of efficient flips from S_ϕ begins with a full assignment of the boolean variables, and every possible assignment can be reached using only efficient flips.

Lemma 2.13.

$$S_\phi \Longrightarrow \{S_\phi[\mathcal{P}] \mid \mathcal{P} \textit{ full assignment}\}$$

Proof. We prove $S_\phi \Longrightarrow \{S_\phi[\mathcal{P}] \mid \mathcal{P}\ r\text{-assignment}\}$ by induction for all $r \in [\![0\,;\,l]\!]$, and the lemma is deduced from the case $r = l$.

There is only one 0-assignment, which is $\mathcal{P}_0 = (\emptyset, \emptyset)$, and $S_\phi = S_\phi[\mathcal{P}_0]$. Consider now any $r < l$. We use notations $\mathcal{P}_1$ and $\mathcal{P}_2$ from Property 2.12. Then any $(r+1)$-assignment can be written $\mathcal{P}_1$ or $\mathcal{P}_2$, where $\mathcal{P}$ is some r-assignment. We have

$$S_\phi \Longrightarrow \{S_\phi[\mathcal{P}] \mid \mathcal{P}\ r\text{-assignment}\} \text{ by induction hypothesis}$$
$$S_\phi \Longrightarrow \{S_\phi[\mathcal{P}_1], S_\phi[\mathcal{P}_2] \mid \mathcal{P}\ r\text{-assignment}\} \text{ by Property 2.12}$$
$$= \{S_\phi[\mathcal{P}'] \mid \mathcal{P}'\ (r+1)\text{-assignment}\}$$

□

2.4.2 Going through Clauses

Now that each variable is assigned a boolean value, we need to verify with each clause that this assignment satisfies the formula ϕ. This is done by selecting, for each clause, a literal which is true, and testing the corresponding lock. As in Definition 2.8, for any $i \in [\![1\,;\,k]\!]$ we write (a_i, b_i, c_i) for the indices such that the i-th clause of ϕ is $\lambda_{a_i} \vee \lambda_{b_i} \vee \lambda_{c_i}$ (thus, $a_i, b_i, c_i \in [\![1\,;\,m]\!]$).

Definition 2.10 (Selection)**.** *Let $t \in [\![0\,;\,k]\!]$ and $\mathcal{P}$ be a full assignment. A t-*selection *σ is a subset of $[\![1\,;\,m]\!]$ such that*

- *$|\sigma| = t$*
- *for each $i \in [\![1\,;\,t]\!]$, $|\{a_i, b_i, c_i\} \cap \sigma| = 1$*

A t-selection σ and a full assignment $\mathcal{P} = (T, F)$ are compatible*, if, for every $i \in \sigma$, literal λ_i is true according to assignment $\mathcal{P}$ (that is, $\lambda_i = x_j$ and $j \in T$, or $\lambda_i = \neg x_j$ and $j \in F$).*

A k-selection is called a full selection*. Given a t-selection σ and a full assignment $\mathcal{P} = (T, F)$ which are compatible, we define the sequence $S_\phi[\mathcal{P}, \sigma]$ by:*

$$\text{For all } i \in [\![1\,;\,l]\!]\,, \quad V'_i = \begin{cases} V^1_i \text{ if } i \in T \\ V^2_i \text{ if } i \in F \end{cases}$$

$$\text{For all } i \in [\![1\,;\,t]\!]\,, \quad \Gamma'_i = \begin{cases} \Gamma^1_i \text{ if } a_i \in \sigma \\ \Gamma^2_i \text{ if } b_i \in \sigma \\ \Gamma^3_i \text{ if } c_i \in \sigma \end{cases}$$

$$O = \bigcup_{i \in T} P_i \cup \bigcup_{i \in F} N_i - \sigma$$

$$I = \sigma$$

$$S_\phi[\mathcal{P}, \sigma] = \langle \gamma_{t+1}, \ldots, \gamma_k, V'_1, \ldots, V'_l, \Gamma'_1, \ldots, \Gamma'_t, \Gamma_{t+1}, \ldots, \Gamma_k, D_1, \ldots, D_l, \Delta_1, \ldots, \Delta_k, \Lambda^O_I \rangle$$

We now aim at proving Lemma 2.16, which ensures that after the truth assignment, every efficient path starting from S_ϕ needs to select a literal in each clause, under the constraint that the selection is compatible with the assignment. We will use the following two properties.

Property 2.14. *Let $\mathcal{P}$ be a full assignment and $t \in [\![0\,;\,k]\!]$, $t < k$. Let σ' be a $(t+1)$-selection compatible with $\mathcal{P}$, then there exists a t-selection σ compatible with $\mathcal{P}$ such that $\sigma \subset \sigma'$.*

Proof. It is obtained by $\sigma = \sigma' - \{a_{t+1}, b_{t+1}, c_{t+1}\}$. It is trivially a t-selection included in σ, and it is compatible with $\mathcal{P}$ (all selected literals in σ are also selected in σ', and thus are true according to $\mathcal{P}$). □

Property 2.15. *Let $t \in [\![0\,;\,k]\!]$, $t < k$, $\mathcal{P}$ be a full assignment, and σ be a t-selection compatible with $\mathcal{P}$.*

$$S_\phi[\mathcal{P}, \sigma] \Longrightarrow \{S_\phi[\mathcal{P}, \sigma'] \mid \sigma' \ (t+1)\text{-selection compatible with } \mathcal{P}; \sigma \subset \sigma'\}$$

Note that the right-hand side can be the empty set, in which case $S_\phi[\mathcal{P}, \sigma] \Longrightarrow \emptyset$.

Proof. First note that there are 3 $(t+1)$-selections σ' such that $\sigma \subset \sigma'$, and they are $\sigma'_1 = \sigma \cup \{a_{t+1}\}$, $\sigma'_2 = \sigma \cup \{b_{t+1}\}$, and $\sigma'_3 = \sigma \cup \{c_{t+1}\}$. Since σ is compatible with $\mathcal{P}$, σ'_1 is compatible with $\mathcal{P}$ iff literal $\lambda_{a_{t+1}}$ is true in $\mathcal{P}$ (and similarly with couples $(\sigma'_2, \lambda_{b_{t+1}})$ and $(\sigma'_3, \lambda_{c_{t+1}})$). We now define sequences X and Y and sets I and O such that $S_\phi[\mathcal{P}, \sigma] = \langle \gamma_{t+1}, X, \Gamma_{t+1}, Y, \Lambda_I^O \rangle$, that is:

$$\begin{aligned} X &= \langle \gamma_{t+2}, \ldots, \gamma_k, V'_1, \ldots, V'_l, \Gamma'_1, \ldots, \Gamma'_t \rangle \\ Y &= \langle \Gamma_{t+2}, \ldots, \Gamma_k, D_1, \ldots, D_l, \Delta_1, \ldots, \Delta_k, \rangle \\ O &= \bigcup_{i \in T} P_i \cup \bigcup_{i \in F} N_i - \sigma \\ I &= \sigma \end{aligned}$$

Using Property 2.10 on clause gadget $(\gamma_{t+1}, \Gamma_{t+1}, \Delta_{t+1})$, we obtain:

$$S_\phi[\mathcal{P}, \sigma] \Longrightarrow \mathbb{T}$$

where $\mathbb{T}$ is defined by:

$$\begin{aligned} \langle X, \Gamma^1_{t+1}, Y, \Lambda^{O-\{a_{t+1}\}}_{I \cup \{a_{t+1}\}} \rangle \in \mathbb{T} \quad &\text{iff} \quad a_{t+1} \in O \\ \langle X, \Gamma^2_{t+1}, Y, \Lambda^{O-\{b_{t+1}\}}_{I \cup \{b_{t+1}\}} \rangle \in \mathbb{T} \quad &\text{iff} \quad b_{t+1} \in O \\ \langle X, \Gamma^3_{t+1}, Y, \Lambda^{O-\{c_{t+1}\}}_{I \cup \{c_{t+1}\}} \rangle \in \mathbb{T} \quad &\text{iff} \quad c_{t+1} \in O \end{aligned}$$

Note that $a_{t+1} \notin \sigma$, hence $a_{t+1} \in O$ iff $\exists i \in T$ s.t. $a_{t+1} \in P_i$ or $\exists i \in F$ s.t. $a_{t+1} \in N_i$. Equivalently, $a_{t+1} \in O$ iff $\lambda_{a_{t+1}}$ is a positive occurrence of a variable assigned True in $\mathcal{P}$, or a negative occurrence of a variable assigned False in $\mathcal{P}$. Finally, $a_{t+1} \in O$ iff σ'_1 is compatible with $\mathcal{P}$. Likewise, $b_{t+1} \in O$ iff σ'_2 is compatible with $\mathcal{P}$, and $c_{t+1} \in O$ iff σ'_3 is compatible with $\mathcal{P}$.

$$\begin{aligned} S_\phi[\mathcal{P}, \sigma'_1] = \langle X, \Gamma^1_{t+1}, Y, \Lambda^{O-\{a_{t+1}\}}_{I \cup \{a_{t+1}\}} \rangle \in \mathbb{T} \quad &\text{iff} \quad \sigma'_1 \text{ is compatible with } \mathcal{P} \\ S_\phi[\mathcal{P}, \sigma'_2] = \langle X, \Gamma^2_{t+1}, Y, \Lambda^{O-\{b_{t+1}\}}_{I \cup \{b_{t+1}\}} \rangle \in \mathbb{T} \quad &\text{iff} \quad \sigma'_2 \text{ is compatible with } \mathcal{P} \\ S_\phi[\mathcal{P}, \sigma'_3] = \langle X, \Gamma^3_{t+1}, Y, \Lambda^{O-\{c_{t+1}\}}_{I \cup \{a_{t+1}\}} \rangle \in \mathbb{T} \quad &\text{iff} \quad \sigma'_3 \text{ is compatible with } \mathcal{P} \end{aligned}$$

Thus $\mathbb{T}$ is indeed the set of sequences $S_\phi[\mathcal{P}, \sigma']$ where σ' is a $(t+1)$-selection which contains σ and is compatible with $\mathcal{P}$: the property is proved. □

Lemma 2.16. *Let $\mathcal{P}$ be a full assignment. Then*

$$S_\phi[\mathcal{P}] \Longrightarrow \{S_\phi[\mathcal{P}, \sigma] \mid \sigma \textit{ full selection compatible with } \mathcal{P}\}$$

Proof. The proof follows the same pattern as the one of Lemma 2.13, that is, we prove

$$S_\phi[\mathcal{P}] \Longrightarrow \{S_\phi[\mathcal{P}, \sigma] \mid \sigma \ t\text{-selection compatible with } \mathcal{P}\}$$

by induction for all $t \in [\![0\,;\, k]\!]$, and the lemma is deduced from the case $t = k$.

There is only one 0-selection, which is $\sigma_0 = \emptyset$, it is compatible with $\mathcal{P}$, and $S_\phi[\mathcal{P}] = S_\phi[\mathcal{P}, \sigma_0]$. Consider now any $t < k$. We have

$$\begin{aligned} S_\phi[\mathcal{P}] &\Longrightarrow \{S_\phi[\mathcal{P}, \sigma] \mid \sigma \ t\text{-selection compatible with } \mathcal{P}\} \text{ (by induction hypothesis)} \\ S_\phi[\mathcal{P}] &\Longrightarrow \{S_\phi[\mathcal{P}, \sigma'] \mid \sigma' \ (t+1)\text{-selection compatible with } \mathcal{P} \text{ and} \\ &\qquad \exists\, \sigma \ t\text{-selection compatible with } \mathcal{P},\ \sigma \subset \sigma'\} \text{ by Property 2.15} \\ &= \{S_\phi[\mathcal{P}, \sigma'] \mid \sigma' \ (t+1)\text{-selection compatible with } \mathcal{P}\} \text{ by Property 2.14} \end{aligned}$$

□

2.4.3 Beyond Clauses

Lemma 2.17. *Let $\mathcal{P}$ be a full assignment and σ be a full selection, such that $\mathcal{P}$ and σ are compatible (provided such a pair exists for ϕ). Then*

$$S_\phi[\mathcal{P},\sigma] \Longrightarrow \mathcal{I}_n^1$$

Proof. Write $\mathcal{P} = (T,F)$. Since σ is a full selection, $S_\phi[\mathcal{P},\sigma]$ can be written (see Definition 2.10):

$$\text{For all } i \in [\![1\,;\,l]\!]\,, \quad V'_i = \begin{cases} V_i^1 \text{ if } i \in T \\ V_i^2 \text{ if } i \in F \end{cases}$$

$$\text{For all } i \in [\![1\,;\,k]\!]\,, \quad \Gamma'_i = \begin{cases} \Gamma_i^1 \text{ if } a_i \in \sigma \\ \Gamma_i^2 \text{ if } b_i \in \sigma \\ \Gamma_i^3 \text{ if } c_i \in \sigma \end{cases}$$

$$O = \bigcup_{i\in T} P_i \cup \bigcup_{i\in F} N_i - \sigma$$

$$I = \sigma$$

$$S_\phi[\mathcal{P},\sigma] = \left\langle V'_1,\dots,V'_l,\Gamma'_1,\dots,\Gamma'_k,D_1,\dots,D_l,\Delta_1,\dots,\Delta_k,\Lambda_I^O\right\rangle$$

We extend the definition of set O to O_r, for any $r \in [\![0\,;\,l]\!]$, as follows:

$$O_r = \bigcup_{0<i\le r} (P_i \cup N_i) \cup \bigcup_{i\in T} P_i \cup \bigcup_{i\in F} N_i - \sigma$$

Note that $O_0 = O$, and that $O_l = [\![1\,;\,m]\!] - \sigma$.

$$\begin{aligned} S_\phi[\mathcal{P},\sigma] &= \left\langle \boldsymbol{V'_1},\dots,V'_l,\Gamma'_1,\dots,\Gamma'_k,\boldsymbol{D_1},\dots,D_l,\Delta_1,\dots,\Delta_k,\boldsymbol{\Lambda_I^{O_0}}\right\rangle \\ &\underset{\text{2.9.b/c}}{\Longrightarrow} \left\langle \boldsymbol{V'_2},\dots,V'_l,\Gamma'_1,\dots,\Gamma'_k,\mathcal{I}_{31}^1,\boldsymbol{D_2}\dots,D_l,\Delta_1,\dots,\Delta_k,\boldsymbol{\Lambda_I^{O_1}}\right\rangle \\ &\underset{\text{2.9.b/c}}{\Longrightarrow} \left\langle \boldsymbol{V'_3},\dots,V'_l,\Gamma'_1,\dots,\Gamma'_k,\mathcal{I}_{31}^1,\mathcal{I}_{62}^{32},\boldsymbol{D_3}\dots,D_l,\Delta_1,\dots,\Delta_k,\boldsymbol{\Lambda_I^{O_2}}\right\rangle \\ &\dots \\ &\underset{\text{2.9.b/c}}{\Longrightarrow} \left\langle \Gamma'_1,\dots,\Gamma'_k,\boldsymbol{\mathcal{I}_{31}^1,\mathcal{I}_{62}^{32},\dots,\mathcal{I}_{31l}^{31l-30}},\Delta_1,\dots,\Delta_k,\Lambda_I^{O_l}\right\rangle \\ &= \left\langle \Gamma'_1,\dots,\Gamma'_k,\mathcal{I}_{31l}^1,\Delta_1,\dots,\Delta_k,\Lambda_I^{O_l}\right\rangle \end{aligned}$$

Finally, for the last part, we use a similar procedure, with the following sets, for $t \in [\![0\,;\,k]\!]$:

$$O'_t = [\![1\,;\,m]\!] - \left(\sigma \cup \bigcup_{0<i\le t} \{a_i,b_i,c_i\}\right)$$

$$I'_t = \sigma \cup \bigcup_{0<i\le t} \{a_i,b_i,c_i\}$$

Note that $O'_0 = O_l$, $I'_0 = I$, $O'_k = \emptyset$, $I'_k = [\![1\,;\,m]\!]$, and more importantly, for $i > t$, assuming that $a_i \in \sigma$ (cases $b_i \in \sigma$ and $c_i \in \sigma$ are similar), then $a_i \in I'_t$,

$b_i \in O'_t$ and $c_i \in O'_t$. Hence we can successively apply Property 2.11 (either .a, .b or .c) on each clause gadgets.

$$
\begin{aligned}
&\left\langle \boldsymbol{\Gamma'_1}, \dots, \Gamma'_k, \mathcal{I}^1_{31l}, \boldsymbol{\Delta_1}, \dots, \Delta_k, \boldsymbol{\Lambda^{O'_0}_{I'_0}} \right\rangle \\
&\underset{2.11.}{\Longrightarrow} \left\langle \boldsymbol{\Gamma'_2}, \dots, \Gamma'_k, \mathcal{I}^1_{31l}, \mathcal{I}^{31l+1}_{31l+62}, \boldsymbol{\Delta_2}, \dots, \Delta_k, \boldsymbol{\Lambda^{O'_1}_{I'_1}} \right\rangle \\
&\underset{2.11.}{\Longrightarrow} \left\langle \boldsymbol{\Gamma'_3}, \dots, \Gamma'_k, \mathcal{I}^1_{31l}, \mathcal{I}^{31l+1}_{31l+62}, \mathcal{I}^{31l+63}_{31l+124}, \boldsymbol{\Delta_3}, \dots, \Delta_k, \boldsymbol{\Lambda^{O'_2}_{I'_2}} \right\rangle \\
&\dots \\
&\underset{2.11.}{\Longrightarrow} \left\langle \mathcal{I}^1_{31l}, \boldsymbol{\mathcal{I}^{31l+1}_{31l+62}, \mathcal{I}^{31l+63}_{31l+124}, \dots, \mathcal{I}^{31l+62k-61}_{31l+62k}}, \Lambda^{O'_k}_{I'_k} \right\rangle \\
&= \left\langle \mathcal{I}^1_{31l}, \mathcal{I}^{31l+1}_{31l+62k}, \Lambda^{\emptyset}_{[\![1\,;m]\!]} \right\rangle \\
&= \left\langle \mathcal{I}^1_{31l}, \mathcal{I}^{31l+1}_{31l+62k}, \mathcal{I}^{31l+62k+1}_{31l+62k+12m} \right\rangle \\
&= \mathcal{I}^1_n
\end{aligned}
$$

□

Theorem 2.18. *For any boolean formula ϕ instance of 3-SAT,*

$$S_\phi \Longrightarrow \mathcal{I}^1_n \text{ iff } \phi \text{ is satisfiable.}$$

Proof. Assume first that $S_\phi \Longrightarrow \mathcal{I}^1_n$. By Lemma 2.13, since

$$S_\phi \Longrightarrow \{S_\phi[\mathcal{P}] \mid \mathcal{P} \text{ full assignment}\},$$

there exists a full assignment $\mathcal{P} = (T, F)$ such that some path from S_ϕ to the identity uses $S_\phi[\mathcal{P}]$. Note that $S_\phi[\mathcal{P}] \Longrightarrow \mathcal{I}^1_n$. Now, by Lemma 2.16, since

$$S_\phi[\mathcal{P}] \Longrightarrow \{S_\phi[\mathcal{P}, \sigma] \mid \sigma \text{ full selection compatible with } \mathcal{P}\},$$

there exists a full selection σ, compatible with $\mathcal{P}$, such that some path from $S_\phi[\mathcal{P}]$ to the identity uses $S_\phi[\mathcal{P}, \sigma]$. Consider the truth assignment $x_i := \text{True} \Leftrightarrow i \in T$. Then each clause of ϕ contains at least one literal that is true (the literal whose index is in σ), and thus ϕ is satisfiable.

Assume now that ϕ is satisfiable: consider any truth assignment making ϕ true, write T the set of indices such that $x_i = \text{True}$, and $F = [\![1\,;\,l]\!] - T$. Write also σ a set containing, for each clause of ϕ, the index of one literal being true under this assignment. Then σ is a full selection, compatible with the full assignment $\mathcal{P} = (T, F)$. By Lemmas 2.13, 2.16 and 2.17 respectively, there exist efficient paths $S_\phi \Longrightarrow S_\phi[\mathcal{P}]$, $S_\phi[\mathcal{P}] \Longrightarrow S_\phi[\mathcal{P}, \sigma]$ and $S_\phi[\mathcal{P}, \sigma] \Longrightarrow \mathcal{I}^1_n$. Thus sequence S_ϕ is efficiently sortable. □

Using Theorem 2.18, we can now prove the main result of this chapter.

Theorem 2.19. *Sorting By Prefix Reversals is* NP*-hard.*

Proof. By reduction from 3-SAT. Given any formula ϕ, create S_ϕ (see Definition 2.8, the construction requires a linear time). By Theorem 2.18, the minimum number of flips necessary to sort S_ϕ is $d_b(S_\phi)$ iff ϕ is satisfiable. □

Corollary 2.20. *Deciding, given a sequence S, whether S can be sorted in $d_b(S)$ flips, is an* NP*-hard problem.*

Conclusion

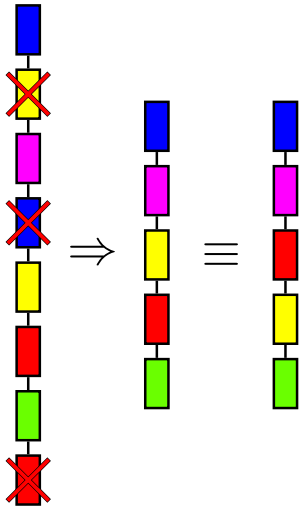
In this chapter, we have shown that the Pancake Flipping problem, i.e. the Sorting by Prefix Reversals problem is NP-hard, thus answering a long-standing open question. We have also provided a stronger result, namely, deciding whether a permutation can be sorted with no more than one flip per breakpoint is also NP-hard. However, the approximability of SBPR is still open: it can be seen that sequence S_ϕ can be sorted in $d_b(S_\phi) + 2$ flips, whatever the formula ϕ, hence our construction does not prove the APX-hardness of the problem.

Among related important problems, the last one having an open complexity is now the burnt variant of the Pancake Flipping problem. An interesting insight into this problem is given in a recent work from Labarre and Cibulka [100], where the authors characterize a non-trivial subclass of permutations that can be sorted in polynomial time, using the breakpoint graph [11]. Another development consists in trying to improve the approximation ratio of 2 for the Pancake Flipping problem, both in its burnt and unburnt versions.

Distances Between Strings

3

Exemplar Distances



The Exemplar Distance problem asks, given two input genomes with duplicates, to extract exemplarizations of the genomes (i.e. subsequences containing exactly one copy of each gene) minimizing a given dissimilarity measure.

In this chapter, we prove that the Exemplar Distance problem is NP-hard to approximate for the rearrangement distances sorting by reversal and sorting by DCJ, and for edit distances such as Hamming and Levenshtein distances.

The results in this chapter have been obtained through a joint work with Minghui Jiang. They have been presented at the *8th International Symposium on Bioinformatics Research and Applications* (ISBRA 2012, Dallas [39]), and accepted for publication in *IEEE Transactions on Computational Biology and Bioinformatics* (TCBB 2013 [40]).

Introduction

In this second part of the manuscript, we now consider comparative genomics problems where genomes are represented no longer as permutations, but as strings. In this case, several copies of the same gene may occur in the same chromosome, and we no longer have the complete information about how markers of one sequence should be matched with markers of the other sequence. The objective of a number of string-based comparative genomics problems is to understand the separate evolution history of each copy of the genes since the common ancestor of the studied species. With this understanding, it is possible to obtain a one-to-one matching of the markers, enabling further and more complex computations.

In this chapter, a genome is thus represented by a sequence of signed integers (or *markers*): each integer represents a gene of a given gene family, and its sign represents its orientation. Given two genomes possibly with duplicate genes, the *exemplar distance* problem [115] is that of removing all but one copy of each gene in each genome, so as to minimize the distance between the two reduced genomes according to some measure. The reduced genomes are said to be *exemplar subsequences* of the original genomes. This approach amounts to considering that, in the evolution history, duplications have taken place after the speciation of the genomes (or more generally, that we are able to distinguish genes that have been duplicated before the speciation). Hence, in each genome, only one copy of each gene may be matched to an ortholog gene in the other genome.

For example, the following two sequences

$$\begin{aligned} G_1 = &\quad \langle -4, +1, +2, +3, -5, +1, +2, +3, -6 \rangle \\ G_2 = &\quad \langle -1, -4, +1, +2, -5, +3, -2, -6, +3, +4 \rangle \end{aligned}$$

can both be reduced to the same exemplar subsequence by removing duplicates:

$$G' = \quad \langle -4, +1, +2, -5, +3, -6 \rangle$$

They thus have exemplar distance zero for any reasonable distance measure. In general, unless we are to decide simply whether two genomes can be reduced to the same genome by removing duplicates, the exemplar distance problem is not a single problem but a group of related problems because the choice of the distance measure is not unique. We refer to Figure 3.1 for an example scenario where the underlying distance measure is the signed reversal distance.

We denote by (s,t)-Exemplar Distance the exemplar distance problem on two sequences G_1 and G_2 where each marker occurs at most s times in G_1 and at most t times in G_2 (we say that G_1 and G_2 have *maximal occurrences* s and t respectively). It is known [26, 96] that for any reasonable distance measure, $(2,2)$-Exemplar Distance does not admit any approximation. This is because to decide simply whether two sequences with maximum occurrence 2 can be reduced to the same subsequence by removing duplicates is already NP-hard. In this chapter, we focus on the simplest non-trivial variant of the exemplar distance problem: $(1,2)$-Exemplar Distance. Note that we use the monochromosomal model of genomes in order to have the simplest definition of the problem: thus, our hardness results can be extended to multichromosomal variants of the problems we consider.

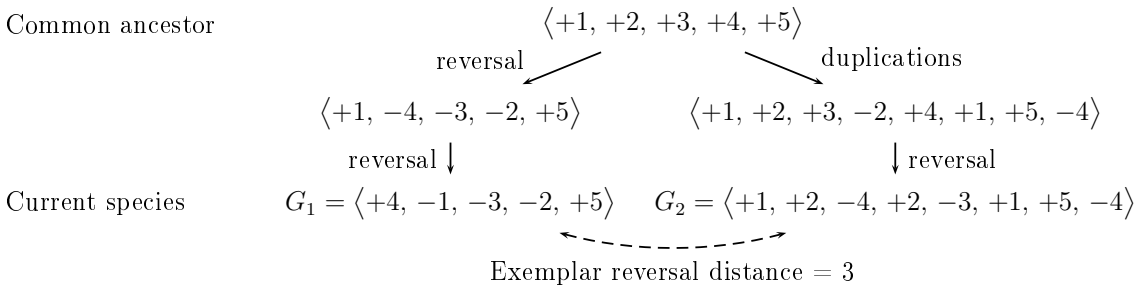


Figure 3.1: During the evolution of two different species with genomes G_1 and G_2 from a common ancestor, duplications occur in G_2, and reversals occur both in G_1 and G_2. By the parsimony principle, the exemplar distance of 3 between G_1 and G_2 corresponds to the number of reversal events in the most likely evolution history of the two species.

The problem $(1,t)$-Exemplar Distance has been studied for several distance measures commonly used in comparative genomics. Angibaud et al. [7] showed that for breakpoint, common interval, and conserved interval distances, the $(1,2)$-Exemplar Distance is APX-hard. Bonizzoni et al. [28] proved that variants of Longest Common Subsequence under exemplar models are also APX-hard. For other measures such as maximum adjacency disruption (MAD) and summed adjacency disruption (SAD, introduced by Sankoff and Haque [116]), computing the exemplar distance is APX-hard [24]. More precisely, we proved in [40] that $(1,2)$-Exemplar MAD Distance is NP-hard to approximate within $2-\epsilon$ for any $\epsilon > 0$, and $(1,2)$-Exemplar SAD Distance is NP-hard to approximate within 1.3606. See also [48, 46] for related results.

For an unsigned permutation $\pi = \langle \pi_1, \ldots, \pi_n \rangle$, an *unsigned reversal* (i,j) with $1 \leq i \leq j \leq n$ turns π into $\langle \pi_1, \ldots, \pi_{i-1},\ \pi_j, \ldots, \pi_i,\ \pi_{j+1}, \ldots, \pi_n \rangle$, where the factor $\pi_i \ldots \pi_j$ is reversed. For a signed permutation $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$, a *signed reversal* (i,j) with $1 \leq i \leq j \leq n$ turns σ into $\langle \sigma_1, \ldots, \sigma_{i-1},\ -\sigma_j, \ldots, -\sigma_i,\ \sigma_{j+1}, \ldots, \sigma_n \rangle$, where the factor $\sigma_i \ldots \sigma_j$ is reversed and negated (see Figure 3.2a). The *unsigned reversal distance* (resp. *signed reversal distance*) between two unsigned (resp. signed) permutations is the minimum number of unsigned (resp. signed) reversals required to transform one to the other. Computing the unsigned reversal distance is APX-hard [20], although the signed reversal distance can be computed in polynomial time [84].

Our first theorem answers an open question of Blin et al. [24] on the inapproximability of the exemplar reversal distance problem:

Theorem 3.1. $(1,2)$-Exemplar Signed Reversal Distance *is* NP*-hard to approximate within* $1237/1236 - \epsilon$ *for any* $\epsilon > 0$.

The double-cut-and-join (DCJ) operation introduced by Yancopoulos et al. [124] consists in cutting the genome in two positions, and joining the four ends in any new way. Note that this operation requires a model more general than sequences to model the genomes, in order to take into account possible circular chromosomes (i.e. with *circular permutations*). In practice, a DCJ operation can correspond to a reversal, to the excision of a factor into a circular permutation, or to the insertion of a circular permutation back into the main sequence, at any position (see Figure 3.2). The problem of computing the DCJ distance between two permutations is known

(a) $\langle +1, +2, \underline{+3, +4, +5}, +6\rangle \longrightarrow \langle +1, +2, -5, -4, -3, +6\rangle$

(b) $\langle +1, +2, \overset{\vartriangle}{} +3, +4, +5, \overset{\vartriangle}{} +6\rangle \longrightarrow \langle +1, +2, +6\rangle\ (+3, +4, +5)$

(c) $\langle +1, +2, \overset{\vartriangle}{} +3\rangle\ (+4, +5, \overset{\vartriangle}{} +6) \longrightarrow \langle +1, +2, -5, -4, -6, +3\rangle$

Figure 3.2: The possible operations allowed for the signed DCJ distance are (a) reversals, (b) excisions, and (c) insertions. We write circular permutations with parentheses, i.e., $(+1\ +2\ +3)$ is equal to $(+2\ +3\ +1)$ and to $(-3\ -2\ -1)$.

to be polynomial in the signed case [124], and is NP-hard in the unsigned case [43]. The following theorem shows the intractability of the exemplar DCJ problem:

Theorem 3.2. $(1,2)$-Exemplar Signed DCJ Distance *is* NP*-hard to approximate within* $1237/1236 - \epsilon$ *for any* $\epsilon > 0$.

In the last theorem of this chapter, we present the first inapproximability result on the exemplar distance problem using the classic string edit distance measure:

Theorem 3.3. $(1,2)$-Exemplar Edit Distance *is* APX*-hard to compute when the cost of a substitution is* 1 *and the cost of an insertion or a deletion is at least* 1.

Note that both the Levenshtein distance and the Hamming distance are special cases of the string edit distance: for Levenshtein distance, the cost of every operation (substitution, insertion, or deletion) is 1; for Hamming distance, the cost of a substitution is 1 and the cost of an insertion or a deletion is $+\infty$. Thus we have the following corollaries:

Corollary 3.4. $(1,2)$-Exemplar Levenshtein Distance *is* APX*-hard.*

Corollary 3.5. $(1,2)$-Exemplar Hamming Distance *is* APX*-hard.*

Our choices of the specific distance measures are based on two considerations. First, for a broader impact, we try to explore a wide variety of distance measures, which are suitable for different requirements of various biological applications: edit distances measure local differences, and reversal and DCJ distances compute global rearrangement schemes. Second, in terms of computational complexity, the exemplar generalization of any measure for sequences with duplicates can only be harder to compute than the basic version of the same measure for sequences without duplicates. In order to obtain unambiguous results on the true difficulty of the exemplar distance problem, we restrict ourselves to measures whose basic versions are easy to compute. For example, given any two sequences, their Hamming distance can be trivially computed in linear time, and their Levenshtein distance can be computed in quadratic time by dynamic programming. Less straightforward but still polynomial-time algorithms exist for signed reversal distance [84] and signed DCJ distance [124].

3.1 Signed Reversal and DCJ Distances

In this section we prove Theorems 3.1 and 3.2. We first show that $(1,2)$-Exemplar Signed Reversal Distance is APX-hard by a reduction from the

problem Sorting by Unsigned Reversals [20], which asks for the minimum number of unsigned reversals to transform a given unsigned permutation into the identity permutation.

Let $\pi = \langle \pi_1, \ldots, \pi_n \rangle$ be an unsigned permutation of $[\![1\,;\,n]\!]$. We construct two sequences $G_1 = \langle +1, \ldots, +n \rangle$ and $G_2 = \langle +\pi_1, -\pi_1, \ldots, +\pi_n, -\pi_n \rangle$.

Lemma 3.6. *π can be sorted into the identity permutation $\mathcal{I}_n$ by at most k unsigned reversals iff G_2 has an exemplar subsequence G_2' with signed reversal distance at most k from G_1.*

Proof. We say that a signed permutation σ is a signed version of π if for all $1 \leq i \leq n$, $\pi_i = |\sigma_i|$. The lemma is based on two key observations. First, the permutation π can be sorted in k reversals iff there exists a signed version σ of π that can be sorted in k (signed) reversals. Second, a signed permutation σ is an exemplar subsequence of G_2 iff it is a signed version of π, that is, for all $1 \leq i \leq n$, $\pi_i = |\sigma_i|$.

The first observation is a classic result: given a sequence of reversals sorting π, the signed version σ of π is constructed by applying the same sequence in reversed order from the signed identity permutation. And conversely, any sequence of signed reversals sorting a signed version of π, seen as a sequence of unsigned reversals, transforms π into the identity. The second observation is obtained by construction of G_2: any signed version of π can be seen as an exemplar subsequence of G_2, and all exemplar subsequences of G_2 are signed versions of π.

The lemma is directly deduced from these two equivalences:
π can be sorted by at most k unsigned reversals
$\Leftrightarrow$ π has a signed version σ that can be sorted by at most k unsigned reversals
$\Leftrightarrow$ G_2 has an exemplar subsequence $G_2' = \sigma$ with signed reversal distance at most k from G_1. □

Since Sorting by Unsigned Reversals is NP-hard to approximate within $1237/1236 - \epsilon$ for any $\epsilon > 0$ [20], $(1,2)$-Exemplar Signed Reversal Distance is NP-hard to approximate within $1237/1236 - \epsilon$ for any $\epsilon > 0$ too.

We now prove Theorem 3.2 by a reduction from Sorting by Unsigned DCJ (see [43]). Given an unsigned permutation π, compose the same sequences G_1 and G_2 as before: $G_1 = \langle +1, \ldots, +n \rangle$ and $G_2 = \langle +\pi_1, -\pi_1, \ldots, +\pi_n, -\pi_n \rangle$. We have the following lemma:

Lemma 3.7. *π can be sorted into the identity permutation $\mathcal{I}_n$ by at most k unsigned DCJs iff G_2 has an exemplar subsequence G_2' with signed DCJ distance at most k from G_1.*

Proof. As in the proof of Lemma 3.6, this result is obtained from the following two equivalences:
π can be sorted by at most k unsigned DCJs
$\Leftrightarrow$ π has a signed version σ that can be sorted by at most k unsigned DCJs
$\Leftrightarrow$ G_2 has an exemplar subsequence $G_2' = \sigma$ with signed DCJ distance at most k from G_1. □

The problem Sorting by Unsigned DCJ has been proved to be NP-hard [43]. We note that it is in fact NP-hard to approximate within $1237/1236 - \epsilon$ for any $\epsilon > 0$ because, according to [43, Theorem 2], Sorting by Unsigned DCJ has the same objective function as Breakpoint Graph Decomposition (formulated as

a minimization problem), and the latter is known to be NP-hard to approximate within $1237/1236-\epsilon$ for any $\epsilon > 0$ [20, Theorem 4]. It follows that $(1,2)$-Exemplar Signed DCJ Distance is also NP-hard to approximate within $1237/1236 - \epsilon$ for any $\epsilon > 0$.

3.2 Edit Distances

In this section we prove Theorem 3.3. For any edit distance where the cost of a substitution is 1 and the cost of an insertion or a deletion is at least 1 (possibly $+\infty$), we show that the problem $(1,2)$-Exemplar Edit Distance is APX-hard by a reduction from the problem Minimum Vertex Cover in Cubic Graphs.

Let $G = (V, E)$ be a graph of n vertices and m edges. It is cubic if all vertices have degree three (hence $3n = 2m$). A *vertex cover* of G is a subset X of V such that for each $e = \{u, v\} \in E$, we have $u \in X$ or $v \in X$.

Problem	Minimum Vertex Cover in Cubic Graphs
Input	A cubic graph $G = (V, E)$
Output	A vertex cover X of G
Maximize	The size of X

The reduction goes as follows. Given a cubic graph $G = (V, E)$, we construct two sequences G_1 and G_2 over an alphabet of

$$3m + 4n + 2(m + 7n) + 2(m - 1) + (n - 1)$$

distinct markers. For each edge $e = \{u, v\} \in E$, we have three edge markers e, e_u, and e_v. For each vertex $v \in V$, we have a vertex marker v and 3 dummy markers v'_1, v'_2, v'_3. In addition, we have $2(m + 7n) + 2(m - 1) + (n - 1)$ markers for separators. The construction is illustrated in Figure 3.3 for the complete graph K_4.

The two sequences G_1 and G_2 are composed from $m + n + 1$ gadgets: an edge gadget for each edge, a vertex gadget for each vertex, and a tail gadget. The $m+n+1$ gadgets are separated by $m+n$ separators of total length $2(m+7n)+2(m-1)+(n-1)$:

- two long separators, each of length $m + 7n$: one between the last edge gadget and the first vertex gadget, one between the last vertex gadget and the tail gadget;
- $m + n - 2$ short separators: a length-2 separator between any two consecutive edge gadgets, and a length-1 separator between any two consecutive vertex gadgets.

For each edge $e = \{u, v\}$, the edge gadget for e is

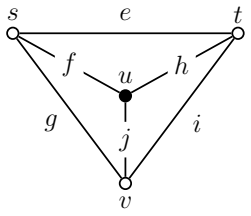
$$G_1[e] = \langle e \rangle$$
$$G_2[e] = \langle e_u, e_v \rangle$$

For each vertex v incident to edges e, f, g, the vertex gadget for v is

$$G_1[v] = \langle v,\ v'_1, v'_2, v'_3 \rangle$$
$$G_2[v] = \langle e_v, f_v, g_v,\ v,\ e,\ f,\ g \rangle$$

Let V' be the $3n$ markers v'_1, v'_2, v'_3 for $v \in V$. Let E' be the $2m = 3n$ markers e_u and e_v for $e = \{u, v\} \in E$. The tail gadget is

$$G_1[tail] = E'$$



$G_1 = \langle\ e,\ \mathbf{S},\ f,\ \mathbf{S},\ g,\ \mathbf{S},\ h,\ \mathbf{S},\ i,\ \mathbf{S},\ j,\ -$
$G_2 = \langle\ e_s, \underline{e_t},\ \underline{\mathbf{S}},\ f_s, \underline{f_u},\ \underline{\mathbf{S}},\ g_s, \underline{g_v},\ \underline{\mathbf{S}},\ h_t, \underline{h_u},\ \underline{\mathbf{S}},\ \underline{i_t},\ i_v,\ \underline{\mathbf{S}},\ \underline{j_u},\ j_v,\ -$

$-\ \mathbf{S},\quad s, s'_1, s'_2, s'_3,\quad \mathbf{S},\quad t, t'_1, t'_2, t'_3,\quad \mathbf{S},\quad \mathbf{u}, u'_1, u'_2, u'_3,\quad \mathbf{S},\quad v, v'_1, v'_2, v'_3,\quad -$
$-\ \underline{\mathbf{S}},\ \underline{e_s, f_s, g_s, s},\ e,\ f,\ g,\ \underline{\mathbf{S}},\ e_t, \underline{h_t},\ i_t, \underline{t, e},\ h, \underline{i,\ \mathbf{S}},\ f_u, h_u, j_u, \underline{\mathbf{u}, f, h, j,\ \mathbf{S}},\ g_v, \underline{i_v, j_v, v, g},\ i,\ j,\ -$

$-\ \mathbf{S},\ e_s, e_t, f_s, f_u, g_s, g_v, h_t, h_u, i_t, i_v, j_u, j_v\ \rangle$
$-\ \underline{\mathbf{S}},\ \underline{s'_1, s'_2, s'_3, t'_1, t'_2, t'_3, u'_1, u'_2, u'_3, v'_1, v'_2, v'_3}\ \rangle$

Figure 3.3: Example for the reduction of Minimum Vertex Cover in Cubic Graphs to $(1,2)$-Exemplar Edit Distance. Above: a cubic graph G with an optimal vertex cover $\{s,t,v\}$ and the corresponding independent set $\{u\}$. Below: the sequences G_1 and G_2 created from G, we use a common symbol S for all separators. An optimal exemplarization of G_2 is underlined, and matched elements in this exemplarization are in bold font.

$$G_2[tail] = V'$$

This completes the construction.

Lemma 3.8. *G has a vertex cover of size at most k iff G_2 has an exemplar subsequence G'_2 with edit distance at most $m + 6n + k$ from G_1.*

Proof. We first prove the direct implication. Let X be a vertex cover of G with $|X| \leq k$. Create G'_2 as follows. For each edge $e = \{u, v\}$, at least one vertex, say u, is in X. Remove e_u and retain e_v in the edge gadget $G_2[e]$, and correspondingly retain e_u in the vertex gadget $G_2[u]$ and remove e_v in the vertex gadget $G_2[v]$, then remove e in $G_2[u]$ and retain e in $G_2[v]$. We claim that the edit distance from G_1 to G'_2 is at most $m + 6n + k$.

It suffices to show that the Hamming distance of G_1 and G'_2 is at most $m+6n+k$ since, for the edit distance that we consider, the cost of a substitution is 1. Observe that in both G_1 and G'_2, each edge gadget has length 1, and each vertex gadget has length 4. Thus all gadgets are aligned and all separators are matched. The Hamming distance for each edge gadget is 1, so the total Hamming distance over all edge gadgets is m. The Hamming distance for each vertex gadget is at most 4. Moreover, for each vertex $v \notin X$ (v incident to edges e, f, g), since the markers e_v, f_v, g_v are removed (and the markers e, f, g are retained) in the vertex gadget, the marker v is matched, which reduces the Hamming distance by 1. Thus the total Hamming distance over all vertex gadgets is at most $4n - (n - |X|) = 3n + |X|$. Finally, since the Hamming distance for the tail gadget is $3n$, the overall Hamming distance between G_1 and G'_2 is at most $m + 6n + |X| \leq m + 6n + k$.

We next prove the reverse implication. Let G'_2 be an exemplar subsequence of G_2 with edit distance at most $m + 6n + k$ from G_1. Compute an alignment of G_1 and G'_2 corresponding to the edit distance, then obtain the following three sets $X_E(G'_2)$, $X_V(G'_2)$, and $X(G'_2)$:

- The set $X_E(G'_2) \subseteq E$ contains every edge $e = \{u, v\}$ such that either $G'_2[e]$ contains both e_u and e_v, or $G'_1[e]$ has an adjacent separator marker which is unmatched.
- The set $X_V(G'_2) \subseteq V$ contains every vertex v (v incident to edges e, f, g) such that either $G'_2[v]$ contains one of $\{e_v, f_v, g_v\}$, or $G'_1[v]$ has an adjacent separator marker (to its left) which is unmatched.
- The set $X(G'_2) \subseteq V$ is the union of $X_V(G'_2)$ and a set composed by arbitrarily choosing one vertex from each edge in $X_E(G'_2)$ (thus $|X(G'_2)| \leq |X_V(G'_2)| + |X_E(G'_2)|$).

We first show that the edit distance from G_1 to G'_2 is at least $m + 6n + |X(G'_2)|$. If a long separator (with $m + 7n$ markers) is completely unmatched, then the edit distance is at least $m + 7n \geq m + 6n + |X(G'_2)|$. Hence we can assume that there is at least one matched marker in each long separator. Consequently, the markers e, e_u, e_v for all $e \in E$ and v'_1, v'_2, v'_3 for all $v \in V$ are unmatched.

Consider an edge $e = \{u, v\} \in E$. If $e \notin X_E(G'_2)$, then the edit distance for $G_1[e]$ is at least 1 since the marker e is unmatched. If $e \in X_E(G'_2)$, then consider the factor of $G_1[e]$ containing the marker e and the at most two separator markers adjacent to it (for the first edge gadget, there is only one separator marker adjacent to e, to its right). The edit distance for this factor is at least 2: the marker e is unmatched, and moreover either an adjacent separator marker is unmatched or an insertion is required. The total edit distance over all edge gadgets is at least $m + |X_E(G'_2)|$.

Consider a vertex $v \in V$ incident to three edges e, f, g. If $v \notin X_V(G'_2)$, then the edit distance for $G_1[v]$ is at least 3 since the markers v'_1, v'_2, v'_3 are unmatched. If $v \in X_V(G'_2)$, then consider the factor of G_1 containing $G_1[v]$ and the separator to its left. The edit distance for this factor is at least 4: the markers v'_1, v'_2, v'_3 are unmatched, and moreover at least one insertion is required unless either the marker v or the separator marker to its left is unmatched. The total edit distance over the vertex gadgets is at least $3n + |X_V(G'_2)|$.

Finally, the edit distance over the tail gadget is equal to the length of $G_1[tail]$, which is $3n$. Hence the overall edit distance is at least

$$m + |X_E(G'_2)| + 3n + |X_V(G'_2)| + 3n \geq m + 6n + |X(G'_2)|.$$

Since the edit distance from G_1 to G'_2 is at most $m + 6n + k$, it follows that

$$|X(G'_2)| \leq k.$$

To complete the proof, we show that $X(G'_2)$ is a vertex cover of G. Consider any edge $e = \{u, v\}$. If $e \in X_E(G'_2)$, then, by our choice of $X(G'_2)$, either $u \in X(G'_2)$ or $v \in X(G'_2)$. Otherwise, if $e \notin X_E(G'_2)$, then in the edge gadget $G_2[e] = \langle e_u, e_v \rangle$, at least one marker is removed to obtain $G'_2[e]$. Assume that e_u is removed: then the second copy, in $G_2[u]$, is retained, and $u \in X_V(G'_2) \subseteq X(G'_2)$. Likewise if e_v is removed, then $v \in X(G'_2)$. In summary, $X(G'_2)$ contains a vertex from every edge in E, hence it is a vertex cover of G. □

The problem Minimum Vertex Cover in Cubic Graphs is APX-hard; see e.g. [4]. For a cubic graph G of n vertices and m edges, where $3n = 2m$, the minimum size k^* of a vertex cover is $\Theta(n)$. By Lemma 3.8, the exemplar edit distance of the two sequences G_1 and G_2 in the reduced instance is also $\Theta(n)$. Thus by the standard technique of L-reduction (see Page 8), it follows that $(1, 2)$-Exemplar Edit Distance, when the cost of a substitution is 1 and the cost of an insertion or a

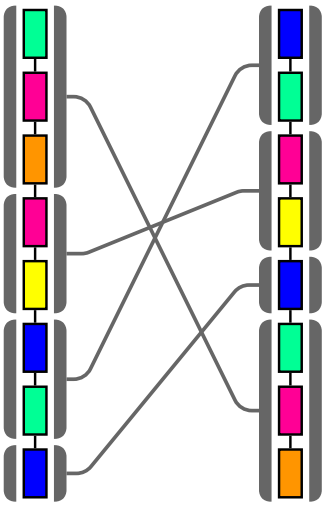
deletion is at least 1, is APX-hard too. Then the APX-hardness of $(1, 2)$-Exemplar Levenshtein Distance and the APX-hardness of $(1, 2)$-Exemplar Hamming Distance follow as special cases. Moreover, since the lengths of the two sequences G_1 and G_2 in the reduced instance are both $\Theta(m + n)$ as well, it follows that the complementary maximization problem $(1, 2)$-Exemplar Hamming Similarity is also APX-hard, if we define the *Hamming similarity* of two sequences of the same length ℓ as ℓ minus their Hamming distance.

Conclusion

The problem $(1, 2)$-Exemplar Distance has now been shown to be APX-hard for a wide variety of distance measures, including breakpoints, conserved intervals, common intervals, MAD, SAD, signed reversals and DCJs, Levenshtein distance, Hamming distance, etc. On the other hand, it seems difficult to improve the constant lower bound in any one of these APX-hardness results into a lower bound that grows with the input size similar to the logarithmic lower bound for Minimum Set Cover. This would indicate that constant-ratio approximation algorithms should exist: we find it most intriguing that no such approximation is known for any one of these measures. Also, no positive FPT result is known for exemplar distance problems. A possible explanation is that, through the exemplarization process, it is hardly possible to keep track of any structure in the gene sequences.

4

Minimum Common String Partition



The NP-hard Minimum Common String Partition problem has as input two strings x and y and asks whether x and y can each be partitioned into at most k substrings, called blocks, such that both partitions use exactly the same blocks in a different order.

In this chapter, we present a fixed-parameter algorithm for k-MCSP using only parameter k.

The results in this chapter have been obtained through a joint work with Christian Komusiewicz. They have been presented at the *Journées GTGC* workshop (2012, Lille), and have been submitted to the 25th ACM-SIAM Symposium on Discrete Algorithms (SODA 2014, Portland OR, USA)

Introduction

The problem we study in this chapter lies in the context of comparative genomics on strings, where the objective is to determine how many operations of a certain kind are necessary to transform one genome into another. The input consists of a pair of strings x and y. The operation to transfer x into y is, informally, to cut x at a minimum number of *breakpoints* and to reorder and concatenate the resulting factors to obtain exactly y. This transformation is formalized by the notion of *common string partition* (CSP): a partition $\mathcal{P}$ of two strings x and y into *blocks* $x_1x_2\cdots x_k$ and $y_1y_2\cdots y_k$ is a common string partition if there is a bijection M between $\{x_i \mid 1 \leq i \leq k\}$ and $\{y_i \mid 1 \leq i \leq k\}$ such that x_i is the same string as $M(x_i)$ for all $1 \leq i \leq k$ (see Figure 4.1 for an example). Herein, k is called the *size* of the common string partition $\mathcal{P}$. We study the problem of finding a minimum-size CSP:

Problem	Minimum Common String Partition (MCSP)
Input	Two strings x and y of length n, and an integer k
Question	Is there a common string partition (CSP) $\mathcal{P}$ of size at most k of x and y?

Another point of view for the problem is to see it as a matching problem. Given the two sequences x and y, we aim at producing a bijection between markers of both strings such that only markers representing the same letter are matched, and that a maximum number of pairs of consecutive markers are matched to consecutive markers. Using MCSP as a pretreatment before applying further algorithm, it gives a way that is optimal (in some sense) to transform strings into duplication-free permutations. Note that due to the fact that (1) each marker must be covered by exactly one block and (2) the matched blocks must correspond precisely to the same string, the strings x and y must use exactly the same multi-set of letters in order to have at least one CSP (the strings must be *balanced*). Due to this strong constraint, MCSP is mostly of theoretical interest, and any algorithm require adaptations before being applied to raw data.

MCSP was introduced independently by Chen et al. [45] and Swenson et al. [120] (who call the problem Sequence Cover). MCSP is NP-hard and APX-hard even when each letter occurs at most twice [77]. Damaschke [61] initiated the study of MCSP in the context of parameterized algorithmics by showing that MCSP is fixed-parameter tractable with respect to the combined parameter "partition size k and repetition number r of the input strings". Subsequently, Jiang et al. [91] showed that MCSP can be solved in $(d!)^k \cdot \text{poly}(n)$ time, where d is the maximum number of occurrences of any letter in either input string. MCSP can be solved in $2^n \cdot \text{poly}(n)$ time [72]. A greedy heuristic for MCSP was presented by Shapira and Storer [119]. In this chapter, we answer an open question [61, 72, 91] by showing that MCSP is fixed-parameter tractable when parameterized only by k, that is, we present an algorithm with running time $f(k) \cdot \text{poly}(n)$.

Basic Notation. A *marker* is an occurrence of a letter at a specific position in a string; we denote the marker at position i in a string x by $x[i]$. For all i, $1 \leq i < n$, the markers $x[i]$ and $x[i+1]$ are called *consecutive*. An *adjacency* is a pair of

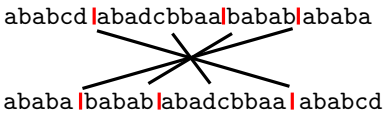


Figure 4.1: An instance of MCSP with a common string partition of size four.

consecutive markers. A *factor* is a set of consecutive markers, that is, a factor is a set $\{x[i], x[i+1], \ldots, x[j]\}$ for some $i \leq j$. We write $[a, b]$ to denote the factor whose first marker is a and whose last marker is b. The *length* $\|I\|$ of a factor I is the number of markers it contains. Given two markers a and b in the same string x, we write $\overline{ab}$ to denote the signed distance between a and b, that is, $\overline{ab} = \|[a, b]\| - 1$ if a appears before b in x, and $\overline{ab} = -\|[b, a]\| + 1$, otherwise. Given two factors s and t, we write $s \equiv t$ if they represent the same string of letters (if they have the same contents) and $s = t$ if they are the same factor, that is, they start and end at the same position in the same string. Similarly, for two markers a and b we write $a \equiv b$ if their letters are the same, and $a = b$ if the markers are identical. We say that a string s has *period* π if $s = \rho\pi^i\tau$, where $i \geq 1$, ρ is a (possibly empty) suffix of π, and τ is a (possibly empty) prefix of π. Without more precision, *the period* of a string s refers to any of its shortest periods. We define *offset* operators $\triangleright$ and $\triangleleft$: For each marker e and integer d, $e' = e \triangleright d$ is the marker such that $\overline{ee'} = d$, and $e \triangleleft d = e \triangleright (-d)$.

4.1 Fundamental Definitions and Algorithm Outline

In this section, we first present the most fundamental definitions used by our algorithm. We then draw a general outline of its main method. A simplified example for $k = 3$ is given in Figure 4.2.

4.1.1 Definitions

Let $\mathcal{P} = \{x_1 x_2 \ldots x_\ell; y_1 y_2 \ldots y_\ell; M\}$ be a CSP of strings x and y. A *breakpoint* of $\mathcal{P}$ is an adjacency in x (or y) that contains the last marker of some block x_i (y_i) and the first marker of the next block x_{i+1} (y_{i+1}). We say that $\mathcal{P}$ *matches two blocks* x_i and y_j if $M(x_i) = y_j$. Furthermore, we say that $\mathcal{P}$ *matches two markers* a and b if a and b are at the same position in matched blocks. By the definition of a CSP, this implies $a \equiv b$.

The algorithm works on subdivisions of both strings into shorter parts. These subdivisions are formalized as follows.

Definition 4.1 (Splitting)**.** *A* splitting *of a string (or a factor) z is a list of factors $[a_1, b_1]$, $[a_2, b_2]$, $\ldots$, $[a_m, b_m]$, each of length at least two, called* pieces *such that $a_1 = z[1]$, $a_{j+1} = b_j$ for all $j < m$, and $b_m = z[\|z\|]$.*

Informally, a splitting is a partition of the adjacencies of a string (or a factor) such that each part contains only consecutive adjacencies.

The strategy of the algorithm is to infer more and more information about an optimal CSP. To put it another way, it makes more and more restrictions on the CSP that it tries to construct. To this end, the algorithm will annotate splittings

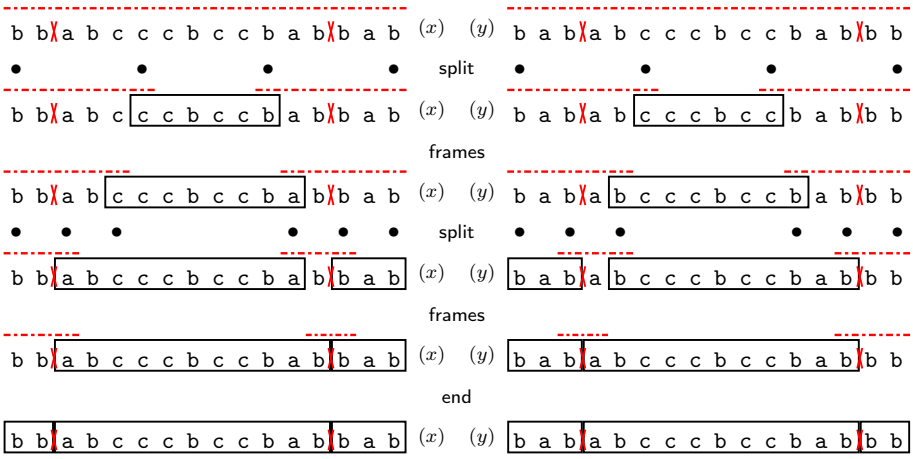


Figure 4.2: Illustration of the algorithm on two strings (x and y) where the optimal solution uses $k = 3$ blocks (the two corresponding breakpoints in each string are marked with red crosses). In *split* steps, the strings are cut at specific locations (bullets), and pieces without breakpoints are marked as solid (black boxes): they are never split again, as opposed to fragile pieces (dashed red lines). In *frames* steps, fragile pieces are reduced in order to fit closer to their inner breakpoints. NB: the example has been simplified to be more readable. A real run would create more pieces, and wider margins in fragile pieces.

as follows: a piece is called *fragile* if it contains at least one breakpoint, and *solid* if it contains no breakpoint. To simplify the representation, the algorithm sometimes *merges* consecutive pieces $[a_i, b_i]$ and $[a_{i+1}, b_{i+1}]$ (where $b_i = a_{i+1}$) into one, that is, it removes $[a_i, b_i]$ and $[a_{i+1}, b_{i+1}]$ from some splitting and adds the factor $[a_i, b_{i+1}]$ to this splitting.

To further restrict the CSP, the algorithm finds pairs of solid pieces in x and y that are contained in blocks that are matched by the CSP. Accordingly, a pair of solid pieces s in x and t in y is called *matched* in a CSP $\mathcal{P}$ if s is contained in a block of $\mathcal{P}$ that is matched to a block that contains t. Note that matched solid pieces may correspond to different parts of their blocks. For example, one piece may contain the first marker but not the last marker of its block in x and it can be matched to a solid piece that contains the last but not the first marker of its block in y. Hence, when looking at the two blocks containing the pieces, there can be a "shift" between the matched pieces. We formalize this as follows, see Fig. 4.3 (left) for an example.

Definition 4.2 (Alignment)**.** *Let* $[a, b]$ *be a piece of a splitting of* x *and* $[c, d]$ *be a piece of a splitting of* y*. The* alignment *of* $[a, b]$ *and* $[c, d]$ *of* shift δ *is the pair of* reference markers a *and* $c \triangleright \delta$*, where*

- $(-\overline{ab}) \leq \delta \leq \overline{cd}$*,*
- $[a, b] \equiv [c \triangleright \delta, c \triangleright (\overline{ab} + \delta)]$ *and* $[c, d] \equiv [a \triangleright (-\delta), a \triangleright (\overline{cd} - \delta)]$*.*

Hence, an alignment fixes how the factor $[a, b]$ is shifted with respect to $[c, d]$ in the matched blocks that contain the factors. That is, if $[a, b]$ starts at position j in its block, then $[c, d]$ starts at position $j - \delta$. For matched solid pieces, an alignment thus fixes which markers are matched to each other by the CSP. In particular, the marker a is matched to $c \triangleright \delta$ and c is matched to $a \triangleleft \delta$. Note that the maximum and minimum values allowed for δ ensure that there is at least one marker in $[a, b]$ that is matched to a marker in $[c, d]$ by a CSP corresponding to this alignment. The

algorithm will only consider such alignments between matched solid pieces. The second condition verifies that all pairs of matched markers indeed correspond to the same letter. Clearly, this restriction is fulfilled by every CSP that does not put breakpoints in the solid pieces $[a, b]$ and $[c, d]$. A pair of matched solid pieces is called *fixed* if it is associated with an alignment (equivalently, with a pair of reference markers) and *repetitive* otherwise (the reason for choosing this term will be given below). For a fixed solid piece s, we use s^* as shorthand for the uniquely determined reference marker of the alignment of s which is in the same string as s.

These restrictions on a possible CSP are summarized in the notion of constraints, defined as follows, see Fig. 4.3 (right) for an example.

Definition 4.3 (Constraint)**.** *A* constraint $\mathcal{C}$ *is a tuple* (S, F, M, R_S) *such that:*
- S *is a set of solid pieces. Let* S_x *(*S_y*) denote the pieces of* S *from* x *(*y*).*
- F *is a set of fragile pieces. Let* F_x *(*F_y*) denote the pieces of* F *from* x *(*y*).*
- *The pieces of* $S_x \cup F_x$ *(*$S_y \cup F_y$*) form a splitting of* x *(*y*) in which solid and fragile pieces alternate.*
- $M : S_x \to S_y$ *is a* matching*, that is, a bijection between* S_x *and* S_y*. As shorthand, we write* $s' = M(s)$ *if* $s \in S_x$ *and* $s' = M^{-1}(s)$ *if* $s \in S_y$*.*
- R_S *is a set of alignments that contains for each matched pair of solid pieces at most one alignment.*

Our algorithm will search for CSPs that satisfy such constraints.

Definition 4.4 (Satisfy)**.** *A CSP* $\mathcal{P}$ satisfies *the constraint* $\mathcal{C} = (S, F, M, R_S)$ *if:*

1. *All breakpoints of* $\mathcal{P}$ *are contained in fragile pieces.*
2. *Each fragile piece contains at least one breakpoint from* $\mathcal{P}$*.*
3. *Matched solid pieces are contained in matched blocks in* $\mathcal{P}$*.*
4. *If* s *is a fixed solid piece, then markers* s^* *and* s'^* *are matched in* $\mathcal{P}$*.*
5. *If* s *is a repetitive solid piece, then* s*,* s' *and the blocks containing them in* $\mathcal{P}$ *all have the same shortest period.*

Equivalent formulations of Conditions 1 and 2 are that (1') all solid pieces are contained in blocks of $\mathcal{P}$, and (2') different solid pieces in the same string are in different blocks. Given a CSP $\mathcal{P}$ that satisfies a constraint $\mathcal{C}$, we call a block *short*, or *undiscovered by* $\mathcal{C}$, if it does not contain a solid piece (equivalently, if it is contained in a fragile piece). The other blocks are called *long* or *discovered by* $\mathcal{C}$.

Finally, we introduce the following notion that concerns reference markers and fixed solid pieces.

Definition 4.5 (Equidistant)**.** *Let* s *and* s' *be fixed matched solid pieces in* x *and* y*. Two markers* a *in* x *and* b *in* y *are* equidistant from s *if* $\overline{s^* a} = \overline{s'^* b}$*. Similarly, two factors* $[a, b]$ *in* x *and* $[c, d]$ *in* y *are* equidistant from s *if* a *and* c *are equidistant from* s *and* b *and* d *are equidistant from* s*.*

We will use this notation to talk about the "local environment" of the reference markers in both strings. In particular, with this notation we can identify (sets of) markers that are matched to each other if they are both in the same block as the reference markers.

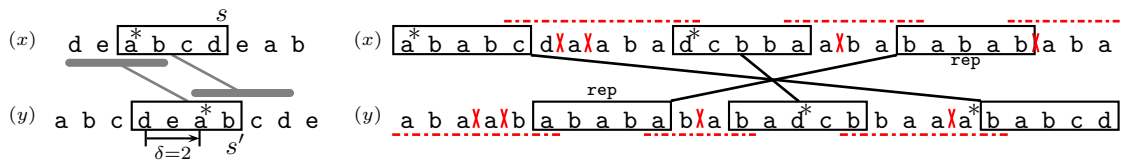


Figure 4.3: Left: Example of alignment between two pieces s and s'. Reference markers are marked with a star, the shift is 2. Intervals having the same content as the pieces according to this alignment are marked in gray. Note that there also exists an alignment of shift -3, where the reference marker in y is the first occurrence of a. Right: A constraint with three pairs of solid pieces illustrated by boxes. Two of these pairs are fixed and one is repetitive (`rep`). Matched solid pieces are linked with edges. The fragile pieces (red and dashed lines) contain the breakpoints (red crosses) of a size-5 CSP satisfying the constraint.

4.1.2 An Outline of the Algorithm

We now give a high-level description of the main idea of the algorithm; the pseudo-code of the main algorithm loop is shown in Algorithm 4.1. [1] For the discussion, assume that the instance is a yes-instance, that is, there exists a CSP $\mathcal{P}$ of size k. Since we can check in polynomial time the size and correctness of any CSP before outputting it, we can safely assume that the algorithm gives no output for no-instances; hence the focus on yes-instances. The algorithm gradually extends a constraint that is satisfied by a solution $\mathcal{P}$ and outputs $\mathcal{P}$ eventually. Initially, the constraint consists solely of two fragile pieces, one containing all of x and one all of y. We assume that the input strings are not identical. Hence, every CSP has at least one breakpoint and the initial constraint is thus satisfied by every size-k CSP.

The algorithm now aims at discovering the blocks of $\mathcal{P}$ successively, from the longest to the shortest. Recall that a block is called discovered by a constraint $\mathcal{C}$ if there is a solid piece in $\mathcal{C}$ that is contained in this block. To execute the strategy of finding shorter and shorter blocks, the algorithm needs some knowledge about the approximate (by a factor of 2) length of the longest undiscovered block in $\mathcal{P}$. To this end, the algorithm keeps and updates an integer variable β which has the following central property: Whenever there is a size-k CSP satisfying the current constraint, then there is in particular one size-k CSP $\mathcal{P}$ such that

1. the longest undiscovered block of $\mathcal{P}$ has length ℓ with $\beta \leq \ell < 2\beta$, and
2. β is minimum among all integers satisfying Property 1.

Accordingly, we call a block *β-critical* if it has length ℓ with $\beta \leq \ell < 2\beta$. To obtain β, we consider all subsets Π' of the set Π containing all powers of 2 that are smaller than n. One of these sets will contain the "correct" approximate block lengths. The central strategy is: Set β to be the largest value in Π'. Discover all β-critical blocks. Then, there is a satisfying CSP such that all undiscovered blocks are shorter than the current β. Thus update β by taking the next largest value from Π'. Then, again discover all β-critical blocks, update β again and so on.

1. Parts of this algorithm, in particular the `split` procedure follow somewhat the approach of Damaschke [61]

Algorithm 4.1 FPT algorithm for $\texttt{MCSP}(x, y, k)$ (main loop).

1: $\Pi \leftarrow \{i \in \mathbb{N} \mid i < n \wedge \exists j \in \mathbb{N} : 2^j = i\}$
2: $\mathcal{C} \leftarrow \{S \leftarrow \emptyset, F \leftarrow \{[x[1], x[n]], [y[1], y[n]]\}, M \leftarrow \emptyset, R_S \leftarrow \emptyset\}$
// initially only two fragile pieces
3: **for each** $\Pi' \subseteq \Pi$ with $\max \Pi' \geq \lceil n/2k \rceil \wedge |\Pi'| \leq k$ **:**
4: $\beta \leftarrow \max \Pi'$; $\Pi' \leftarrow \Pi' \cup \{0\} - \{\beta\}$ *// 2-approx. length of longest block*
5: **repeat until** $\beta < 4$ **:**
6: `split` *// discover blocks of length at least* β
7: $\beta \leftarrow \max \Pi'$; $\Pi' \leftarrow \Pi' - \{\beta\}$ *// update length of longest undiscovered blocks*
8: `frames` *// reduce length of fragile pieces*
9: branch into all cases to set breakpoints within fragile pieces
10: **if** the resulting string partition $\mathcal{P}$ is a size-k CSP **: output** $\mathcal{P}$

First, note that there is at least one block of length at least $\lceil n/k \rceil$ since $\mathcal{P}$ has size k, so $\max \Pi' \geq \lceil n/2k \rceil$. Furthermore, for any CSP of size k, $|\Pi'| \leq k$. Hence, the outer algorithm loop of Algorithm 4.1 is traversed once for the correct Π'. Note furthermore, that the number of subsets of Π is $O(2^{\log n}) = O(n)$. Hence, there are $O(n)$ traversals of the outer loop of the main method.

Consider now the traversal for the correct set Π'. The inner loop of the algorithm consists of two main steps. In the first step, called `split`, the algorithm discovers the β-critical blocks. More precisely, it refines $\mathcal{C}$ by breaking fragile pieces into shorter pieces (of length $\lceil \beta/3 \rceil$) and identifying those that are contained in β-critical blocks. It then produces a matching and, if this is possible without considering too many options, aligns these blocks.

To be efficient `split` requires that the input fragile pieces are short enough compared to β and k. Initially, this is not a problem, since the fragile pieces have length n, and $\beta \geq n/2k$. After `split`, however, we update β. Hence, between two calls to `split` the fragile pieces have to be reduced in order to fit the undiscovered blocks more "tightly". This is the objective of `frames`, which uses a set of rules to identify smaller factors containing all breakpoints of $\mathcal{P}$. It thus shrinks the fragile pieces of $\mathcal{C}$ so that they are sufficiently small for the next call to `split`.

The algorithm now continues with this process for smaller and smaller values of β. It stops in case $\beta < 4$, since it can then locate all breakpoints by applying a brute-force branching. Note that in order to ensure that there is always a $\beta < 4$, we add the value 0 to set Π' in Line 4 of the main method.

In the remainder of this work, we give the details for the procedures `split` and `frames`. In Section 4.2, we describe the `split` procedure, and show its correctness. We also show, using several properties of `frames` as a black box, our main result. Then, in Sections 4.3 and 4.4, we fill in the blanks by proving the properties of `frames`.

The algorithm is a branching algorithm that extends the constraint $\mathcal{C}$ in each branch. In order to simplify the pseudo-code somewhat, we describe the algorithm in such a way that the variables $\mathcal{C}$ and β are global variables. After a branching statement in the pseudo-code, the algorithm continues in each branch with the following line of the pseudo-code. If a branch is known to be unsuccessful, then the algorithm returns immediately to the branching statement that created this branch (or to the branching statement above, if the current branch is the last branch of that statement). We denote this by the "abort branch" command; all modifications within this branch are undone.

Algorithm 4.2 Procedure `split`. Global variables: $\mathcal{C} = (S, F, M, R_S)$ and β.

1: $N \leftarrow \emptyset$ // *the set of new pieces*
2: **for each** fragile piece $f \in F$ **:**
3: $F \leftarrow F - \{f\}$ // *old fragile pieces are removed*
4: $N \leftarrow N \cup$ "$\lceil \beta/3 \rceil$-splitting of f" // *update set of new pieces*
5: **for each** $p \in N$ **:** // *make p either fragile or solid*
6: branch into the case that either $S \leftarrow S \cup \{p\}$ or $F \leftarrow F \cup \{p\}$
7: **while** $\exists$ consecutive pieces p_1, p_2 s.t. $\{p_1, p_2\} \subseteq S$ (or $\{p_1, p_2\} \subseteq F$) **:**
8: $p \leftarrow$ "merged factor of p_1 and p_2"
9: $S \leftarrow (S \cup p) - \{p_1, p_2\}$ (or $F \leftarrow (F \cup p) - \{p_1, p_2\}$)
10: **if** $|S_x| \neq |S_y|$ **: abort branch** // *no bijection of solid pieces exists*
11: **if** $|F_x| \geq k$ or $|F_y| \geq k$ **: abort branch** // *too many fragile pieces in x (or y)*
12: **while** $\exists$ unmatched solid piece $s \in S_x$ **:**
13: branch into the case that $M(s) \leftarrow t$ for each unmatched solid piece t in S_y
14: **for each** new pair (s, t) of matched solid pieces **:**
15: $i \leftarrow$ "number of alignments with shift δ s.t. $|\delta| \leq \lceil \beta/3 \rceil$"
16: **if** $i \leq 6$ **:** for each alignment branch into the case to add this alignment to R_S
17: **else:** branch into the cases to: // *s and s' are periodic*
– align s and s' such that $\mathrm{l}_{\mathrm{break}}(s)$ and $\mathrm{l}_{\mathrm{break}}(s')$ are equidistant from s
– align s and s' such that $\mathrm{r}_{\mathrm{break}}(s)$ and $\mathrm{r}_{\mathrm{break}}(s')$ are equidistant from s
– do not align s and s'

4.2 Splitting of Fragile Pieces

In this section, we describe the procedure `split` and show its correctness. The pseudo-code of `split` is shown in Algorithm 4.2. At the beginning of `split` the constraint contains a set of discovered blocks. Assume that all blocks of length at least 2β are discovered by this constraint. The aim of `split` now is to perform a branching into several cases such that in at least one of the created branches the constraint $\mathcal{C}$ now additionally contains all β-critical blocks. Hence, in this branch all blocks of length at least β are discovered. Procedure `split` starts by replacing each former fragile piece by a splitting where all new pieces have length $\lceil \beta/3 \rceil$ except for the rightmost new piece of each such splitting which can be shorter. We call such a splitting a $\lceil \beta/3 \rceil$-*splitting*. It then considers all branches where each piece is either fragile or solid. In order to maintain the alternating condition, consecutive solid (resp. fragile) pieces are merged into one solid (fragile) piece, Lines 7–9.

Next, `split` extends the matching and the set of alignments of the constraint. All possible matchings are considered in separate branches (Lines 12–13). Then, `split` performs an exhaustive branching over all alignments for a given pair of solid pieces, but only if there are very few of them (Line 16). If there are too many (Line 17), then it can be seen that the pieces are periodic with a short period length. Thus, the blocks containing them might be periodic as well. If the blocks are not periodic, then there are at most two alignments that the algorithm needs to consider: informally, the period in the blocks can be "broken" either to the left or to the right of the pieces. To specify these two possibilities more clearly, we introduce the following notation. Let $s = [a, b]$ be a factor in a string x such that s has period π. Then, we denote by $\mathrm{l}_{\mathrm{break}}(s)$ the rightmost marker in x such that $[\mathrm{l}_{\mathrm{break}}(s), b]$ does not have period π.

Similarly, let $\mathrm{r}_{\mathrm{break}}(s)$ be the leftmost marker in x such that $[a, \mathrm{r}_{\mathrm{break}}(s)]$ does not have period π. If the blocks are periodic, there may be too many possible alignments, and the alignment between the pieces will be fixed at a later point (when β becomes smaller than the period). However, the algorithm will use the "knowledge" that the blocks are periodic in the `frames` procedure.

We now show that `split` is correct if the input constraint can be satisfied and that it discovers all β-critical blocks.

Lemma 4.1. *Let $\mathcal{C}$ be the constraint at the beginning of* `split`*, and let $\mathcal{P}$ be a size-k CSP satisfying $\mathcal{C}$ such that all blocks of length at least 2β of $\mathcal{P}$ are discovered by $\mathcal{C}$. Then,* `split` *creates at least one branch whose constraint $\mathcal{C}$*

- *is satisfied by $\mathcal{P}$, and*
- *all blocks of length at least β are discovered by $\mathcal{C}$*

Proof. Let $B = \{(x^1, y^1), \ldots, (x^\ell, y^\ell)\}$ be the uniquely defined set of matched pairs of undiscovered blocks in $\mathcal{P}$ that are β-critical.

Consider the following branching for Lines 5–6 for each piece $p \in N$: If p is contained in some block x^i or y^i of B, then branch into the case that p is added to S. Otherwise, branch into the case that p is added to F (note that we may add in F some pieces that do not contain any breakpoint, but are contained in blocks not in B).

Now consider the constraint obtained for the above branching after the merging operations performed in Lines 7–9. We show that $\mathcal{P}$ satisfies Conditions 1 and 2 of this constraint. First, consider a breakpoint in $\mathcal{P}$. This breakpoint is contained in some fragile piece f of the input constraint since $\mathcal{P}$ satisfies this input constraint. Hence, it is contained in some new piece p of the splitting of this fragile piece. Clearly, the piece p is added to F in the considered branching. Moreover, in case Lines 7–9 merge fragile pieces, the resulting piece is also fragile, hence p remains in a fragile piece. Consequently, all breakpoints of $\mathcal{P}$ are in fragile pieces of F, and thus Condition 1 is satisfied by $\mathcal{P}$.

Now consider a fragile piece $f \in F$ after Lines 7–9 of the algorithm. We show that f contains at least one breakpoint. Note that f is obtained after a (possibly empty) series of merging operations. After the merging, f is between two solid pieces. If f is also a fragile piece in the input constraint, then f contains a breakpoint since $\mathcal{P}$ satisfies the input constraint. Otherwise, f is contained in a fragile piece of the input constraint, and at least one of its neighbor pieces is a new solid piece s. Since f (or all the smaller pieces that were merged to f) are added to F by the branching, they are not contained in the block that contains s. Hence, f contains the breakpoint between the first (or last) marker of the block containing the new solid piece and its predecessor (or successor). Thus, Condition 2 is also satisfied by $\mathcal{P}$.

Note that the above also implies that, for each x^i of B, there is exactly one new solid piece that is contained in x^i. Similarly, for each y^i of B, there is exactly one new solid piece that is contained in y^i. Note that in this branching, $|S_x| = |S_y|$ and furthermore, since $\mathcal{P}$ has size k, $|F_x| < k$ and $|F_y| < k$. Hence, the algorithm does not abort in Lines 10 and 11. We now consider the branching in which for each pair (x^i, y^i), the two corresponding solid pieces are matched to each other. Clearly, this branching fulfills Condition 3: the condition holds obviously for all pieces contained in blocks of B. Furthermore, it holds for all old solid pieces since for these, the matching M has not changed. Note that the function M also remains

a bijection: it is changed only for unmatched solid pieces, and the number of new solid pieces in x and y is equal.

It remains to show that there is a branching in which Conditions 4 and 5 also hold. Consider a pair of matched solid pieces s and s', and the blocks x^i, y^i containing them. We use the following technical claim in order to clarify the discussion; it will be proven afterwards.

> *Fact.* If there are more than six alignments of s and s' whose shift have an absolute value of at most $\lceil\beta/3\rceil$, then
> i. s and s' are periodic with the same shortest period π (with $\|\pi\| \leq \lceil\beta/3\rceil/2$);
> ii. if the blocks x^i and y^i do not have period π, then in $\mathcal{P}$ either $\mathrm{l}_{\mathrm{break}}(s)$ is matched to $\mathrm{l}_{\mathrm{break}}(s')$, or $\mathrm{r}_{\mathrm{break}}(s)$ is matched to $\mathrm{r}_{\mathrm{break}}(s')$ (or both).

Let a be the leftmost marker of s and $\tilde{a}$ be the marker matched to a in $\mathcal{P}$. Then $\{a, \tilde{a}\}$ is an alignment for (s, s') whose shift has an absolute value less than $\lceil\beta/3\rceil$: there are at most $\lceil\beta/3\rceil - 1$ markers preceding either s or s' that can belong to the same block since the pieces of the $\lceil\beta/3\rceil$-splitting preceding s and s' are fragile and thus not contained in the same blocks. If the condition of Line 16 is satisfied, then there is one branch where alignment $\{a, \tilde{a}\}$ is added to R_S. Otherwise, by the fact above, the following cases are possible. Either $\{a, \tilde{a}\}$ is one of the alignments where $\mathrm{l}_{\mathrm{break}}(s)$ is matched to $\mathrm{l}_{\mathrm{break}}(s')$ or $\mathrm{r}_{\mathrm{break}}(s)$ is matched to $\mathrm{r}_{\mathrm{break}}(s')$, in which cases $\{a, \tilde{a}\}$ is added to R_S in one of the branches. Otherwise, (s, s') is not fixed, and s and s' are contained in blocks having the same shortest periods.

Altogether this shows the first claim of the lemma. The second claim can be seen as follows. The blocks of length $\ell \geq 2\beta$ are already discovered, and the corresponding solid pieces remain in the constraint. It thus remains to consider the β-critical blocks. We show that for each x^i there is at least one piece that is contained in x^i. Consider the marker a at position $\lceil\beta/3\rceil$ in x^i and a piece s of the $\lceil\beta/3\rceil$-splitting that contains this marker. Then s contains only markers from x^i since s has length at most $\lceil\beta/3\rceil$ and x^i has length at least $\beta \geq 2\lceil\beta/3\rceil$ (for $\beta \geq 4$). Afterwards, s is only merged with other pieces that are contained in x^i (recall that in the considered branching there is a fragile piece between all solid pieces from different blocks). Hence, the second claim of the lemma also holds.

It remains to show the correctness of the claimed fact. We first need to prove the following claim. Define the $\lceil\beta/3\rceil$-*middle* of a factor $[u, v]$ as the length-$\lceil\beta/3\rceil$ factor centered in $[u, v]$ (formally, the factor $[\hat{u}, \hat{v}]$ with $\hat{u} = u \triangleright \lfloor(\overline{uv} - \lceil\beta/3\rceil)/2\rfloor$ and $\hat{v} = v \triangleleft \lceil(\overline{uv} - \lceil\beta/3\rceil)/2\rceil$). Then s contains the $\lceil\beta/3\rceil$-middle of x^i and s' contains the $\lceil\beta/3\rceil$-middle of y^i.

The claim is shown for $s = [a, b]$ (the proof for s' is similar). Write $x^i = [u, v]$, and $[\hat{u}, \hat{v}]$ the $\lceil\beta/3\rceil$-middle of x^i. First note that since x^i has length at least β, we have $\overline{uv} \geq \beta - 1$. We show that a is in the factor $[u, \hat{u}]$:

$$\begin{aligned}
\overline{u\hat{u}} &= \lfloor(\overline{uv} - \lceil\beta/3\rceil)/2\rfloor \\
&\geq \lfloor(\beta - 1 - \lceil\beta/3\rceil)/2\rfloor \\
&\geq \lfloor(\lfloor 2\beta/3\rfloor - 1)/2\rfloor \\
&\geq \lfloor(2\beta/3 - 1.7)/2\rfloor \\
&\geq \lfloor\beta/3 - 0.85\rfloor \\
&\geq \lceil\beta/3\rceil - 2.
\end{aligned}$$

Since the piece with right endpoint a in the $\lceil\beta/3\rceil$-splitting is fragile (it has not been merged with s), it contains a breakpoint of $\mathcal{P}$ and hence a marker strictly to the left of u. Moreover it has length at most $\lceil\beta/3\rceil$, so $\overline{ua} \leq \lceil\beta/3\rceil - 2$, which implies that a is in the factor $[u, \hat{u}]$. Similarly, b is in the factor $[\hat{v}, v]$, and $[a, b]$ contains the $\lceil\beta/3\rceil$-middle of x^i.

We can now turn to proving the two statements of the fact.

(i) Let $s = [a, b]$, $s' = [a', b']$ and $\delta_1, \delta_2, \ldots, \delta_m$ be the shifts of the $m \geq 7$ alignments such that $-\lceil\beta/3\rceil \leq \delta_1 \leq \delta_2 \leq \ldots \leq \delta_m \leq \lceil\beta/3\rceil$. Write i the index such that $\delta_{i+1} - \delta_i$ is minimal, and $p = \delta_{i+1} - \delta_i$. We thus have

$$\begin{aligned} p &\leq \frac{2\lceil\beta/3\rceil}{m-1} \\ &\leq \lceil\beta/3\rceil/2 \end{aligned}$$

Recall also that both s and s' have length at least $2\lceil\beta/3\rceil - 1$. Let q be an integer with $p \leq q \leq \overline{ab}$. Using the second condition in the definition of alignment, we have

$$\begin{aligned} a \triangleright q &\equiv a' \triangleright (\delta_i + q) \quad (\text{since } a \triangleright q \in [a, b]) \\ &= a' \triangleright (\delta_{i+1} + q - p) \\ &\equiv a \triangleright (q - p) \quad (\text{since } a \triangleright (q - p) \in [a, b]) \end{aligned}$$

Thus factors $[a, b]$ and (symmetrically) $[a', b']$ are both periodic with period length p: the shortest periods of s and s' have length at most $\lceil\beta/3\rceil/2$.

Since s and s' both contain the $\lceil\beta/3\rceil$-middle of the matched blocks in which they are contained, they have a common substring of length greater than twice their shortest periods. They thus have the same shortest period.

(ii) Recall that x^i (y^i) is the block containing s (s') in $\mathcal{P}$. Write $[\hat{u}, \hat{v}]$ ($[\hat{u}', \hat{v}']$) the $\lceil\beta/3\rceil$-middle of x^i (y^i). Since $[\hat{u}, \hat{v}] \subseteq s$ and $\|[\hat{u}, \hat{v}]\| \geq \|\pi\|$, we have that $\mathrm{l}_{\mathrm{break}}(s)$ is the rightmost marker in x and $\mathrm{l}_{\mathrm{break}}(s)$ is the leftmost marker in x such that factors $[\mathrm{l}_{\mathrm{break}}(s), \hat{v}]$ and $[\hat{u}, \mathrm{r}_{\mathrm{break}}(s)]$ do not have period π. We have the similar property for $\mathrm{l}_{\mathrm{break}}(s')$ ($\mathrm{r}_{\mathrm{break}}(s')$) and $\hat{v}'$ ($\hat{u}'$).

Since x^i contains $[\hat{u}, \hat{v}]$, then either x_i has period π, either it contains $\mathrm{l}_{\mathrm{break}}(s)$ or $\mathrm{r}_{\mathrm{break}}(s)$. Suppose that x^i contains $\mathrm{l}_{\mathrm{break}}(s)$ (the case where x^i contains $\mathrm{r}_{\mathrm{break}}(s)$ is similar). Write l' the marker in y^i matched to $\mathrm{l}_{\mathrm{break}}(s)$ by $\mathcal{P}$. Then $[l', \hat{v}'] \equiv [\mathrm{l}_{\mathrm{break}}(s), \hat{v}]$ does not have period π, and for all $m' \in [l' \triangleright 1, \hat{v}']$, $[m', \hat{v}']$ has period π. Thus, l' is the rightmost marker such that $[l', \hat{v}']$ does not have period π, and $l' = \mathrm{l}_{\mathrm{break}}(s')$: markers $\mathrm{l}_{\mathrm{break}}(s)$ and $\mathrm{l}_{\mathrm{break}}(s')$ are matched in $\mathcal{P}$.

□

The following trivial observation follows from the check in Line 11 of `split`. It is useful for bounding the running time of `split` (in particular for later calls to `split`).

Observation 4.2. *After* `split` *has finished, the constraint contains at most* $2k - 2$ *fragile pieces from each of* x *and* y*. The overall number of solid pieces is thus at most* $2k$*.*

To obtain a fixed-parameter algorithm for parameter k, we now "shrink" the fragile pieces between the solid pieces of the constraint. This will ensure that in the next call to `split`, the number of new pieces created in the splitting will be bounded by a function of k. Note that by Lemma 4.1, `split` has discovered *all*

pieces that have length at least β. Hence, we now update the value β denoting the approximate length of the longest short blocks (by taking the largest remaining value from Π). Then, `frames` uses this updated value of β to shrink the fragile pieces. For the moment, we make some claims about `frames`; their proof is deferred to the Sections 4.3 and 4.4. First, we claim that `frames` is correct, that is, there is at least one good branching for yes-instances.

Lemma 4.3. *If there exists a size-k CSP $\mathcal{P}$ satisfying $\mathcal{C}$ at the beginning of* `frames` *such that the longest undiscovered block is β-critical, then* `frames` *creates at least one branch such that the constraint in this branch is satisfied by a size-k CSP $\mathcal{P}'$ whose longest undiscovered block has length at most $2\beta - 1$.*

Second, `frames` increases the exponential part of the running time by a factor that depends only on k.

Lemma 4.4. *Overall, the calls to* `frames` *create $(4k)^{4k^2} \cdot k^{O(k)}$ branches; all other parts of the algorithm can be performed in $\mathrm{poly}(n)$ time.*

Finally, to bound the number of branches in the subsequent call to `split`, and for the case $\beta < 4$, we use the following lemma.

Lemma 4.5. *When* `frames` *terminates, every fragile piece has length at most $12(k^2+k)k\beta$.*

Note that the above also holds before the first call of `split`. Using these lemmas, we obtain our main result.

Theorem 4.6. *The* Minimum Common String Partition *problem can be solved in $k^{21k^2}\,\mathrm{poly}(n)$ time; it is thus fixed-parameter tractable with respect to the partition size k.*

Proof. For the correctness proof assume that the instance is a yes-instance (for a no-instance the algorithm can always check the correctness and size of a CSP before returning, thus it has empty output for no-instances). Then, assuming that the input strings are not identical, there is a CSP $\mathcal{P}$ satisfying the initial constraint $\mathcal{C}$ which demands only that there is at least one breakpoint in x and in y.

We now show that there is a set Π' of powers of 2, all of which are smaller than n such that the algorithm outputs, in at least one of its branches, a size-k CSP, in case the main algorithm loop is traversed for this set Π'.

Let β be the smallest integer such that there is a size-k CSP in which the longest block is β-critical. Then, the largest integer of Π' is β. Now, if $\beta < 4$ the algorithm directly finds all breakpoints by a brute-force branching. Otherwise, the procedure `split` is called. By Lemma 4.1, this procedure creates at least one branch where the constraint is satisfied by some size-k CSP $\mathcal{P}$ and all its blocks of length at least β are discovered by $\mathcal{C}$. Consider an arbitrary branch with this property. Now, let β denote the smallest power of 2 such that there is a CSP satisfying the current constraint $\mathcal{C}$ in which the longest blocks are β-critical. This integer β is the second largest integer of Π'. The algorithm now calls `frames` and by Lemma 4.3 obtains in at least one branch a constraint such that there is a size-k CSP that satisfies the constraint in this branch. Furthermore, also by Lemma 4.3 the longest undiscovered block in this CSP has length at most $2\beta - 1$. By the choice of β, it follows that the longest undiscovered block of this CSP is β-critical. Now, the algorithm either finds all

breakpoints by brute-force (if $\beta < 4$) or again calls the procedure `split` to discover all β-critical blocks. This whole procedure is repeated for smaller and smaller β, each time β is defined as the smallest power of two such that there is a size-k CSP satisfying the current constraint $\mathcal{C}$ whose longest undiscovered block is β-critical. The set Π' contains exactly all integers obtained this way. Eventually, $\beta < 4$ and the algorithm branches by brute-force into all cases to set the breakpoints without violating the current constraint. Clearly, one of these cases is equivalent to a CSP satisfying this constraint. The algorithm verifies that this is indeed a CSP and that it has size at most k and correctly outputs the CSP. Hence, the algorithm is correct.

It remains to show the running time of the algorithm. First, the for-each-loop in the main method is executed $O(2^{\log n}) = O(n)$ times. Second, by the restriction on Π', the repeat-loop in the main method is executed at most k times. To obtain the claimed running time, we bound the number of branches created in each call to `split`.

In each call to `split` the total length of the fragile pieces is less than $(2k)12(k^2+k)k\beta = 24(k^4+k^3)\beta$: In the first call, $\beta > n/2k$, so the bound holds. In the other cases, there are, by Observation 4.2 at most $2k-2$ fragile pieces in x and y. Furthermore, in this case `split` is called after `frames`. Thus, by Lemma 4.5, each fragile piece has length at most $12(k^2+k)k\beta$, and the overall bound follows.

The procedure splits the fragile pieces into new pieces of length at most $\lceil \beta/3 \rceil$ (i.e. there is a distance $\lceil \beta/3 \rceil - 1$ between the left endpoints of two consecutive pieces of the same splitting). Since $\beta \geq 4$, we have $\lceil \beta/3 \rceil - 1 \geq \beta/6$. Hence, this creates less than $144(k^4+k^3)$ new pieces of length $\lceil \beta/3 \rceil$ plus at most one additional shorter piece at the end of each fragile piece. Hence, $145k^4$ is an upper bound on the number of new pieces. Branching for each piece into the case that it is solid or fragile can be done in 2^{145k^4} branches. The number of necessary branches for this part of `split` can be reduced as follows: Since we merge series of consecutive pieces in F or S, and since we do not need to consider branches with more than k solid pieces, we can directly look for the first and last piece of each β-critical block. This creates $O(\binom{145k^4}{4k}) = O(\frac{(145k^4)^{4k}}{(4k)!})$ branches in each call of `split`.

The matching requires up to $k!$ branches, and the alignment at most 6^k. Since $145^{4k}k!6^k = o((4k)!)$, we can bound the number of branches in each call of `split` by k^{16k}. The `split` procedure is called at most k times (by the restriction on Π), thus creating $O(\left(k^{16k}\right)^k) = O(k^{16k^2})$ branches throughout the algorithm. Finally, the number of branches created in `frames` is $(4k)^{4k^2} \cdot k^{O(k)}$ by Lemma 4.4, and the number of branches created in the final brute-force can be bounded as follows. The length of the fragile pieces is $O(k^4+k^3)$ and we need to guess at most $2k-2$ precise breakpoint positions from this number. This can be done with $k^{O(k)}$ branches.

Finally, note that all other steps of the algorithm can be clearly performed in polynomial time. Altogether, the total running time of the algorithm thus is

$$O(k^{2k}n) \cdot k^{16k^2} \cdot k^{O(k)} \cdot (4k)^{4k^2} \cdot k^{O(k)} \cdot \mathrm{poly}(n) = k^{21k^2}\,\mathrm{poly}(n).$$

□

4.3 Putting Frames Next to Fixed Pieces

In this and the next section, we prove the two claimed lemmas concerning `frames`. Informally, we show that, with the right constraint in the beginning, `frames` finds

a constraint $\mathcal{C}$ that is satisfied by a size-k CSP $\mathcal{P}$ whose longest undiscovered block has length at most $2\beta - 1$. Moreover, the length of each fragile piece is $O(k^3\beta)$ in every constraint produced by `frames`. The pseudo code of `frames` is shown in Algorithm 4.3.

The approach of `frames` is to use a set of reduction rules to put "frames" into the fragile pieces, where a frame is a factor within the fragile piece that contains *all* breakpoints that are contained in this piece. We call the actual shortest factor containing all breakpoints of a fragile piece a "window", defined as follows. Let $\mathcal{P}$ be a size-k CSP satisfying $\mathcal{C}$, and let f be a fragile piece in $\mathcal{C}$. The *window* of f is the factor $[a, b]$ such that $\{a, a \triangleright 1\}$ is the leftmost breakpoint of $\mathcal{P}$ in f and $\{b \triangleleft 1, b\}$ is the rightmost breakpoint of f. Since a frame is required to contain all breakpoints of a fragile piece it can be seen as a "super"-approximation of the actual window. A formal definition of frames is as follows.

Definition 4.6 (Frames)**.** *Let $\mathcal{C}$ be a constraint. A* frame $[a, b]$ *for a fragile piece f of $\mathcal{C}$ is a factor that is contained in f. A* frame set *for $\mathcal{C}$ is a set Φ of frames such that each fragile piece f contains at most one frame. A CSP $\mathcal{P}$ that satisfies $\mathcal{C}$* satisfies *a frame set Φ for $\mathcal{C}$ if each breakpoint of $\mathcal{P}$ is contained in some frame of Φ.*

The approach to add the frames to the constraint can be summarized as follows: first, we compute an upper bound w on the length of the windows that only depends on β and k. The $w = 2\beta k + 1$ is easily obtained since each window can contain at most k undiscovered blocks of maximum length 2β. Then, we apply a series of *frame rules* that eventually place a frame in all fragile pieces (Lines 4–5). As we will show, the frame length then depends on w (and thus on k and β) and on the maximum period length of the unfixed (repetitive) solid pieces. Since the frames contain all breakpoints of $\mathcal{P}$, it is possible to reduce fragile pieces until they "fit" their frames (Line 6). We now check whether there are some unfixed solid pieces with a long period compared to w. If this is the case, then the number of "feasible" alignments for these pieces is small, and we can thus branch how to align these pieces (Lines 7–11). Formally, we call an alignment of $s = [a, b]$ and $s' = [a', b']$ *feasible* for $\mathcal{C}$ if the factor equidistant to $[a, b]$ ($[a', b']$) from s does not intersect any other solid piece than s' (s) in $\mathcal{C}$. Note that each satisfying CSP can only have feasible alignments, otherwise there is at least one fragile piece without breakpoints.

Afterwards, we go back to applying the frame rules (we will obtain shorter frames since the number of fixed pieces has increased). If this is not the case, that is, all periods are short compared to w, then we show that the maximum frame length depends only on w. Hence, in this case they are sufficiently short, and the `frames` procedure has achieved its goal. The algorithm thus returns to the main method where it calls `split` to find new solid pieces.

In this section, we describe the frame rules that place frames in fragile pieces which are next to fixed pieces and some further simple frame rules. Before doing so, we define two concepts that will be used by the frame rules: *maximum extensions* and the *piece graph*. Roughly speaking, maximum extensions are used locally to bound the position of some breakpoints in the fragile pieces. The piece graph provides a structural view of the relationship between pieces and is used to show that one of the frame rules can always be applied in case there is a frameless fragile piece.

Algorithm 4.3 Procedure `frames`. Global variables: $\mathcal{C}$, β.

```
1: w ← 2βk + 1                                   // upper bound on window length
2: repeat :
3:     Compute the maximum extension of each solid piece,
       the piece graph G[C, Φ ← ∅], and the strips of each rep−rep path
4:     while there is a frameless fragile piece :    // place frames in fragile pieces
5:         apply Frame Rules 1–6
6:     for each fragile piece : apply Fitting Rule 1
7:     new-align ← False                          // fix pieces with long periods
8:     for each repetitive solid piece s (with period π_s) :
9:         if all fragile pieces adjacent to s or s' have length at most 6(k^2 + k) ||π_s|| :
10:            for each feasible alignment branch into the case to add this alignment to R_S
11:            new-align ← True
12: until new-align = False
```

Maximum extension of solid pieces. Let s be a solid piece in a constraint $\mathcal{C}$. The *maximum extension* of s is the factor $[\mathrm{l}_{\mathrm{ext}}(s), \mathrm{r}_{\mathrm{ext}}(s)]$ containing s where $\mathrm{r}_{\mathrm{ext}}(s)$ and $\mathrm{l}_{\mathrm{ext}}(s)$ are defined as follows. If s is fixed, then let ℓ be the largest integer such that $[s^*, s^* \triangleright \ell] \equiv [s'^*, s'^* \triangleright \ell]$, and that no marker of $[s^*, s^* \triangleright \ell]$ or $[s'^*, s'^* \triangleright \ell]$ is in a solid piece other than s or s'. Then $\mathrm{r}_{\mathrm{ext}}(s) = s^* \triangleright \ell$ and $\mathrm{r}_{\mathrm{ext}}(s') = s'^* \triangleright \ell$. If s is repetitive with shortest period π_s, then let a be the leftmost marker in s and define $\mathrm{r}_{\mathrm{ext}}(s)$ as the rightmost marker such that the factor $[a, \mathrm{r}_{\mathrm{ext}}(s)]$ has period π_s, and that no marker in $[a, \mathrm{r}_{\mathrm{ext}}(s)]$ is in a solid piece other than s. Marker $\mathrm{l}_{\mathrm{ext}}(s)$ is obtained symmetrically.

The following proposition is a straightforward consequence of the definition of maximum extension.

Proposition 4.7. *Let s be a fixed solid piece, and let $[a, b]$ and $[c, d]$ be two factors that are equidistant from s and such that $[a, b]$ is contained in $[\mathrm{l}_{\mathrm{ext}}(s), \mathrm{r}_{\mathrm{ext}}(s)]$ and $[c, d]$ is contained in $[\mathrm{l}_{\mathrm{ext}}(s'), \mathrm{r}_{\mathrm{ext}}(s')]$. Then, $[a, b] \equiv [c, d]$.*

Note that, as a special case, the above proposition includes single markers (that is, length-one factors). The next proposition simply states formally that the maximum extensions of a solid piece contain the block which contains the solid piece.

Proposition 4.8. *Let $\mathcal{C}$ be a constraint and s be a solid piece of $\mathcal{C}$. Any CSP that satisfies $\mathcal{C}$ has a block which contains s and is contained in $[\mathrm{l}_{\mathrm{ext}}(s), \mathrm{r}_{\mathrm{ext}}(s)]$. Furthermore, let f be a fragile piece next to s. Then, the window in f contains at least one marker of $[\mathrm{l}_{\mathrm{ext}}(s), \mathrm{r}_{\mathrm{ext}}(s)]$.*

Proof. If s is fixed, then the block containing s cannot contain the marker $\mathrm{l}_{\mathrm{ext}}(s) \triangleleft 1$: if this marker is contained in the block, it is matched to $\mathrm{l}_{\mathrm{ext}}(s') \triangleleft 1$. By definition of $\mathrm{l}_{\mathrm{ext}}(s)$, either $\mathrm{l}_{\mathrm{ext}}(s) \triangleleft 1 \not\equiv \mathrm{l}_{\mathrm{ext}}(s') \triangleleft 1$ or one of $\mathrm{l}_{\mathrm{ext}}(s) \triangleleft 1$, $\mathrm{l}_{\mathrm{ext}}(s') \triangleleft 1$ belongs to a different solid piece t. In the first case, we do not obtain a CSP; in the second case, there is at least one fragile piece without a breakpoint. Similarly, the block containing s cannot contain $\mathrm{r}_{\mathrm{ext}}(s) \triangleright 1$.

Every repetitive solid piece s is contained in a block which is periodic with the same shortest period π as s. By definition of $\mathrm{l}_{\mathrm{ext}}()$ and $\mathrm{r}_{\mathrm{ext}}()$ for repetitive solid pieces this block must thus be contained in $[\mathrm{l}_{\mathrm{ext}}(s), \mathrm{r}_{\mathrm{ext}}(s)]$.

Finally, consider a fragile piece f to the right (to the left) of s. The window in f contains the last (first) marker of the block containing s. By the above it thus contains at least one marker of $[\mathrm{l}_{\mathrm{ext}}(s), \mathrm{r}_{\mathrm{ext}}(s)]$. □

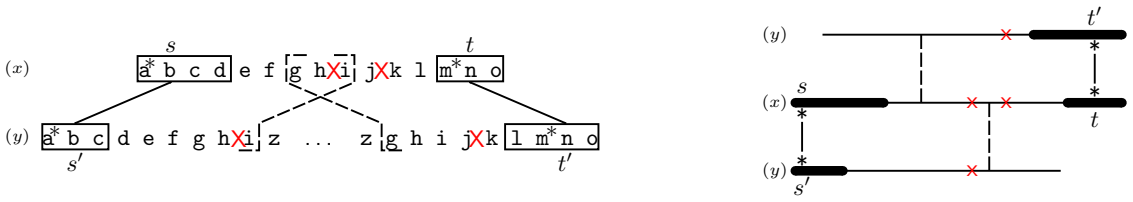


Figure 4.4: Left: two pairs of fixed solid pieces (s, s') and (t, t'). Reference markers are shown with an asterisk, maximum extensions are delimited with dashed lines, and the breakpoints of some possible CSP are marked with red crosses. Right: a simplified representation of the same pieces, where thick (resp. thin) lines are used for solid (resp. fragile) pieces.

The Piece Graph. Given a constraint $\mathcal{C}$ and a frame set Φ, the *piece graph* $G[\mathcal{C}, \Phi]$ is the bipartite graph $G = (V_S \cup V_F, E)$ constructed as follows.

- V_F contains one vertex v_f for each frameless fragile piece $f \in F$,
- V_S contains, for each repetitive solid piece $s \in S_x$ a vertex v_s, and for each fixed piece $s \in S_x$, two vertices l_s and r_s (for *left* and *right*).
- For a fixed solid piece s and a fragile piece $f \in F$, G contains the edge $\{v_f, l_s\}$ if the last marker of f is the first marker of s or of s', and the edge $\{v_f, r_s\}$ if the first marker of f is the last marker of s or of s'.
- For an unfixed solid piece s, G contains the edge $\{v_f, v_s\}$ if the first marker of f is the last marker of either s or s' or if the last marker of f is the first marker of either s or s'.

Note that the vertices v_s or l_s and r_s are only defined for pieces $s \in S_x$, but they represent both pieces s and s'. Observe furthermore that in case $V_F \neq \emptyset$, there are fragile pieces in $\mathcal{C}$ that do not have a frame in Φ. Moreover, note that in this case the edge set of the piece graph is nonempty. Our aim will thus be to gradually apply the frame rules until the piece graph is edge-less. Each vertex is called *fragile*, *fixed* or *repetitive* depending on the nature of the piece it represents. Note that most vertices of the graph have degree at most 2, except for repetitive vertices which can have degree up to 4. Vertices with smaller degree correspond initially to the four pieces at the end of the sequences.

In order to deal seamlessly with pieces at the end of the input strings (where no fragile piece is adjacent on one side), we introduce "phantom frames" as follows. If s contains the first element of a string, i.e. $x[1]$ or $y[1]$, we say that s has the *phantom frame* $[x[0], x[1]]$ (resp. $[y[0], y[1]]$) to its left. Likewise, if s contains $x[n]$ or $y[n]$, it has the *phantom frame* $[x[n], x[n+1]]$ (resp. $[y[n], y[n+1]]$) to its right.

We now have collected the prerequisites to state Frame Rules 1 to 3 (Frame Rules 4 to 6 use the notion of rep−rep path and strip, they are defined in Section 4.4). A frame rule is an algorithm that receives as input a constraint $\mathcal{C}$ and a frame set Φ and updates both into a constraint $\mathcal{C}$' and a frame set Φ'. A frame rule is *correct* if the following holds. First, if there is a size-k CSP $\mathcal{P}$ satisfying $\mathcal{C}$ and Φ, then there is also a size-k CSP $\mathcal{P}'$ satisfying $\mathcal{C}'$ and Φ'. Second, the longest undiscovered block in $\mathcal{P}'$ is at most as long as the longest undiscovered block in $\mathcal{P}$ (this additional restriction will be used to argue that the choice of β remains correct). Note that wlog., we describe all rules so that they consider a frameless fragile piece in x but they apply equally to fragile pieces in y. Finally, we state the additional frames of all rules by defining a factor which contains the window, in order to ensure that the frames are within the fragile pieces, we always intersect this factor with the

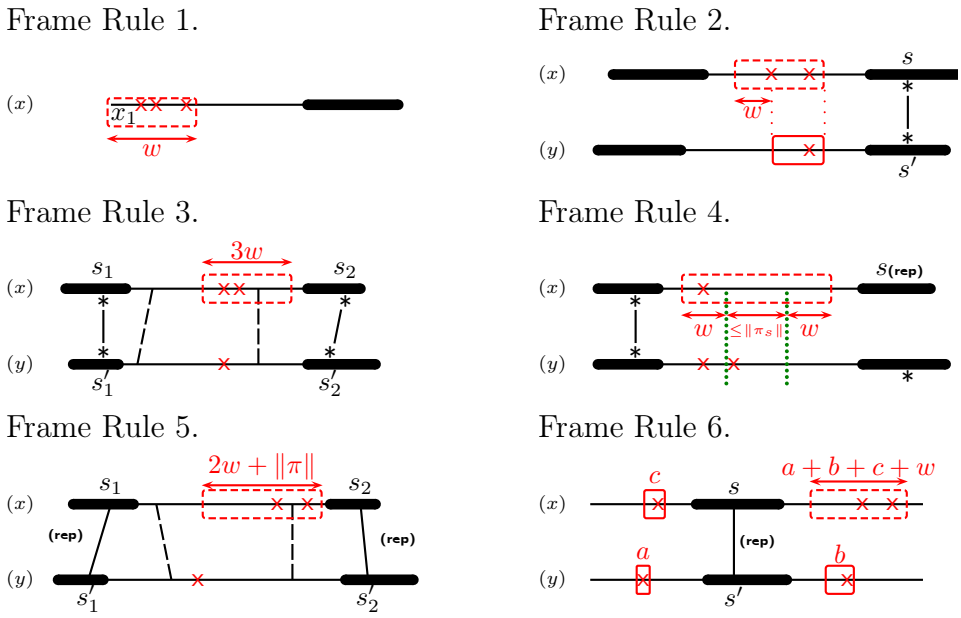


Figure 4.5: Frame Rules 1–6 of `frames`. Frames are drawn as red boxes, the frame created at each step is dashed. Possible breakpoint positions in $\mathcal{P}$ are shown as red crosses.

considered fragile piece f. The first rule puts frames into fragile pieces at the end of the string.

Frame Rule 1. *If the piece graph contains a fragile degree-one vertex v_f, then f contains either $x[1]$ or $x[n]$. If f contains $x[1]$ add $f \cap [x[1], x[1] \triangleright w]$ to Φ, otherwise add $f \cap [x[n] \triangleleft w, x[n]]$ to Φ.*

Proof. Fragile pieces of x that do not contain the first or the last marker of x are preceded and followed by a solid piece (since the splitting is alternating) and thus the corresponding vertex in the piece graph has degree two. Hence, a fragile piece in x corresponding to a degree-one vertex of the piece graph contains either the first or the last marker of x. Assume wlog. that f contains $x[1]$. The leftmost block of $\mathcal{P}$ in x is necessarily a short block since it is contained in the fragile piece f. Hence, marker $x[1]$ belongs to the first undiscovered block of $\mathcal{P}$. Since the window (which contains all undiscovered blocks in f) has length at most w, it is contained in the created frame $[x[1], x[1] \triangleright w]$. □

Frame Rule 2. *If the piece graph contains a degree-one vertex l_s (or r_s, the rule is symmetrical) with neighbor v_f such that f is next to s, then: let $[s'^* \triangleleft u, s'^* \triangleleft v]$ be the (possibly phantom) frame to the left of s' in y; add the frame $f \cap [s^* \triangleleft (u+w-1)), s^* \triangleleft v]$ to Φ.*

Proof. Consider first the case where $[s'^* \triangleleft u, s'^* \triangleleft v]$ is a phantom frame: in this case, $s'^* \triangleleft v$ is $y[1]$ and $u = v + 1$. Hence, $y[1]$ is the first element of the block containing s'. Since $y[1]$ and $s^* \triangleleft v$ are equidistant from s, $s^* \triangleleft v$ is the first element of the block containing s and the last element of the window in f. Since the window has length at most $w-1$, it is contained in the frame $[s^* \triangleleft (v+w), s^* \triangleleft v] = [s^* \triangleleft (u+w-1)), s^* \triangleleft v]$.

Consider now the (regular) case where s' has a fragile piece g to its left. By the frame definition, all breakpoints of a satisfying CSP $\mathcal{P}$ that are in g are within

$[s'^* \triangleleft u, s'^* \triangleleft v]$. Hence, $s'^* \triangleleft v$ is in the same block as s'. Consequently, the right limit of the window in f is to the left of $s^* \triangleleft v$ in f. Similarly, $s'^* \triangleleft u$ is in a different block than s' and thus there is a breakpoint to the right of $s^* \triangleleft u$ in f. All other breakpoints in f can have distance at most w from this breakpoint. Hence, all breakpoints in f are contained in the created frame $[s^* \triangleleft (u + w - 1), s^* \triangleleft v]$. □

The above rules are relatively straightforward inferences of frame positions that can be made because the piece graph has degree-one vertices. We now show some more intricate rules that deal with the remaining cases. In particular, we show how to deal with cycles in the piece graph. We first consider cycles without repetitive solid pieces. Note that the following rule performs a branching. We thus extend the correctness notion to hold if there is at least one branch in which the created constraint and frame set can be satisfied.

Frame Rule 3. *If the piece graph contains a simple cycle without repetitive vertices, then create one branch for each edge $\{v_f, u_s\}$ of this cycle. In each branch, add to Φ the frame*

- *$f \cap [\mathrm{l}_{\mathrm{ext}}(s) \triangleleft w, \mathrm{l}_{\mathrm{ext}}(s) \triangleright (2w)]$ if $u_s = l_s$ for some solid piece s, or*
- *$f \cap [\mathrm{r}_{\mathrm{ext}}(s) \triangleleft (2w), \mathrm{r}_{\mathrm{ext}}(s) \triangleright w]$ to f if $u_s = r_s$ for some solid piece s.*

The following is a straightforward property of constraints and satisfying solutions and used for showing the correctness of Frame Rule 3.

Proposition 4.9. *Let s be a fixed solid piece in a constraint $\mathcal{C}$. If markers a and a' are equidistant from s, then for any integer i, $a \triangleright i$ and $a' \triangleright i$ are equidistant from s. Moreover, given a CSP $\mathcal{P}$ satisfying $\mathcal{C}$, the first markers (the last markers) of the blocks of $\mathcal{P}$ containing s and s' are equidistant from s.*

Proof. The first part is directly obtained by definition:

$$\overline{s^*(a \triangleright i)} = \overline{s^* a} + i = \overline{s'^* a'} + i = \overline{s'^*(a' \triangleright i)}.$$

For the second part, simply note that if s^* is at position j in the block containing s, then s'^* is also at position j in s'. Hence, the first markers (and thus also the last markers) of both blocks are equidistant from s. □

Proof. Let $\mathfrak{P}$ be the set of CSPs that satisfy the constraint $\mathcal{C}$ and frame set Φ and additionally have a minimum total length of short blocks. We show that there is a $\mathcal{P} \in \mathfrak{P}$ which has all breakpoints in $[\mathrm{l}_{\mathrm{ext}}(s) \triangleleft w, \mathrm{l}_{\mathrm{ext}}(s) \triangleright (2w)]$ for some vertex l_s of the cycle, thus showing correctness of the rule.

Since the piece graph $G[\mathcal{C}, \Phi]$ is bipartite with partition V_S and V_F, the cycle alternates between vertices of V_S and V_F. Moreover, all cycle vertices from V_S are fixed, and alternate between left and right vertices (each fragile vertex of the cycle is adjacent to a left vertex and to a right vertex). Hence there exist solid pieces $s_1, s_2, \ldots, s_\ell$ and fragile pieces $f_1, f_2, \ldots, f_\ell$ such that the cycle is $(l_{s_1}, v_{f_1}, r_{s_2}, v_{f_2}, \ldots, l_{s_{\ell-1}}, v_{f_{\ell-1}}, r_{s_\ell}, v_{f_\ell})$. For simplicity, we consider indices only modulo ℓ (that is, $s_{\ell+1} = s_1$, $f_0 = f_\ell$, etc.), and we assume that fragile pieces with odd indices are in x and those with even indices are in y. Consider a CSP $\mathcal{P} \in \mathfrak{P}$ such that there is no l_s (r_s) with an adjacent window contained in $[\mathrm{l}_{\mathrm{ext}}(s) \triangleleft w, \mathrm{l}_{\mathrm{ext}}(s) \triangleright (2w)]$ ($[\mathrm{r}_{\mathrm{ext}}(s) \triangleleft (2w), \mathrm{r}_{\mathrm{ext}}(s) \triangleright w]$). We transform this CSP into one that fulfills this property. We first prove that in $\mathcal{P}$ either all fragile pieces with odd or all fragile pieces with even indices contain only one breakpoint. Assume towards a contradiction,

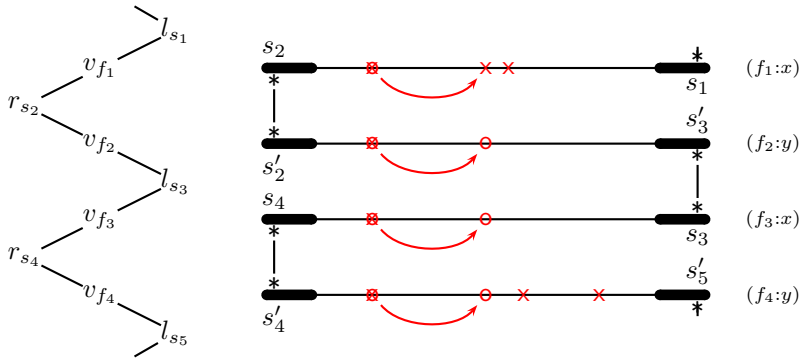


Figure 4.6: Illustration for the first part of the correctness proof of Frame Rule 3. If two fragile windows f_i, f_j with different parity have several breakpoints (here, $i = 1$ and $j = 4$), then we can shift the position of the leftmost breakpoint in each fragile piece of the path to reduce the length of short blocks. The modifications (breakpoints added or deleted) are shown as red circles.

that there exist integers $i < j$ of different parity such that f_i and f_j both have windows with at least two breakpoints and for each h with $i < h < j$, f_h contains only one breakpoint. Assume wlog. that i is odd and j is even. Hence, f_i is in x to the right of s_{i+1} and f_j is in y to the right of s_j.

For all h, $i \leq h \leq j$, let a_h be the leftmost marker of the window in f_h, and $b_h = a_h \triangleright 1$. For odd h, a_h and a_{h+1} are the rightmost markers of the blocks containing s_{h+1} and s'_{h+1} and thus equidistant from s_{h+1}. For even $h < j$, b_h and b_{h+1} are the left endpoints of the blocks containing s_{h+1} and s'_{h+1}, so they are equidistant from s_{h+1}. By Proposition 4.9, for all $i \leq h < j$, $[a_h, b_h]$ and $[a_{h+1}, b_{h+1}]$ are equidistant from s_{h+1}. By definition of a_h, the window in each f_h is contained in $[a_h, a_h \triangleright w]$. If one of these factors is not contained in the maximum extension of an adjacent solid piece, say $[a_h, a_h \triangleright w]$ is not contained in the maximum extension of s_{h+1}, then $\mathrm{l}_{\mathrm{ext}}(s_{h+1})$ is contained in $[a_h, a_h \triangleright w]$. Hence, the window is contained in $[\mathrm{l}_{\mathrm{ext}}(s_{h+1}) \triangleleft w, \mathrm{l}_{\mathrm{ext}}(s_{h+1}) \triangleright w]$, contradicting our assumption on $\mathcal{P}$. In the following, we thus assume that all factors $[a_h, a_h \triangleright w]$ are contained in the maximum extension of adjacent solid pieces, which by Proposition 4.7 implies that they all have the same content. In particular, this implies $[a_i, a_i \triangleright w] \equiv [a_j, a_j \triangleright w]$.

We now describe a modification of $\mathcal{P}$ that results in a new CSP which is not larger than $\mathcal{P}$, also satisfies the constraint and frame set but has smaller total length of short blocks; the modification is illustrated in Figure 4.6. Let $u + 1$ and $v + 1$ be the lengths of the leftmost short blocks in f_i and f_j respectively (assume wlog. that $u \leq v$). These two short blocks are thus $[b_i, b_i \triangleright u]$ and $[b_j, b_j \triangleright v]$, and they are matched in $\mathcal{P}$ to other short blocks $[b'_i, b'_i \triangleright u]$ and $[b'_j, b'_j \triangleright v]$. Note that since f_i is odd and f_j is even, $[b_i, b_i \triangleright u]$ is in a different string than $[b_j, b_j \triangleright v]$. To create the new solution $\mathcal{P}$' from $\mathcal{P}$ apply the following modifications. First, cut out $u + 1$ markers from the left of $[b'_j, b'_j \triangleright v]$ (recall that $u \leq v$) which gives two new blocks $[b'_j, b'_j \triangleright u]$ and $[b'_j \triangleright (u + 1), b'_j \triangleright v]$ if $u < v$ and leaves the block $[b'_j, b'_j \triangleright v]$ unmodified if $u = v$. Now, match block $[b'_i, b'_i \triangleright u]$ to $[b'_j, b'_j \triangleright u]$ (recall that these blocks are in different strings). Now, shift the breakpoints of the fragile pieces of the cycle as follows. For every odd h, $i < h < j$, cut out $u + 1$ markers from the left of s'_h and s_h. And for every even h, $i < h \leq j$, add $u + 1$ markers to the right of the blocks containing s_h and s'_h. Finally, in case $u < v$, match the shortened block $b_j \triangleright (u + 1), b_j \triangleright v$ to the block $b'_j \triangleright (u + 1), b'_j \triangleright v$ created in the first step. Note that by the discussion

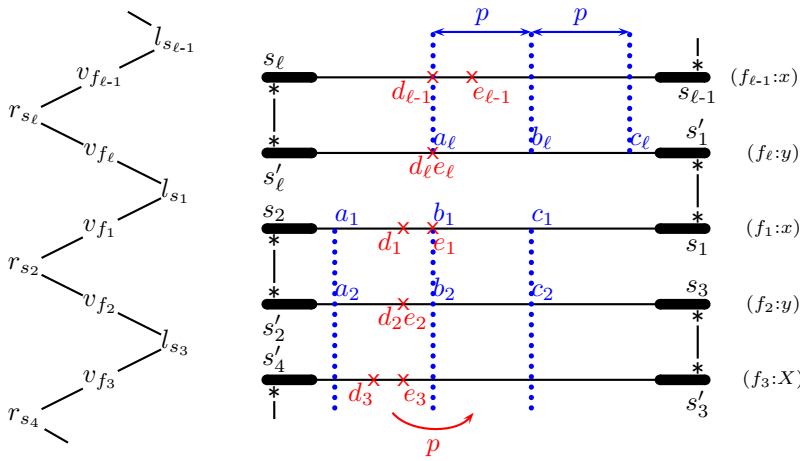


Figure 4.7: Illustration for the correctness proof of the second part of Frame Rule 3. Given a cycle with total length of short blocks $p > 0$ we construct factors $[a_h, b_h]$ and $[b_h, c_h]$ as shown (delimited by blue dotted lines). All the breakpoints in factors $[a_h, b_h]$ can be shifted to the corresponding $[b_h, c_h]$.

above, the pieces added to s_h and s'_h for even h have the same content. Hence, all matched blocks have equal content. Furthermore, since the block $[b_i, b_i \triangleright u]$ is now unmatched, its markers are free to be added to s_{i+1}.

This new solution has at most as many blocks as $\mathcal{P}$: we have created at most one new breakpoint in $[b'_j, b'_j \triangleright v]$ and removed a breakpoint in f_i by adding exactly $u+1$ markers to the right of s_{i+1}. For all other fragile pieces f_h, the breakpoint has "only" been shifted to the right. Furthermore, $\mathcal{P}'$ satisfies the same constraint $\mathcal{C}$ as $\mathcal{P}$: the matching only changed between short blocks which are not constrained. Moreover, the fragile pieces for which the breakpoints have been modified are either frameless (if they are on the cycle) or the modification adds a breakpoint that is between two breakpoints (in the modification of $[b'_j, b'_j \triangleright v]$) However,the total length of the short blocks has been reduced by $2(u+1)$, which contradicts the choice of $\mathcal{P}$. We now know that in $\mathcal{P}$ the short blocks of the cycle are either all in x or all in y. In the following, we assume they are all in x, that is, in fragile pieces f_j with odd j. We now consider the following two cases: either there is no short block, even in x, or there is at least one.

First consider the case that there is no short block in the cycle, that is, all the windows contain only one breakpoint $[a_h, b_h]$. If all markers b_h are within the maximum extensions of both adjacent solid pieces, we create a new solution $\mathcal{P}'$ from $\mathcal{P}$ as follows: for every odd h, cut out b_h and b_{h-1} from the left end of the blocks containing s_h and s'_h, and for every even h, add b_h and b_{h-1} to the right end of the blocks containing s'_h and s_h. The solution $\mathcal{P}'$ satisfies the same constraints as $\mathcal{P}$, with the same total length of short blocks. Repeat this operation of shifting the breakpoints to the right until for some i (wlog., assume i is even), b_i is to the right of $\mathrm{r}_{\mathrm{ext}}(s_i)$. Then, the rule is correct, since for some branch the edge (r_{s_i}, v_{f_i}) is selected and the frame $[\mathrm{r}_{\mathrm{ext}}(s_i) \triangleleft 2w, \mathrm{r}_{\mathrm{ext}}(s_i) \triangleright w]$ which contains the only breakpoint of $\mathcal{P}$ in f_i is added to Φ.

It remains to show the case where there is at least one short block in the fragile pieces of the cycle, that is, the total length p of the short blocks of the cycle is at least one. Note that by the choice of w, $p < w$. We now show that the strings around the windows are periodic with period length p, so that we can again shift all

the breakpoints of the fragile pieces to the right by steps of length p, until at least one of them has distance at most p from the end of a maximum extension.

We first introduce some notations (see Figure 4.7 for an illustration): for each h, let $[d_h, e_h]$ denote the window of f_h. Let $b_1 = e_1$, $a_1 = b_1 \triangleleft p$, $c_1 = b_1 \triangleright p$, and for each h, $2 \leq h \leq \ell$, let a_h, b_h, and c_h be the markers equidistant with a_{h-1}, b_{h-1}, and c_{h-1} from s_h.

We first show that for every h with $2 \leq h \leq \ell$, we have

$$\overline{e_h b_h} = \overline{d_{h-1} b_{h-1}} - 1. \tag{4.1}$$

For even values of h, d_{h-1} and d_h are equidistant from s_h, so $\overline{d_h b_h} = \overline{d_{h-1} b_{h-1}}$. Since f_h is in string y, it contains only one breakpoint, and thus $\overline{d_h e_h} = 1$ and Equation (4.1) follows. For odd values of h, we have $\overline{d_{h-1} e_{h-1}} = 1$, and e_{h-1} and e_h are equidistant from s_h, thus $\overline{e_h b_h} = \overline{e_{h-1} b_{h-1}}$, which also implies equation (4.1). Hence the distance between window endpoint e_h and the marker b_h increases, compared to the distance of e_{h-1} and b_{h-1}, by the length of the short blocks contained in the window of f_{h-1}. This has two implications: first, in f_ℓ, we have $\overline{e_\ell b_\ell} = p$ and thus $e_\ell = a_\ell$ (by definition a_1 has distance p from b_1, and this distance is conserved through the cycle). Second, for every j, the short blocks in f_j are contained in $[a_j, b_j]$, and the window is contained in $[a_j \triangleleft 1, b_j]$.

First, consider the case where each factor $[a_h, c_h]$ is contained in the maximal extensions of both adjacent blocks. Thus, with Proposition 4.7, we have $[a_h, b_h] \equiv [a_1, b_1]$ and $[b_h, c_h] \equiv [b_1, c_1]$ for all h. We can now "close" the cycle: since e_ℓ and e_1 are the left endpoints of the blocks containing s'_1 and s_1, they are equidistant from s_1. Moreover, $e_\ell = a_\ell$ and $e_1 = b_1$, so a_ℓ and b_1 are equidistant from s_1, which implies that $[a_\ell, b_\ell] \equiv [b_1, c_1]$. This now implies that, for all h, $[a_h, b_h] \equiv [b_h, c_h]$. We now create a solution $\mathcal{P}'$ from $\mathcal{P}$ as follows: for odd values of h, cut out the p leftmost markers from each block containing s_h or s'_h. For even values of h, add p markers to the right of blocks containing s_h or s'_h for even values of h. Match every short block that was matched to some $[u, v]$ in some f_h to $[u \triangleright p, v \triangleright p]$ instead. The solution $\mathcal{P}'$ is again a CSP satisfying the same constraints, with the same total length of short blocks but with all the breakpoints in the cycle shifted to the right by p positions. Repeat this operation until for some h the factor $[a_h, c_h]$ is no longer contained in the maximal extension of the block to its left. Then, $[a_h, c_h]$ contains $\mathrm{r}_{\mathrm{ext}}(s_h)$, and factor $[a_h \triangleleft 1, b_h]$ is contained in $[\mathrm{r}_{\mathrm{ext}}(s_h) \triangleleft w, \mathrm{r}_{\mathrm{ext}}(s_h) \triangleright (2w =]$. As argued above, the rule is correct if such a $\mathcal{P} \in \mathfrak{P}$ exists. Note that the modifications made in the proof do not increase the length of any short block. Hence, the second requirement for correctness is also satisfied.

□

The rules presented so far deal with fixed solid pieces. In fact, if all solid pieces are fixed, then these rules suffice to obtain frames in all fragile pieces. We deal with the presence of repetitive solid pieces with the frame rules presented in the following section.

4.4 Frame Rules for Repetitive Pieces

In the rules, we have to deal with cycles in the piece graph that contain some repetitive vertices. We introduce the following concepts in order to analyze the structure of paths between repetitive vertices that contain fixed solid vertices. A rep−rep

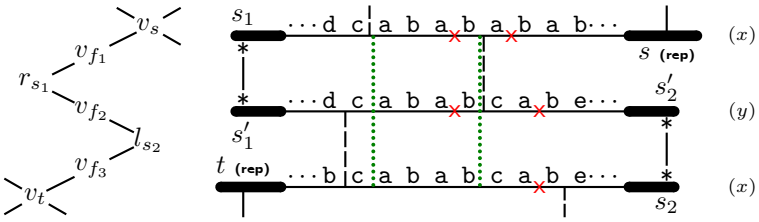


Figure 4.8: Example of a rep−rep path joining repetitive vertices v_s and v_t (with respective periods `ab` and `ababc`), and going through three fragile vertices and their adjacent fixed vertices. The strip of each fragile piece is delimited by the green dotted lines.

path $(v_s, v_{f_1}, u_1, v_{f_2}, u_2, \ldots, u_{\ell-1}, v_{f_\ell}, v_t)$ is a simple path of the piece graph such that the two endpoints v_s and v_t are repetitive vertices, and each u_i is a fixed solid vertex. Given a rep−rep path joining repetitive vertices v_s, v_t and going through fragile vertices $v_{f_1}, v_{f_2}, \ldots, v_{f_\ell}$, we define the *strip* of the path (see Figure 4.8) as a set of factors $\{I_{f_1}, I_{f_2}, \ldots, I_{f_\ell}\}$ such that:

1. Consecutive factors $I_{f_i}, I_{f_{i+1}}$ are equidistant from the solid piece represented by u_i.
2. Each factor I_{f_i} is contained in the maximum extensions of the two solid pieces next to f_i.
3. The length of I_{f_1} is maximal under Conditions 1 and 2.

Proposition 4.10. *All the strips in a rep−rep path have the same length and content. Each factor of the strip is contained in its respective fragile piece. Moreover, the strip of a rep−rep path is uniquely defined and computable in polynomial time.*

Proof. The fact that strips have the same length and content is a direct consequence of Proposition 4.7, which can be applied according to Conditions 1 and 2. Each strip must be contained in its fragile piece since it is in the intersection of the maximum extensions of the two adjacent solid pieces.

The second part of the claim can be seen by considering the following algorithm to compute the strip. First, check whether the strip is nonempty. That is, try the following for each marker a_1 in f_1. Compute the marker a_2 in f_2 that is equidistant with a_1 from u_1. Then, compute the marker a_3 in f_3 that is equidistant with a_2 from u_2, and so on. If all a_i's are in the maximum extensions of both solid pieces next to f_i, then the strip is nonempty. Otherwise, the length of I_{f_1} is zero. Now, assume the case that there was one a_1 for which the above procedure is successful, that is, I_{f_1} contains one or more markers. Then, set $I_{f_i} = \{a_i\}$ for each i. Now try to simultaneously expand all I_{f_i}'s. That is, check whether one can add the marker to the left of each I_{f_i} without violating Condition 2 of the strip definition. If this is the case, then add these markers to the I_{f_i}'s. If this is not the case, then continue by adding markers to the right until this is also not possible anymore. The resulting set of I_{f_i}'s is the strip of the rep−rep-path. □

Proposition 4.11. *Let $\mathcal{P}$ be any solution satisfying constraint $\mathcal{C}$ such that the total length of all windows in $\mathcal{P}$ is p. In each fragile piece f of a rep−rep path of $\mathcal{C}$, writing $I_f = [c, d]$, the window of f is contained in $[c \triangleleft p, d \triangleright p]$.*

Proof. We first introduce some notations: let $f_1, f_2, \ldots, f_\ell$ be the fragile pieces of the path, and, for every $1 \leq j \leq \ell$, let $[a_j, b_j]$ denote the window of f_j, $I_{f_j} = [c_j, d_j]$, $\alpha_j = \overline{d_j a_j}$ and $\beta_j = \overline{d_j b_j}$.

Hence we aim at showing that for all j, $\beta_j \leq p$, that is, b_j is either to the left or at at most p markers to the right of d_j. The proof for the left bound, that is, to show that a_j is at most p markers to the left of c_j is symmetrical.

By maximality of the strip length (Condition 3), the factors of the strip cannot be extended to the right. Condition 2 is the one constraining the strip length, hence there exists a fragile piece f_{j_0} in the path such that this condition is tight, that is, $d_{j_0} = \mathrm{r}_{\mathrm{ext}}(s)$, where s is the solid piece to the left of f_{j_0}. Hence, a_{j_0} is not to the right of d_{j_0}, and thus $\alpha_{j_0} = \overline{d_{j_0} a_{j_0}} = \overline{\mathrm{r}_{\mathrm{ext}}(s) a_{j_0}} \leq 0$.

Now for all j, $\beta_j - \alpha_j = \|[a_j, b_j]\| - 1$, that is, it is the length of the window contained in f_j minus one. Consequently, $\beta_{j_0} < \|[a_{j_0}, b_{j_0}]\|$. Moreover, for every $1 \leq j < \ell$, either the first markers of the window of f_j and f_{j+1} are matched and thus equidistant to the piece represented by u_i or the last markers of the window of f_j and f_{j+1} are matched and thus equidistant to u_i. Hence, either $\alpha_j = \alpha_{j+1}$ or $\beta_j = \beta_{j+1}$. In the first case, β_{j+1} increases, compared to β_j, by at most $\|[a_{j+1}, b_{j+1}]\| - 1$. Hence, $\beta_j \leq \beta_{j_0} + p$ for all $j \geq j_0$. By symmetry, the same holds for all $j \leq j_0$. □

The following rule serves as a "preparation" of our main rule that deals with cycles containing repetitive vertices. It will ensure that if there is a cycle containing repetitive vertices, then these repetitive vertices will have the same period.

Frame Rule 4. *If the piece graph contains a rep−rep path between repetitive vertices v_s and v_t with strip $\{I_{f_1}, \ldots, I_{f_\ell}\}$ such that the strip $I_f = [u, v]$ in f is shorter than the period π_s of s, then add the frame $f \cap [u \triangleleft w, v \triangleright w]$ to f.*

Proof. By definition, w is at least the total length of the windows of $\mathcal{P}$. By Proposition 4.11, the endpoints of the window of f thus have distance at most w from I_f. □

Frame Rule 5. *If Frame Rule 4 does not apply and the piece graph contains a simple cycle with repetitive vertices, then do the following. Let $\|\pi\|$ be the length of the period of any repetitive solid piece in the cycle. Then, create one branch for each edge $\{v_f, u_s\}$ of the cycle where u_s is a solid vertex for the solid piece s. In each branch, add to Φ the frame*

- *$f \cap [\mathrm{r}_{\mathrm{ext}}(s) \triangleleft (\|\pi\| + w), \mathrm{r}_{\mathrm{ext}}(s) \triangleright w]$ if f is to the right of s, or*
- *$f \cap [\mathrm{l}_{\mathrm{ext}}(s) \triangleleft w, \mathrm{l}_{\mathrm{ext}}(s) \triangleright (\|\pi\| + w)]$ if f is to the left of s.*

Proof. First, all repetitive pieces of the path have the same period. Indeed, consider any two consecutive repetitive pieces s and t of the cycle: they are linked by a rep−rep path, in which we compute the strips. All strips in this path have equal length S and also equal content (Proposition 4.10). Hence, the maximal extensions of repetitive pieces s and t have a common substring of length S. Since Frame Rule 4 does not apply, we have $\|\pi_s\| \leq S$ and $\|\pi_t\| \leq S$. Thus, the maximum extensions of s and t contain a common substring longer than their respective periods. Consequently, their periods are equal, and thus all repetitive pieces of the cycle have the same period π.

Let $s_1, s_2, \ldots, s_\ell$ denote the repetitive pieces crossed successively by the cycle (again, we write $s_{\ell+1} = s_1$). For each i, $1 \leq i \leq \ell$, let $x_i^{\triangleleft}$, $x_i^{\triangleright}$, $y_i^{\triangleleft}$, $y_i^{\triangleright}$ be the fragile pieces to the left and right of s_i in x and s'_i in y, respectively. For each rep−rep path of the cycle from s_i to s_{i+1}, we say the path is *positive* if the first vertex after

s_i is $v_{x_i^\triangleleft}$ or $v_{y_i^\triangleright}$, and *negative* otherwise. In positive rep−rep paths, fragile pieces in x are crossed from right to left (that is, the solid piece to the right of the fragile piece is "seen" before the solid piece to its left), and fragile pieces in y are crossed from left to right. Thus a positive path enters s_{i+1} via either $v_{x_{i+1}^\triangleright}$ or $v_{y_{i+1}^\triangleleft}$, and likewise a negative path enters s_{i+1} via either $v_{x_{i+1}^\triangleleft}$ or $v_{y_{i+1}^\triangleright}$.

First, consider the case that all windows are contained within the strip and that both endpoints of the piece have distance at least $\|\pi\|$ to the borders of the strip. We show that in this case, we can shift all breakpoints in positive paths to the right by $\|\pi\|$ positions and all breakpoints in negative paths to the left by $\|\pi\|$ positions. This is done as follows:

- For each fixed vertex l_s in a positive path, cut out $\|\pi\|$ markers from the left of the blocks containing s and s'.
- For each fixed vertex r_s in a positive path, add $\|\pi\|$ markers from the right of the blocks containing s and s'.
- For each fixed vertex l_s in a negative path, add $\|\pi\|$ markers to the left of the blocks containing s and s'.
- For each fixed vertex r_s in a negative path, cut out $\|\pi\|$ markers from the right of the blocks containing s and s'.
- Replace each short block $[a, b]$ in a fragile piece of a positive path by $[a \triangleright \|\pi\|, b \triangleright \|\pi\|]$.
- Replace each short block $[a, b]$, in a fragile piece of a negative path by $[a \triangleleft \|\pi\|, b \triangleleft \|\pi\|]$.
- For a repetitive vertex v_{s_i} such that the paths before and after v_{s_i} enter and leave v_{s_i} via the same side (either $x_i^\triangleleft$ and $y_i^\triangleleft$, or $x_i^\triangleright$ and $y_i^\triangleright$) either both paths are positive or both paths are negative. Apply the same operation as if the piece was fixed:
  - If the path enters v_{s_i} via $x_i^\triangleleft$ and leaves via $y_i^\triangleleft$, then cut out the $\|\pi\|$ leftmost markers of s and s' if the path is positive or add the $\|\pi\|$ markers to the left of s and s' if the path is negative.
  - If the path enters v_{s_i} via $x_i^\triangleright$ and leaves via $y_i^\triangleright$, then cut out the $\|\pi\|$ rightmost markers of s and s' if the path is negative or add the $\|\pi\|$ markers to the right of s and s' if the path is positive.
- For a repetitive vertex v_{s_i} such that the paths enter and leave the vertex via the same string (either $x_i^\triangleleft$ and $x_i^\triangleright$, or $y_i^\triangleleft$ and $y_i^\triangleright$) it holds that the paths have the same orientation. Apply a similar operation as for a short block (assume wlog. that the path enters and leaves via x):
  - If v_{s_i} is between two positive paths then replace the block $[a, b]$ of x containing s_i by $[a \triangleright \|\pi\|, b \triangleright \|\pi\|]$.
  - If v_{s_i} is between two negative paths then replace the block $[a, b]$ of x containing s_i by $[a \triangleleft \|\pi\|, b \triangleleft \|\pi\|]$.
- For all other repetitive vertices, the paths enter from one string and leave via the other string and enter from one side and leave via the other side. Then the paths have opposite orientations; assume wlog. that the entering path is positive and the outgoing path is negative. Let $[a, b]$ denote the block in x containing s_i, and let $[a', b']$ denote the block in y containing s_i'.
  - If the cycle enters from $y_i^\triangleleft$ and leaves via $x_i^\triangleright$, then replace $[a, b]$ by $[a, b \triangleleft \|\pi\|]$ and $[a', b']$ by $[a' \triangleright \|\pi\|, b']$ ($\|\pi\|$ markers are cut out of both blocks).
  - If the cycle enters from $x_i^\triangleright$ and leaves via $y_i^\triangleleft$, then replace $[a, b]$ by $[a, b \triangleright \|\pi\|]$ and $[a', b']$ by $[a' \triangleleft \|\pi\|, b']$ ($\|\pi\|$ markers are added to both blocks).

Thus, all the breakpoints in fragile pieces have been shifted to the right (in positive paths) or to the left (in negative paths) by a period length $\|\pi\|$. Hence, this modification still gives a partition of both strings. This partition has the same size as the original one. Furthermore, it is also a common string partition which can be seen as follows. The set of strings represented by the short blocks of x and y remains exactly the same since they were shifted by the period length. Hence, there is a matching for the short blocks such that each short block is matched to one representing the same string. For the long blocks, the old matching remains a valid matching: The blocks containing fixed solid pieces have both been modified on the same side. Thus, they are either shortened by $\|\pi\|$ markers; in this case, the matched blocks clearly represent equivalent strings. Or $\|\pi\|$ markers have been added on one side. In this case, the matched strings are also equivalent, since the windows have distance at least $\|\pi\|$ to the borders of the strip. The blocks containing repetitive pieces have either been moved by $\|\pi\|$ positions, shortened by $\|\pi\|$ markers on the same side, $\|\pi\|$ markers on the same side have been added, or they have been shortened or extended on different sides. In the first three cases, the strings represented by the new blocks remain equivalent for the same reasons as for the blocks containing fixed solid pieces. It remains to show the case in which blocks have been modified on different sides.

First, consider the case in which $[a, b]$ is replaced by $[a, b \triangleleft \|\pi\|]$ and $[a', b']$ by $[a' \triangleright \|\pi\|, b']$. Since the blocks are periodic with period length $\|\pi\|$ we have $[a' \triangleright \|\pi\|, b'] \equiv [a', b' \triangleleft \|\pi\|]$. In the old solution, this subfactor of $[a', b']$ was matched with $[a, b \triangleleft \|\pi\|]$, and thus $[a' \triangleright \|\pi\|, b'] \equiv [a', b' \triangleleft \|\pi\|] \equiv [a, b \triangleleft \|\pi\|]$.

Now consider the case in which $[a, b]$ is replaced by $[a, b \triangleright \|\pi\|]$ and $[a', b']$ by $[a' \triangleleft \|\pi\|, b']$. Since the blocks are periodic with period length $\|\pi\|$ we have $[a', b'] \equiv [a' \triangleleft \|\pi\|, b' \triangleleft \|\pi\|]$. Since $[a, b] \equiv [a', b']$ this implies that the first $\|[a, b]\|$ markers of the new blocks are equivalent. Also because of the periodicity, we have $[b, b \triangleright \|\pi\|] \equiv [b \triangleleft \|\pi\|, b]$. Since $[b \triangleleft \|\pi\|, b] \equiv [b' \triangleleft \|\pi\|, b']$, this implies that also the last $\|\pi\|$ markers of the new blocks are equivalent.

Altogether, the modification gives a CSP of the same size, in which the distance between the window endpoints and the strip endpoints has decreased. The above operation can be repeated until at least one breakpoint is at distance less than $\|\pi\|$ from the border of a strip. In this case, all breakpoints of the corresponding path are at distance at most $w + \|\pi\|$ from the border of their corresponding strip (an argument similar to the proof of Proposition 4.11 applies). In some fragile piece f, the border of I_f coincides with the maximum extension of an adjacent solid piece s, thus, in f, the window is contained in either $[\mathrm{l}_{\mathrm{ext}}(s) \triangleleft (\|\pi\| + w), \mathrm{l}_{\mathrm{ext}}(s) \triangleright w]$ or $[\mathrm{r}_{\mathrm{ext}}(s) \triangleleft w, \mathrm{r}_{\mathrm{ext}}(s) \triangleright (\|\pi\| + w)]$. Since in one of the considered branches, the rule adds the frame to this piece s and to the correct side of the strip factor it is correct. Note that the modifications made in the proof do not increase the length of any short block. Hence, the second requirement for correctness is also satisfied. □

The final case that needs to be considered is the one in which the piece graph is acyclic but none of the other rules applies. Then, the piece graph contains a repetitive degree-one vertex.

Frame Rule 6. *If the piece graph contains an edge $\{v_s, v_f\}$ such that v_s is repetitive and has degree one, then assume wlog. that f is to the right of s in x, and do the following. Let $[a_l, a_r]$, $[b_l, b_r]$, and $[c_l, c_l]$ be the (possibly phantom) frames such that $[a_l, a_r]$ is to the left of s' in y, that $[b_l, b_r]$ is to the right of s' in y, and that $[c_l, c_r]$*

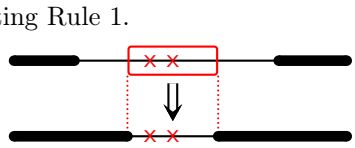


Figure 4.9: An illustration of Fitting Rule 1.

is to the left of s in x. Add the frame $f \cap [f_l, f_r]$ to f, where $f_l = c_l \triangleright (\overline{a_r b_l} + 1)$ and $f_r = c_r \triangleright (\overline{a_l b_r} + w - 2)$.

Proof. The windows to the left and right of s' in y are contained in $[a_l, a_r]$ and $[b_l, b_r]$ respectively, and the windows to the left of s in x is contained in $[c_l, c_r]$. Consider the blocks containing s and s', and let ℓ be their length. The two endpoints of the block containing s' are in $[a_l \triangleright 1, a_r]$ and $[b_l, b_r \triangleleft 1]$. Hence $\ell \geq \overline{a_r b_l}$ and $\ell \leq \overline{(a_l \triangleright 1)(b_r \triangleleft 1)} = \overline{a_l b_r} - 2$.

The leftmost marker of the block containing s is contained in $[c_l \triangleright 1, c_r]$. Thus, the rightmost marker (the one in f) is necessarily in $[c_l \triangleright (\ell + 1), c_r \triangleright (\ell)]$ which, by the above upper and lower bounds on ℓ, is contained in $[c_l \triangleright (\overline{a_r b_l} + 1), c_r \triangleright (\overline{a_l b_r} - 2)]$. This marker is the leftmost marker of the window of f which has length at most w. Hence the frame $[c_l \triangleright (\overline{a_r b_l} + 1), c_r \triangleright (\overline{a_l b_r} + w - 2)]$ contains the window of f. The rule is still correct if s or s' corresponds to the end of a string, since the phantom frames contain the leftmost or rightmost marker of the blocks containing s or s'. □

After exhaustively applying the frame rules, the parts of fragile pieces that are outside of frames do not contain a breakpoint. Hence, we perform the following rule which shrinks fragile pieces such that they fit their frame; at the same time, the solid pieces are extended accordingly.

Fitting Rule 1. *If there is a fragile piece $f = [a, b]$ with frame $[c, d] \subset [c, d]$ then add $[a, c]$ to the solid piece left of f, add $[d, b]$ to the solid piece right of f, and set $f = [c, d]$.*

We now show two important properties of instances for which none of the frame rules applies. First, every fragile piece of these instances has a frame. Second, the frame lengths are upper-bounded by a function of k, β, and the longest period of any repetitive piece.

Lemma 4.12. *Let $\mathcal{C}$ be a constraint with frame set ϕ such that none of the Frame Rules 1–6 applies. Then, each fragile piece has a frame, and all frames have length at most $(6k^2w + 3kw + 3k \max\{w, \|\pi\|\})$, where $\|\pi\|$ denotes the length of the longest period of all repetitive solid pieces.*

Proof. First, we show that every fragile piece has a frame. If the piece graph contains a cycle, then either Frame Rule 3, 4, or 5 applies. Otherwise, the piece graph is acyclic, and thus it either contains a degree-one vertex and one of the other Frame Rules applies, or all vertices have degree zero which means that all fragile pieces have frames.

Next, we show the upper bound on the frame length. Let L be the length of the longest frame created in this procedure, and let π be the longest period over all repetitive pieces. We show that

$$L \leq 6k^2w + 3kw + 3k \max\{w, \|\pi\|\} \tag{4.2}$$

Let h be the number of frames created before Frame Rule 6 is first applied, $1 \leq h \leq 2k$. Frame Rules 1, 3, 4 and 5 produce frames of length at most $(\max\{w, \|\pi\|\}+2w)$. Since each application of Frame Rule 2 increases the maximum frame length by w, all frames have length at most $(\max\{w, \|\pi\|\} + (h+1)w)$ before the first application of Frame Rule 6. Note that once Frame Rule 6 is applied for the first time, only Frame Rules 2 and 6 can be applied. We introduce the following notations. A solid vertex (fixed or repetitive) is *closed* if all its adjacent fragile pieces have frames, and *open* otherwise. The *weight* of an open vertex is the total length of the frames in the adjacent fragile pieces. Let W denote the sum of the weights of all open vertices.

Before the first application of Frame Rule 6, $W \leq 3k[\max\{w, \|\pi\|\} + (h+1)w]$ (for each solid piece $s \in S_x$, the weight of either v_s or l_s and r_s together is at most the sum of the weights of three different frames). Afterwards, each time Frame Rule 2 or 6 is applied, an open vertex with some weight u is closed, and a frame of length $u + w$ is created in a fragile piece f which is adjacent to at most one open vertex. Thus, the total weight of open vertices W is increased by at most $u + w - u = w$ with each application of Frame Rule 2 or 6. They are applied at most $2k - h$ times, hence W is at most

$$\begin{aligned} W &\leq 3k\,[\max\{w, \|\pi\|\} + (h+1)w] + (2k-h)w \\ &\leq 6k^2w + 3kw + 3k\max\{w, \|\pi\|\}. \end{aligned}$$

Since no frame of length more than W can be created, we have $L \leq W$, which proves the second part of the claim. □

The bound given by the lemma above still contains the maximum period length π which means that it is too large to be useful for the `split` procedure. However, the algorithm can now either find a repetitive piece which can be fixed with few options (see Lemma 4.13) or the maximum period length is not too long.

Lemma 4.13. *Let $\mathcal{C}$ be a constraint that contains a repetitive solid piece s with period π_s such that each fragile piece adjacent to s or s' has length at most $6(k^2 + k)\,\|\pi_s\|$. Then, there are at most $12(k^2 + k)$ feasible alignments, and any CSP satisfying $\mathcal{C}$ matches elements of s according to a feasible alignment.*

Proof. The alignment corresponding to any CSP satisfying $\mathcal{C}$ is necessarily feasible, since otherwise two distinct solid pieces would be contained in the same block.

Wlog., let $\|s\| \geq \|s'\|$. Thus, in a satisfying CSP $\mathcal{P}$, either the leftmost marker of s is matched to a marker left of s' (or to the leftmost marker of s'), either the rightmost marker of s is matched to a marker right of s'. Consider the first case; by Condition 2 of satisfying CSPs, the leftmost marker of s is matched to some marker in the fragile piece to the left of s'. Note that since s and s' have a shortest period π_s, two different alignments are separated by a a multiple of $\|\pi_s\|$ markers. Hence, there are at most $6(k^2+k)$ different alignments in which the leftmost marker of s is matched to some marker of the fragile piece to the left of s'. Similarly, there are at most $6(k^2 + k)$ possible alignments in which the rightmost marker of s is matched to a marker of the fragile piece to the left of s' (for $\overline{s} = \overline{s'}$, it is possible that both left and right endpoints of s are matched to markers of the fragile pieces to the left and right of s'). Overall, the total number of alignments between s and s' thus is at most $12(k^2 + k)$. □

By guessing the alignments of the long periods we have finally achieved the goal of `frames`: all frames are "short" enough to be split by `split`.

Lemma 4.5 (cf. Page 96). When `frames` terminates, every fragile piece has length at most $12(k^2+k)k\beta$.

Proof. By Lemma 4.12, an instance in which no frame rule applies has frames of length at most $(6k^2w+3kw+3k\max\{w,\|\pi\|\})$ where π is the longest period among all repetitive pieces. In case $\|\pi\| \geq w$, then for each repetitive piece s with period π, there are at most $6(k^2+k)\cdot\|\pi\|$ possibilities to align s. Hence, at least one repetitive piece is fixed in the loop Lines 8–11, and `new-align` is set "True", which means that the outer loop in `frames` will be repeated. Otherwise, $\pi \leq w$ and thus $6k^2w+3kw+3k\|\pi\| < 6(k^2+k)w \leq 12(k^2+k)k\beta$. □

The correctness of `frames` is simply a consequence of the correctness of all single steps (always considering the correct branching in each branching step).

Lemma 4.3 (cf. Page 96). If there exists a size-k CSP $\mathcal{P}$ satisfying $\mathcal{C}$ at the beginning of `frames` such that longest undiscovered block is β-critical, then `frames` creates at least one branch such that the constraint in this branch is satisfied by a size-k CSP $\mathcal{P}'$ whose longest undiscovered block has length at most $2\beta-1$.

Proof. The correctness of all frame rules have already been proven. The correctness of Fitting Rule 1 is trivial. Finally, the correctness of Lines 8–11 follows simply from the fact that the alignment in one of the branches is the correct one (it considers all feasible alignments). Since the correctness definition of the frame rules demands that all undiscovered blocks are at most as long as before adding the frame, also the size bound for the longest undiscovered block holds. □

It thus remains to bound the running time of `frames`. In particular, we need to show that the number of branches is bounded by a function of k.

Lemma 4.4 (cf. Page 96). Overall, the calls to `frames` create $(4k)^{4k^2}\cdot k^{O(k)}$ branches; all other parts of the algorithm can be performed in $\mathrm{poly}(n)$ time.

Proof. First, note that the outer repeat-until loop of `frames` is repeated at most $2k$ times over the course of *all* calls to `frames`: The procedure `frames` is called at most k times from the main method. Each additional time the repeat-until loop is repeated, there is a pair of repetitive solid pieces that becomes a pair of fixed solid pieces at Line 10 of the previous pass of the repeat-until loop. This can happen at most k times.

Second, note that the while loop of Lines 4–5 is iterated at most $2k$ times in each repetition of the other repeat-until loop of `frames`: each rule creates exactly one frame, and, by Observation 4.2 there are at most $2k-2$ fragile pieces.

Hence, there are at most $4k^2$ times in which one of the frame rules at Line 5 creates branches and at most k times in which branches are created at Line 10. The only frame rules that perform branchings are Frame Rules 3 and 5. In both cases, the rule branches into at most $2k$ cases, since each cycle has at most $2k$ solid vertices and thus at most $4k$ edges in the cycle under consideration. Hence, the branchings performed by the frame rules increase the running time by a factor of $O((4k)^{4k^2})$. Each of the at most k branchings in Line 10 is among at most $(12k)^2+k$ choices (Lemma 4.13). Hence, these branchings increase the running time by a factor of $O((12k)^{2k}\cdot k^k)$. Hence, the overall increase due to the branching is by a factor of $(4k)^{4k^2}\cdot k^{O(k)}$; all other steps can be performed in polynomial time. □

Conclusion

We have presented the first fixed-parameter algorithm for Minimum Common String Partition (MCSP) parameterized by the partition size k. Clearly, an improvement of the so far very impractical running time is desirable; the bottleneck appears to be that some of the frame rules still have to branch. Furthermore, it remains open whether MCSP admits a polynomial problem kernel for this parameter. Another important open problem is to design constant ratio approximation algorithms.

The main drawback of the MCSP problem is that instances can be too constrained for practical applications, in particular because of the constraint imposing that both strings use exactly the same multi-set of markers. We list below four possible extensions of the problem, more fit for practical applications, which might be solvable using our framework.

Could the algorithm be extended so that ...

A. it admits signed strings? That is, each marker is annotated with one direction and blocks can be reversed before matching. See the signed variant(s) of MCSP [45, 69, 120].

B. it admits three or more input sequences?

C. it allows for a bounded number ℓ of erroneous markers which can be deleted? These markers could appear either between consecutive blocks (they correspond to genes which do not appear in synteny blocks) or even within one block (they correspond to reconstruction errors).

D. it allows for uncertainties in the periodic parts of input sequences? Indeed, one major difficulty in genome reconstruction is to know exactly the number of consecutive repetitions of periodic factors. Thus, input data, instead of being "`c a b a b a b a c`" could be "`c (a b)`{from 2 to 4 copies}` a c`".

For A. and B., we conjecture that adapting the algorithm would be straightforward, with additional frame rules to take into account the new cases that may appear. Note that for A., we can branch on the orientation of the block containing each solid piece as soon as it is discovered. For B., the major difficulty would be to extend Frame Rules 3 and 5 so that they apply to general connected components instead of cycles.

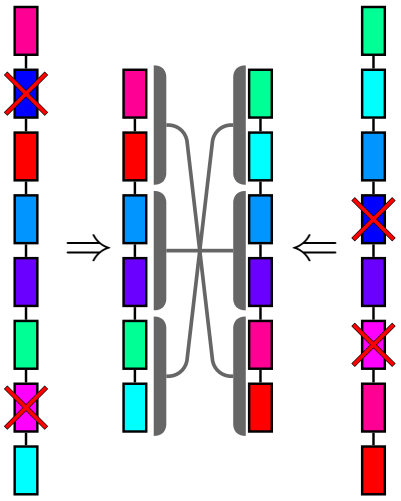
For C. and D. the modifications go deeper and imply to reconsider key concepts such as maximum extension (for C.) and splitting (for D.). But we conjecture that our framework should allow to solve these harder problems.

A final problem, which we raise out of pure theoretical interest, is the bidimensional extension of MCSP: given two images x and y of size $n \times m$ using the same multi-set of pixel colors, is it possible to "partition" these images into k identical rectangles? It is obviously harder than MCSP, but might also be fixed-parameter tractable for parameter k.

Dealing with Imprecise Genomic Data

5

Maximal Strip Recovery



The Maximal Strip Recovery (MSR) problem aims at extracting synteny blocks in genomes where the sequences may contain errors. Formally, some markers must be deleted in order to be able to partition the sequences into common strips of length at least 2, covering a maximum number of markers.

In this chapter, we study a number of variants of MSR, especially the δ-gap variant where at most δ consecutive markers may be deleted, presenting in each case computational complexity results, and approximation or FPT algorithms.

This chapter gathers results from two different articles. The first has been presented at the *20th International Symposium on Algorithms and Computation* (ISAAC 2009, Honolulu [31]), and has been published in the *Journal of Discrete Algorithms* (JDA 2013, [36]). The second, a joint work with Minghui Jiang, has been presented at the *22nd Annual Symposium on Combinatorial Pattern Matching* (CPM 2011, Palermo [29]) and has been published in *Theoretical Computer Science* (TCS 2012 [30]).

Introduction

An essential task in comparative genomics is to decompose two or more genomes into synteny blocks that are segments of chromosomes with similar contents. Synteny blocks represent units of the genomes that have not been disrupted by large-scale rearrangements such as reversals and transpositions, and thus form the input for genome rearrangement algorithms. They also give useful clues regarding the role of each gene, since genes belonging to the same synteny block often produce proteins with related functions. Extracting synteny blocks from genomic maps, however, is a non-trivial task when the genomic maps contain noise and ambiguities, which need to be removed before we can give a precise synteny block decomposition. This motivates the MAXIMAL STRIP RECOVERY problem [128]: to delete a set of markers (genes) from the genomic maps until the remaining markers can be partitioned into a set of strips (synteny blocks) of maximum total length.

We introduce some definitions. A genome consists of one or more chromosomes; each chromosome is a sequence of genes. Correspondingly, a *genomic map* consists of one or more sequences of gene markers. Each marker is a signed integer representing a gene: the absolute value of the integer represents the family of the gene; the sign of the integer represents the orientation. A marker has *duplicates* if it is contained more than once in some genomic map, possibly in different orientations. A *strip* of $d \geq 2$ genomic maps is a sequence of at least two markers appearing consecutively in each map, such that the order of the markers and the orientation of each marker are either both preserved or both reversed. The *reversed opposite* of a sequence $s = \langle x_1, \ldots, x_h \rangle$ is $-s = \langle -x_h, \ldots, -x_1 \rangle$. The MAXIMAL STRIP RECOVERY problem on d input maps is the following maximization problem MSR-d [47, 128] (or MSR, when $d = 2$):

Problem	MSR-d
Input	d genomic maps $\mathcal{M}_{1\ldots d}$ [1] each containing the same n markers without duplicates
Output	d subsequences $\mathcal{M}'_{1\ldots d}$ of $\mathcal{M}_{1\ldots d}$ respectively, each containing the same ℓ markers, such that all the markers in $\mathcal{M}'_{1\ldots d}$ can be partitioned into strips
Maximize	the number ℓ of selected markers

Example 5.1. *Consider the following three genomic maps.*

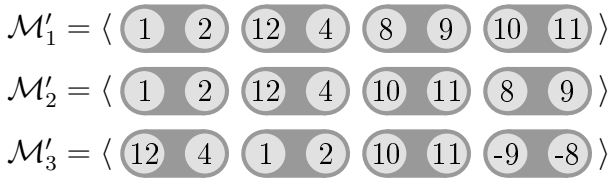
$\mathcal{M}_1 = \langle$ 1 2 3 12 4 5 8 9 10 6 7 11 $\rangle$

$\mathcal{M}_2 = \langle$ 1 -7 -6 2 12 3 4 10 5 11 8 9 $\rangle$

$\mathcal{M}_3 = \langle$ 12 4 1 5 2 3 10 11 6 -9 7 -8 $\rangle$

Then an optimal solution of MSR-3 on ($\mathcal{M}_1$, $\mathcal{M}_2$, $\mathcal{M}_3$) consists of the three subsequences of length $\ell = 8$ below. They can be partitioned into the set of strips $\{\langle 1, 2\rangle, \langle 12, 4\rangle, \langle 10, 11\rangle, \langle 8, 9\rangle\}$. The subsequences corresponding to each strip are highlighted both in the original maps and in the solution.

1. For convenience, we abbreviate the notation for a list of d genomic maps $(\mathcal{M}_1, \ldots, \mathcal{M}_d)$ to $\mathcal{M}_{1\ldots d}$ and $(\mathcal{M}'_1, \ldots, \mathcal{M}'_d)$ to $\mathcal{M}'_{1\ldots d}$



The maximization problem MSR-d has a *complement* minimization problem called CMSR-d [122, 95] that minimizes the parameter $k = n - \ell$, the number of deleted markers. For genomic maps of close species with few errors, k can be much smaller than ℓ, thus FPT and approximation algorithms are sometimes more relevant for CMSR than for MSR.

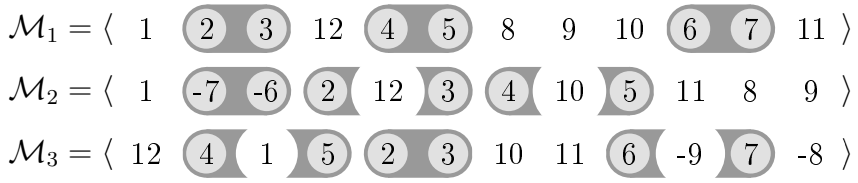
For the more general case where input maps can have duplicate elements (that is, the genes belong to *paralogy families*), the following variant has been introduced [47]:

Problem	MSR-DU-d
Input	d genomic maps $\mathcal{M}_{1\ldots d}$ each containing n markers, possibly with duplicates
Output	d subsequences $\mathcal{M}'_{1\ldots d}$ of $\mathcal{M}_{1\ldots d}$ respectively, each containing the same ℓ markers, such that all the markers in $\mathcal{M}'_{1\ldots d}$ can be partitioned into strips
Maximize	the number ℓ of selected markers

Given d subsequences $\mathcal{M}'_{1\ldots d}$ of d genomic maps $\mathcal{M}_{1\ldots d}$, respectively, the *gap* between two consecutive markers a and b of $\mathcal{M}'_i$ is the number of markers appearing between a and b in $\mathcal{M}_i$, a and b excluded. The *gap* of a strip s is the maximum gap between any two consecutive markers of s in any map $\mathcal{M}'_i$. In Example 5.1, the strip $\langle 1, 2\rangle$ in the given solution has gap 2, because markers 1 and 2 are separated by markers 6 and 7 in $\mathcal{M}_2$. The deleted markers between markers of a strip correspond to noise and ambiguities, which occur infrequently. From a biological point of view, a synteny block is a segment of chromosomes that remain undisrupted by genome rearrangements during evolution. Hence, consecutive elements of a synteny block can only be separated in a data set due to noise and ambiguities, and a strip having a large gap is unlikely to correspond to a synteny block. This leads to the following gap-constrained variant of MSR-d which we introduced formally in [31] (related experimental questions appear in [128]):

Problem	δ-gap-MSR-d
Input	d genomic maps $\mathcal{M}_{1\ldots d}$ each containing the same n markers without duplicates
Output	d subsequences $\mathcal{M}'_{1\ldots d}$ of $\mathcal{M}_{1\ldots d}$ respectively, each containing the same ℓ markers, such that the markers in $\mathcal{M}'_{1\ldots d}$ can be partitioned into strips, and such that each strip has gap at most δ
Maximize	the number ℓ of selected markers

In Example 5.1, the strips in the given subsequences $(\mathcal{M}'_1, \mathcal{M}'_2, \mathcal{M}'_3)$ have gap at most 2, hence they are an optimal solution of δ-gap-MSR-3 for $\delta = 2$. For $\delta = 1$, the optimal solution contains only 6 markers in 3 strips: $\{\langle 2, 3\rangle, \langle 4, 5\rangle, \langle 6, 7\rangle\}$. They are highlighted below.



No doubt that MSR-d (which can be written ∞-gap-MSR-d) is a more elegant problem from a theoretical perspective, but δ-gap-MSR-d could be more relevant in biological applications, especially since it sometimes allows for more efficient algorithms. The gap-constrained variants of MSR-DU-d and CMSR-d, denoted respectively δ-gap-MSR-DU-d and δ-gap-CMSR-d, can be similarly defined. Similarly to MSR-d and CMSR-d, the parameter for δ-gap-MSR-d is ℓ, and the parameter for δ-gap-CMSR-d is k. In most cases, δ and d are assumed to be constants, although our FPT algorithms in Section 5.4 do not depend on this assumption and can take δ and d as parameters besides ℓ or k. There is no known direct reduction from δ-gap-MSR-d to MSR-d or vice versa. Although the gap constraint appears to be an additional burden that the algorithm has to take care of, it also limits the set of candidate strips and their intersection pattern, especially when δ is small, which may make the problem easier to handle.

The earliest results on MSR are two heuristics [53, 128], the NP-hardness of MSR-3 [47], and a 4-approximation algorithm for MSR [47] (which can be extended to a $2d$-approximation for MSR-d and δ-gap-MSR-d, see [95] and Section 5.3.1). Parallel to our work, several hardness results have been obtained for the variants of Maximal Strip Recovery [123, 95, 94], and a variety of algorithms have been developed, including approximation [129, 89, 102] and FPT algorithms [123, 129, 89]. The tightest hardness results regarding these problems are summarized as follows. MSR-d, CMSR-d and MSR-DU-d are APX-hard for any $d \geq 2$, even in the δ-gap variant, for $\delta \geq 2$ [95]; MSR-d is W[1]-hard for any $d \geq 4$ [94]. Finally, along some very recent development [123, 129, 89, 102, 90] on the CMSR problem parallel to our work, Lin et al. [102] presented a 2.667-approximation algorithm for CMSR-2, which is based on an interesting idea called local amortization with re-weighting. From a parameterized point of view, Jiang et al. [89] presented an FPT algorithm for CMSR-2 running in time $O(3^k \operatorname{poly}(n))$, and Jiang & Zhu [90] obtained a linear kernel for CMSR-2 which leads (using the algorithm we presented in [29]) to an algorithm running in time $O(2.36^k k^2 + n^2)$.

In this chapter, we present a panel of new or improved FPT and approximation algorithms for the many variants of Maximal Strip Recovery. The current best results, including our contribution, are summarized in Table 5.1. The organization of the chapter is as follows. In Section 5.1, we introduce some prerequisites. In Section 5.2, we present hardness results, with the NP-hardness of 1-gap-MSR (and 1-gap-CMSR), and the APX-hardness of 2-gap-MSR and 0-gap-MSR-DU. We then give polynomial-time algorithms in Section 5.3: first a reduction to Maximum Weight Independent Set in 5.3.1, which implies a general $2d$ approximation for variants of MSR, and an exact algorithm for 0-gap-MSR. Then, we give in Sections 5.3.2 to 5.3.5 a panel of algorithms for δ-gap constrained problems with small values of δ, relying on existing work on claw-free graphs. Finally, we give in Section 5.3.6 an approximation for δ-gap-CMSR-d and CMSR-d which is valid for any values of δ and d. In the final part of our study of Maximal Strip Recovery we present three fixed-parameter tractable algorithms in order to solve exactly the problems δ-gap-MSR-d (Section 5.4.1), δ-gap-CMSR-d (Section 5.4.2) and 1-gap-

Table 5.1: Algorithmic results for variants of MSR. Variable d represents the number of input genomic maps ($d = 2$ for MSR, MSR-DU and CMSR). A reference to the relevant section of the chapter and/or article is given for each result. Note that we omit MSR-DU-d with $d \geq 3$ due to the lack of results specific to this problem.

δ-gap constraint	**Variant of the problem**				
	MSR	MSR-d $d\geq3$	MSR-DU	CMSR	CMSR-d $d\geq3$
	Complexity class of the decision problem				
$\delta = 0$	P 5.3.1			P 5.3.1	
$\delta = 1$	NP-complete 5.2.2		APX-complete	NP-complete 5.2.2	
$\delta = 2$	APX-complete			APX-complete	
$\delta \geq 3$	5.2.3 and [95]		5.2.4		
no	W[1]-hard ($d \geq 4$) and APX-complete [94, 95, 123]			[95, 123]	
	Ratio of approximation algorithm				
$\delta = 0$	P 5.3.1		2.25 5.3.5	P 5.3.1	
$\delta = 1$	1.8 5.3.3	$0.75d+0.75+\epsilon$		2.778 5.3.4	
$\delta = 2$	$1.5d + \epsilon$				
$\delta = 3$	$1.5d + 0.75 + \epsilon$ 5.3.2			$d + 1.5$	
$\delta \geq 4$	$2d$				
no			5.3.1 and [47, 95]	2.667 [102]	5.3.6
	Running-time of FPT algorithm				
$\delta = 0$	P			P	
$\delta = 1$	$O(2^t td\delta^2 + nd\delta)$			$O(2^k \text{ poly}(nd))$ 5.4.3	
$\delta = 2$	with $t = \ell(1 + \frac{3}{2}d\delta)$		?		
$\delta \geq 3$	5.4.1			$O(2.36^k \text{ poly}(nd))$	
no	? (W[1]-hard for $d \geq 4$ [94])			Linear kernel [90]	5.4.2

CMSR-d (Section 5.4.3).

5.1 Preliminaries

We first present in this section the different concepts and notations used in our algorithms. Since we explore a variety of different approaches, a large number of concepts are introduced. We then give a summary of graph theory definitions and results, mostly related to the Maximum Independent Set problem, which are used for reductions in algorithms and hardness proofs.

5.1.1 Notations and Definitions

We assume wlog. that all markers in the first input map $\mathcal{M}_1$ have positive sign. Unless explicitly noted, our uses of some standard terms such as *solution* and *optimal solution*, our previous definitions of *strip* and *gap*, as well as other definitions that we will introduce in this section — all these apply to some "current" set of genomic maps $\mathcal{M}'_{1\ldots d}$ implicit from context, which can be either the set of original maps $\mathcal{M}_{1\ldots d}$ given as input, or some set of reduced maps (subsequences obtained from $\mathcal{M}_{1\ldots d}$ by deleting some markers) during or after the execution of a recursive algorithm.

If a sequence of markers forms a strip in some maps $\mathcal{M}'_{1\ldots d}$, then these markers are either all selected or all deleted in *any* optimal solution for these maps. This is because any solution that includes only a subset of the markers in a strip can be extended to a better solution to include all markers in that strip. Hence such a maximal sequence of markers is treated as an atomic unit, and called a *super-marker*, whose *size* is the number of markers it contains. Note that the size of a super-marker is always at least 2. A marker that does not belong to any super-marker is a *single-marker*. We use the term *single-super-marker* to refer to either a single-marker or a super-marker. A common step in several of our algorithms is to partition the markers in the current maps into single-super-markers. If the current maps contain only super-markers, then we have a straightforward decomposition into strips, without deleting any (other) marker.

Let x and y be two single-super-markers of some maps $\mathcal{M}'_{1\ldots d}$. We write $S^i(x, y)$ for the set of markers appearing between x and y in map $\mathcal{M}'_i$. We say that y *follows* x in map $\mathcal{M}'_i$ if one of $\langle +x, +y\rangle$, $\langle -y, -x\rangle$ is a subsequence of $\mathcal{M}'_i$. For δ-gap constrained problems (δ-gap-MSR-d and δ-gap-CMSR-d), we add the constraint that the number of markers appearing between x and y in the original maps is at most δ (x and y excluded). For multichromosomal genomes, that is, when a genomic map consists of several sequences of gene markers (several chromosomes), we require that x and y belong to the same chromosome in $\mathcal{M}'_i$. We define the relation "y *precedes* x" symmetrically.

We say that y is a *candidate successor* of x, and we write $x \prec y$, if y follows x in all maps, and if there is no other y' such that for all $1 \leq i \leq d$, y' follows x in $\mathcal{M}'_i$ and appears in $S^i(x, y)$. We define symmetrically the candidate predecessors, hence y is a candidate successor of x iff x is a candidate predecessor of y. If y is a candidate successor or predecessor of x, $\mathsf{gap}(x, y)$ is the set of all markers appearing in $S^i(x, y)$ for some $1 \leq i \leq d$. In Example 5.1, we have $1 \prec 2$, and $\mathsf{gap}(1, 2) = \mathsf{gap}(2, 1) = \{5, 6, 7\}$.

A *prestrip* is a sequence $\sigma = \langle \sigma_1, \sigma_2, \ldots, \sigma_h \rangle$ such that $\sigma_1 \prec \sigma_2 \prec \ldots \prec \sigma_h$ and $h \geq 2$ (equivalently, a prestrip is a common subsequence that satisfies the "current" δ-gap constraint; we use the notation *δ-prestrip* in case of ambiguities). For any map $\mathcal{M}$, we write $\text{idx}(\sigma, \mathcal{M})$ for the vector $(i_1, \ldots, i_h)$ such that $\sigma_k = \mathcal{M}[i_k]$ for all $1 \leq k \leq h$. The *length* $|\sigma|$ of σ is h. Two prestrips σ and τ are *non-overlapping* if, in each $\mathcal{M}_i$, one appears strictly before the other (there is no i such that an element of $\text{idx}(\sigma, \mathcal{M}_i)$ is between the first and last elements of $\text{idx}(\tau, \mathcal{M}_i)$). The *reversed opposite* of $\langle \sigma_1, \ldots, \sigma_h \rangle$ is $\langle -\sigma_h, -\sigma_{h-1}, \ldots, -\sigma_1 \rangle$. A *sub-prestrip* σ' of a prestrip σ is a prestrip such that σ' is a subsequence of σ.

A set of prestrips $\mathcal{S}$ is said to be *feasible* if it contains pairwise non-overlapping prestrips, and we write $||\mathcal{S}||$ for its *total size*: $||\mathcal{S}|| = \sum_{\sigma \in \mathcal{S}} |\sigma|$. With this formalism, we can see that the strips of any solution correspond to a feasible set of prestrips in the input genomes, thus the objective of the MSR problem can be seen as finding a feasible set $\mathcal{S}$ of prestrips such that $||\mathcal{S}|| = \ell$ is maximum.

We call *peg marker*, and we write $\times$, a marker that cannot belong to any strip of an optimal solution (its role is to affect the gap of other prestrips). A sequence of h consecutive peg markers is written $\times^h$.

The following lemma gives some basic properties of the function gap:

Lemma 5.1. (a) *Let u, v, w be three markers or single-super-markers. If u and v are two candidate successors of w with $u \neq v$, then $u \in \mathsf{gap}(w, v)$ and $v \in \mathsf{gap}(w, u)$.* (b) *Let u and v be two single-super-markers. If $u \prec v$ or $u \succ v$, then $\mathsf{gap}(u, v)$ is not empty.*

Proof. (a) By definition of a candidate successor, there exists some i such that $v \notin S^i(w, u)$, and some j such that $u \notin S^j(w, v)$. Assume wlog. that w has positive sign in $\mathcal{M}_i$. Then there exist sequences $s_{w,u}$ and $s_{w,v}$, using markers of $S^i(w, u)$ and $S^i(w, v)$ respectively, such that both $w\ s_{w,u}\ u$ and $w\ s_{w,v}\ v$ appear in map $\mathcal{M}_i$. We now compare $|S^i(w, u)|$ and $|S^i(w, v)|$:

- if $|S^i(w, u)| = |S^i(w, v)|$, then $s_{w,u} = s_{w,v}$ and $u = v$ (this case is impossible),
- if $|S^i(w, u)| < |S^i(w, v)|$, then $s_{w,u}\ u$ is a prefix of $s_{w,v}$, and $u \in S^i(w, v) \subseteq \mathsf{gap}(w, v)$,
- if $|S^i(w, u)| > |S^i(w, v)|$, then $s_{w,v}\ v$ is a prefix of $s_{w,u}$ and $v \in S^i(w, u)$ (this case is impossible).

Likewise, using map $\mathcal{M}_j$, we have $v \in S^j(w, u) \subseteq \mathsf{gap}(w, u)$. This proves the first property.

(b) Assume wlog. that $u \prec v$ (the other case $u \succ v$ is symmetric). If $\mathsf{gap}(u, v) = \emptyset$, then for all i, $S^i(u, v) = \emptyset$, and hence either $\langle +u, +v \rangle$ or $\langle -v, -u \rangle$ appears in map $\mathcal{M}_i$. It follows that $\langle u, v \rangle$ could form a super-marker: a contradiction. □

5.1.2 Graph Theory Background

In this chapter, the majority of hardness results and polynomial-time algorithms we present for the variants of MSR use reductions from/to restrictions of the Maximum Independent Set problem. We thus make use of a number of algorithms and hardness results from the literature which we list below, after recalling some definitions. Unless noted otherwise, all considered graphs are unoriented.

Maximum Independent Set

Given a graph $G = (V, E)$, a set $X \subset V$ is said to be *independent* if for every edge $(u, v) \in E$, $u \notin X$ or $v \notin X$. The (optimization) problem of computing an independent set of maximum cardinality is defined as follows.

Problem	Maximum Independent Set (MIS)
Input	A graph $G = (V, E)$
Output	An independent set X of G
Maximize	$\|X\|$

The cardinality of a maximum independent set of G is written $\alpha(G)$. In the decision formulation of Maximum Independent Set, one has to decide, given a graph G and an integer k, whether $\alpha(G) \geq k$.

Cubic Graphs

A graph $G = (V, E)$ is *cubic* if every vertex $u \in V$ has degree exactly 3. For every cubic graph, $|E| = \frac{3}{2}|V|$. We call 3-MIS the variant of Maximum Independent Set restricted to cubic graphs, defined below.

Problem	3-MIS
Input	A cubic graph $G = (V, E)$
Output	An independent set X of G
Maximize	$\|X\|$

The 3-MIS problem is APX-hard [3], and, more precisely, NP-hard to approximate within 95/94 [52].

Interval and d-Interval Graphs

A *d-interval graph*, for $d \geq 1$, is a graph $G = (V, E)$, where every vertex in V is seen as a list of d disjoint intervals $(I_1, \ldots, I_d)$ of $\mathbb{R}$ (also called a *d-interval*), and such that two distinct d-intervals $(I_1, \ldots, I_d)$ and $(J_1, \ldots, J_d)$ form an edge in E iff $(\bigcup_{h=1}^{d} I_h) \cap (\bigcup_{h=1}^{d} J_h) \neq \emptyset$. For $d = 1$, such a graph is simply called an *interval graph*.

Problem	d-Interval-MWIS
Input	A d-interval graph $G = (V, E)$, a weight function $w : V \to \mathbb{R}^+$
Output	An independent set X of G
Maximize	The total weight of X, $\sum_{x \in X} w(x)$

The problem 1-Interval-MWIS (known as Interval-MWIS) is polynomial [78]. On the other hand, d-Interval-MWIS is APX-hard for $d \geq 2$, and a $2d$-approximation is known for it [15].

Claw-Free and k-Claw-Free Graphs

Given a graph $G = (V, E)$, a *k-claw* ($k \geq 3$) is a subgraph of G of $k + 1$ vertices $u, v_1, \ldots, v_k$, such that $(u, v_i) \in E$ for all i, but $(v_i, v_j) \notin E$ for all i, j. A *k-claw-free graph* (or *claw-free graph*, for $k = 3$) is a graph that has no k-claw as subgraph. We are interested in the following two variants of the Maximum Independent Set problem.

Problem	Claw-Free-MWIS
Input	A claw-free graph $G = (V, E)$, a weight function $w : V \to \mathbb{R}^+$
Output	An independent set X of G
Maximize	The total weight of X, $\sum_{x \in X} w(x)$

Problem	k-Claw-Free-MIS
Input	A k-claw-free graph $G = (V, E)$
Output	An independent set X of G
Maximize	$\lvert X \rvert$

The problem Claw-Free-MWIS is in P [105], and, for $k \geq 4$, k-Claw-Free-MIS is approximable within $\frac{k-1}{2} + \epsilon$ for any $\epsilon > 0$ [81].

Cubic, Planar, 2-Connected Graphs and 3-edge-colorings

A graph is *planar* if it can be embedded on the plan without having crossing edges. It is *2-connected* if it is connected, and cannot be disconnected by removing less than two vertices. We consider the following decision problem focused on the class of graphs which are at the same time cubic, planar and 2-connected.

Problem	CP2C-MIS
Input	A cubic, planar, 2-connected graph $G = (V, E)$, an integer k
Question	Is $\alpha(G) \geq k$?

The problem CP2C-MIS is NP-hard since Maximum Independent Set is the complement of Vertex Cover, and Vertex Cover is known to be NP-hard on this class of graphs [21].

This class is particularly related to the notion of 3-edge-coloring (also known as Tait Coloring): a *3-edge-coloring* of a cubic graph $G = (V, E)$ is a partition of its edges in three disjoint sets $E = E^{\mathrm{A}} \cup E^{\mathrm{B}} \cup E^{\mathrm{C}}$ such that if two edges $e_1, e_2 \in E$ are incident to a common vertex, they belong to different sets (they have different *colors*). Note that if a cubic graph with n vertices admits a 3-edge-coloring, then each set contains $n/2$ edges.

Lemma 5.2. *Every cubic planar 2-connected graph admits a 3-edge coloring, and such a coloring can be computed in polynomial time.*

Proof. This construction uses a well-known equivalence between the 4-coloring of a planar graph and the 3-edge-coloring of a cubic graph [22]. Let $G = (V, E)$ be a cubic, planar, and 2-connected graph. The Four Color Theorem [9] ensures that its region graph admits a 4-coloring. We compute such a coloring, with colors taken in the set $\{0, 1, 2, 3\}$, using e.g. the quadratic time algorithm from Robertson et al. [114].

For every edge $e \in E$, let $\phi(e)$ be the pair of colors associated to its two adjacent faces (since the graph is 2-connected, e is adjacent to two different faces which have different colors). We deduce a 3-edge-coloring of G from the following formulas:

- If $\phi(e) = \{0, 1\}$ or $\phi(e) = \{2, 3\}$, give to e the color A.
- If $\phi(e) = \{0, 2\}$ or $\phi(e) = \{1, 3\}$, give to e the color B.
- If $\phi(e) = \{0, 3\}$ or $\phi(e) = \{1, 2\}$, give to e the color C.

If two edges e_1 and e_2 are incident to the same vertex, then, since this vertex has degree 3, there are 3 faces f_0, f_1, f_2 (with 3 different colors) such that e_1 is adjacent to f_0 and f_1, and e_2 is adjacent to f_0 and f_2. So $\phi(e_1) \cap \phi(e_2)$ has size 1, and e_1 and e_2 have different colors. This completes the proof of Lemma 5.2. □

5.2 Hardness Results

In this section we study the complexity of problems δ-gap-MSR and δ-gap-MSR-DU. We first observe in Section 5.2.1 that these problems become more difficult to solve when δ grows. Then in Section 5.2.2 we focus on 1-gap-MSR and prove that this problem is NP-hard. Finally, in Sections 5.2.3 and 5.2.4, we prove the APX-hardness of δ-gap-MSR (for all $\delta \geq 2$) and δ-gap-MSR-DU (for all $\delta \geq 0$) respectively.

5.2.1 Hardness Increases with the Gap

In this section, we show that the problems δ-gap-MSR and δ-gap-MSR-DU become more and more difficult as δ increases. However, this result does not allow us to compare these problems to MSR and MSR-DU, for which the hardness results are quite independent (see [93] for the APX-hardness of these problems).

Theorem 5.3. *Let $\delta' > \delta \geq 2$ and $d \geq 2$ be integers. Then there exists an L-reduction from δ-gap-MSR-d to δ'-gap-MSR-d, and from δ-gap-MSR-DU-d to δ'-gap-MSR-DU-d.*

Proof. Let $\mathcal{M}_{1\ldots d}$ be an instance of δ-gap-MSR-d (the proof is similar for δ-gap-MSR-DU-d). For any $i \in [\![1\,;\,d]\!]$ and $k \geq 0$, we write

$$K_i^\delta(k) = \big\langle \mathcal{M}_i[\delta k + 1], \mathcal{M}_i[\delta k + 2], \ldots, \mathcal{M}_i[\delta k + \delta] \big\rangle.$$

We construct an instance of δ'-gap-MSR-d with d genomic maps $\mathcal{M}'_{1\ldots d}$ defined as follows:

$$\mathcal{M}_i^* = \big\langle K_i^\delta(0), \times^{\delta'-\delta}, K_i^\delta(1), \times^{\delta'-\delta}, \ldots \big\rangle$$

We show that there is a one-to-one correspondence between the δ-prestrips of $\mathcal{M}_{1\ldots d}$ and the δ'-prestrips of $\mathcal{M}^*_{1\ldots d}$. Let σ be a δ-prestrip of $\mathcal{M}_{1\ldots d}$, then it is also a subsequence of each genomic map in of $\mathcal{M}^*_{1\ldots d}$. Moreover, let $i \in [\![1\,;\,d]\!]$ and a and b be two consecutive markers of σ, such that a appears in $K_i^\delta(k)$ for some k, then b is in $K_1^\delta(k)$, $K_1^\delta(k-1)$ or $K_1^\delta(k+1)$. In the first case the gap between a and b is the same in $\mathcal{M}_i^*$ as in $\mathcal{M}_i$, and otherwise the gap is increased by exactly $\delta' - \delta$. Hence, since the gap between a and b is at most δ in $\mathcal{M}_i$, it is at most δ' in $\mathcal{M}_i^*$. Overall, σ is a δ'-prestrip in $\mathcal{M}^*_{1\ldots d}$.

Conversely, a δ'-prestrip σ' in $\mathcal{M}^*_{1\ldots d}$ corresponds to a δ-prestrip in $\mathcal{M}_{1\ldots d}$, and if a, b are two consecutive elements in σ' with a gap strictly greater than δ in $\mathcal{M}^*_{1\ldots d}$, then they cannot appear in the same $K_i^\delta(k)$, and thus the gap between a and b is reduced by at least $\delta' - \delta$. Hence σ' has gap at most δ in $\mathcal{M}_{1\ldots d}$.

This one-to-one correspondence is enough to prove that we have an L-reduction, since it preserves the prestrip lengths and the overlapping relation. □

5.2.2 1-gap-MSR Is NP-hard

In this section, we prove the following theorem.

Theorem 5.4. *1-gap-MSR is* NP*-hard.*

The proof uses a reduction from the NP-hard [21] restriction of Maximum Independent Set to the class of cubic, planar, and 2-connected graphs (CP2C-MIS, see Section 5.1.2). We also use the notion of 3-edge-coloring presented in the same section. Starting from any instance G of CP2C-MIS, we compute a 3-edge-coloring $(E^{\mathrm{A}}, E^{\mathrm{B}}, E^{\mathrm{C}})$ of G by Lemma 5.2, and we construct two genomic maps as follows.

First, we assign a list of 4 distinct positive integers (or 4 "markers") to each vertex $u \in V$: they are denoted y_u^{A1}, y_u^{A2}, y_u^{B1} and y_u^{B2}. We also assign a list of 10 distinct integers $x_{uv}^1, \ldots, x_{uv}^{10}$ to each edge $(u,v) \in E^{\mathrm{C}}$, in such a way that no integer appears in two different lists. We construct the genomic maps $\mathcal{M}_{1,2}$ with the following iterative procedure. Suppose we have arbitrarily ordered the vertices in V. In that case:

1. For all $(u,v) \in E^{\mathrm{A}}$ such that $u < v$, add $\left\langle y_u^{\mathrm{A1}}, y_v^{\mathrm{A1}}, y_u^{\mathrm{A2}}, y_v^{\mathrm{A2}}, \times, \times \right\rangle$ to $\mathcal{M}_1$.
2. For all $(u,v) \in E^{\mathrm{B}}$ such that $u < v$, add $\left\langle y_u^{\mathrm{B1}}, y_v^{\mathrm{B1}}, y_u^{\mathrm{B2}}, y_v^{\mathrm{B2}}, \times, \times \right\rangle$ to $\mathcal{M}_2$.
3. For all $(u,v) \in E^{\mathrm{C}}$ such that $u < v$, add $\Gamma_1(u,v)$ to $\mathcal{M}_1$, $\Gamma_2(u,v)$ to $\mathcal{M}_2$, where Γ_1 and Γ_2 are defined as:

$$\begin{aligned}\Gamma_1(u,v) = \langle & x_{uv}^1, x_{uv}^5, x_{uv}^2, x_{uv}^6, x_{uv}^3, x_{uv}^7, x_{uv}^4, \times, \times, \\ & y_u^{\mathrm{B1}}, x_{uv}^8, y_u^{\mathrm{B2}}, x_{uv}^9, y_v^{\mathrm{B1}}, x_{uv}^{10}, y_v^{\mathrm{B2}}, \times, \times \rangle \\ \Gamma_2(u,v) = \langle & x_{uv}^1, x_{uv}^8, x_{uv}^2, x_{uv}^9, x_{uv}^3, x_{uv}^{10}, x_{uv}^4, \times, \times, \\ & y_u^{\mathrm{A1}}, x_{uv}^5, y_u^{\mathrm{A2}}, x_{uv}^6, y_v^{\mathrm{A1}}, x_{uv}^7, y_v^{\mathrm{A2}}, \times, \times \rangle.\end{aligned}$$

Property 5.5. *Let $G = (V,E)$ be an n-vertex cubic graph with a 3-edge-coloring, and let $\mathcal{M}_{1,2}$ be the two genomic maps obtained by the construction defined above. Then the optimal value of 1-gap-MSR over $\mathcal{M}_{1,2}$ equals $4n + 2\alpha(G)$.*

Proof. In this proof, we use the following notations: for $u \in V$, $Y_u^{\mathrm{A}} = \left\langle y_u^{\mathrm{A1}}, y_u^{\mathrm{A2}} \right\rangle$ and $Y_u^{\mathrm{B}} = \left\langle y_u^{\mathrm{B1}}, y_u^{\mathrm{B2}} \right\rangle$. We write $\mathcal{Y} = \{Y_u^{\mathrm{A}} \mid u \in V\} \cup \{Y_u^{\mathrm{B}} \mid u \in V\}$, and Ω for the set of all 1-prestrips of $\mathcal{M}_{1,2}$. We also write $\ell_1(\mathcal{M}_{1,2})$ the optimal value of 1-gap-MSR$(\mathcal{M}_{1,2})$.

We first enumerate the prestrips of $\mathcal{M}_{1,2}$ appearing in Ω:

- For all $(u,v) \in E^{\mathrm{A}}$, both Y_u^{A} and Y_v^{A} belong to Ω. Moreover, Y_u^{A} and Y_v^{A} overlap in $\mathcal{M}_1$ (see step 1 of the construction).
- For all $(u,v) \in E^{\mathrm{B}}$, both Y_u^{B} and Y_v^{B} belong to Ω. Moreover, Y_u^{B} and Y_v^{B} overlap in $\mathcal{M}_2$ (see step 2 of the construction).
- For all $(u,v) \in E^{\mathrm{C}}$, $\left\langle x_{uv}^1, x_{uv}^2, x_{uv}^3, x_{uv}^4 \right\rangle$, $\left\langle x_{uv}^5, x_{uv}^6, x_{uv}^7 \right\rangle$ and $\left\langle x_{uv}^8, x_{uv}^9, x_{uv}^{10} \right\rangle$ belong to Ω. We write γ_{uv} the set containing those three prestrips and all their sub-prestrips (see Figure 5.1a).

Because of the gap condition, which prevents prestrips from overlapping the peg markers, there are no other prestrips in Ω.

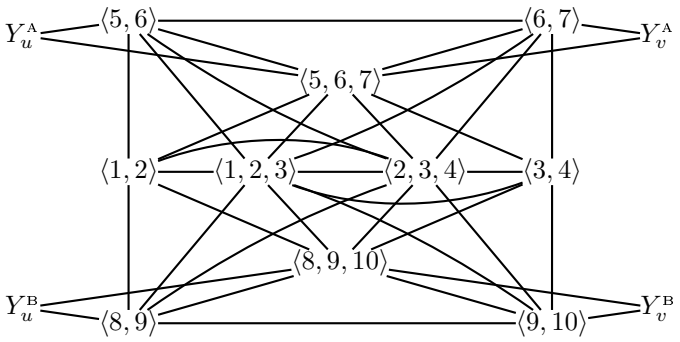
For an edge $(u,v) \in E^{\mathrm{C}}$, we name three feasible subsets of γ_{uv} (see Figure 5.1b):

$$\begin{aligned}\gamma_{uv}^{01} &= \left\{ \left\langle x_{uv}^1, x_{uv}^2 \right\rangle, \left\langle x_{uv}^6, x_{uv}^7 \right\rangle, \left\langle x_{uv}^9, x_{uv}^{10} \right\rangle \right\}, \\ \gamma_{uv}^{10} &= \left\{ \left\langle x_{uv}^3, x_{uv}^4 \right\rangle, \left\langle x_{uv}^5, x_{uv}^6 \right\rangle, \left\langle x_{uv}^8, x_{uv}^9 \right\rangle \right\}, \\ \gamma_{uv}^{00} &= \left\{ \left\langle x_{uv}^1, x_{uv}^2 \right\rangle, \left\langle x_{uv}^3, x_{uv}^4 \right\rangle \right\}.\end{aligned}$$

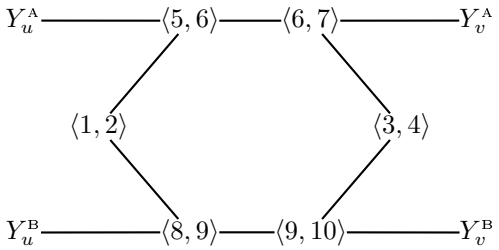
The first inequality we need to prove is the following:

$$\ell_1(\mathcal{M}_{1,2}) \geq 4n + 2\alpha(G).$$

Consider X a maximal independent set of G ($|X| = \alpha(G)$). Construct a set of prestrips $\mathcal{S}$ in the following way:



(a) All prestrips of γ_{uv} of length 2 or 3, and prestrips of $\mathcal{Y}$



(b) Induced subgraph with prestrips of the sets γ_{uv}^{01}, γ_{uv}^{10} and γ_{uv}^{00}

Figure 5.1: Overlapping prestrips of γ_{uv} for an arc $(u,v) \in E^{\mathrm{C}}$. We use $x_{uv}^i = i$ for all $1 \leq i \leq 10$, and edges are drawn between overlapping prestrips.

1. For all $(u,v) \in E^{\mathrm{A}}$, if $u \notin X$, then add Y_u^{A} to $\mathcal{S}$. Else, $v \notin X$: add Y_v^{A} to $\mathcal{S}$.
2. For all $(u,v) \in E^{\mathrm{B}}$, if $u \notin X$, then add Y_u^{B} to $\mathcal{S}$. Else, $v \notin X$: add Y_v^{B} to $\mathcal{S}$.
3. For all $(u,v) \in E^{\mathrm{C}}$, there are three possible cases:
   - If $u \notin X$ and $v \notin X$, add γ_{uv}^{00} to $\mathcal{S}$.
   - If $u \in X$ and $v \notin X$, add γ_{uv}^{10} to $\mathcal{S}$.
   - If $u \notin X$ and $v \in X$, add γ_{uv}^{01} to $\mathcal{S}$.

Before considering the overlaps in $\mathcal{S}$, we compute its total size: starting from 0, $||\mathcal{S}||$ is increased by 2 for each edge in E^{A} and in E^{B} (steps 1 and 2), and it is increased by either 6 or 4 for each edge in E^{C}, depending on whether this edge is incident to a vertex of X (step 3). As each vertex is incident to exactly one edge in E^{C}, we have the following formula:

$$||\mathcal{S}|| = 2|E^{\mathrm{A}}| + 2|E^{\mathrm{B}}| + 4|E^{\mathrm{C}}| + 2|X|.$$

Since each class E^{A}, E^{B} and E^{C} contains exactly $n/2$ edges, and $|X| = \alpha(G)$, we have

$$||\mathcal{S}|| = 4n + 2\alpha(G).$$

We now prove that $\mathcal{S}$ is a feasible set of prestrips. First note that prestrips of $\mathcal{S} \cap \mathcal{Y}$ are pairwise non-overlapping, since $\mathcal{S}$ never contains both Y_u^{A} and Y_v^{A} (resp. Y_u^{B} and Y_v^{B}) for $(u,v) \in E^{\mathrm{A}}$ (resp. $(u,v) \in E^{\mathrm{B}}$).

If $\sigma_1, \sigma_2 \in \mathcal{S} - \mathcal{Y}$, then there exist (u,v) and (u',v') such that $\sigma_1 \in \gamma_{uv}^{ij}$ and $\sigma_2 \in \gamma_{u'v'}^{i'j'}$ are prestrips of $\mathcal{S}$ (where (i,j) and (i',j') are in $\{(0,0),(0,1),(1,0)\}$). They also are non-overlapping: if $(u,v) \neq (u',v')$ then they appear in different sequences Γ_1 and Γ_2, and thus they cannot overlap. Otherwise, that is if $(u,v) = (u',v')$, then they appear in the same set γ_{uv}^{ij} which is, by construction, a set of non-overlapping prestrips.

Now suppose $\sigma_1 \in \mathcal{S} \cap \mathcal{Y}$ (e.g. $\sigma_1 = Y_w^{\mathrm{A}}$ for some vertex w, the case $\sigma_1 = Y_w^{\mathrm{B}}$ is similar) and $\sigma_2 \in \gamma_{uv}^{ij}$ are overlapping prestrips of $\mathcal{S}$. Then they can only overlap in $\Gamma_2(u, v)$, and $w = u$ or $w = v$. In the case $w = u$, (resp. $w = v$), σ_2 necessarily contains the element x_{uv}^5 (resp. x_{uv}^7), and thus γ_{uv}^{10} (resp. γ_{uv}^{01}) has been selected. It implies that $u \in X$ (resp. $v \in X$): in both cases, $w \in X$, which is a contradiction since Y_w^{A} can only be selected if $w \notin X$.

We conclude that prestrips in $\mathcal{S}$ are non-overlapping: consequently, $\mathcal{S}$ is a feasible set and we have

$$\ell_1(\mathcal{M}_{1,2}) \geq ||\mathcal{S}|| = 4n + 2\alpha(G).$$

To prove the other inequality of Property 5.5, that is

$$\alpha(G) \geq \frac{\ell_1(\mathcal{M}_{1,2}) - 4n}{2},$$

we consider $\mathcal{S}$, a maximal feasible set of prestrips of $\mathcal{M}_{1,2}$. Then $||\mathcal{S}|| = \ell_1(\mathcal{M}_{1,2})$.

We first enumerate and name the feasible subsets of γ_{uv} with total size at least 5, for any $(u, v) \in E^{\mathrm{C}}$. They are:

Name	Subset	Overlaps with
γ_{uv}^{10}	$\{\langle x_{uv}^3, x_{uv}^4\rangle, \langle x_{uv}^5, x_{uv}^6\rangle, \langle x_{uv}^8, x_{uv}^9\rangle\}$	Y_u^{A}, Y_u^{B}
γ_{uv}^{1a}	$\{\langle x_{uv}^5, x_{uv}^6, x_{uv}^7\rangle, \langle x_{uv}^8, x_{uv}^9\rangle\}$	Y_u^{A}, Y_u^{B}, Y_v^{A}
γ_{uv}^{1b}	$\{\langle x_{uv}^5, x_{uv}^6\rangle, \langle x_{uv}^8, x_{uv}^9, x_{uv}^{10}\rangle\}$	Y_u^{A}, Y_u^{B}, Y_v^{B}
γ_{uv}^{01}	$\{\langle x_{uv}^1, x_{uv}^2\rangle, \langle x_{uv}^6, x_{uv}^7\rangle, \langle x_{uv}^9, x_{uv}^{10}\rangle\}$	Y_v^{A}, Y_v^{B}
γ_{uv}^{a1}	$\{\langle x_{uv}^5, x_{uv}^6, x_{uv}^7\rangle, \langle x_{uv}^9, x_{uv}^{10}\rangle\}$	Y_u^{A}, Y_v^{A}, Y_v^{B}
γ_{uv}^{b1}	$\{\langle x_{uv}^6, x_{uv}^7\rangle, \langle x_{uv}^8, x_{uv}^9, x_{uv}^{10}\rangle\}$	Y_u^{B}, Y_v^{A}, Y_v^{B}
γ_{uv}^{11}	$\{\langle x_{uv}^5, x_{uv}^6, x_{uv}^7\rangle, \langle x_{uv}^8, x_{uv}^9, x_{uv}^{10}\rangle\}$	Y_u^{A}, Y_u^{B}, Y_v^{A}, Y_v^{B}

There is no feasible subset of γ_{uv} with total size 7 or more.

We construct a feasible set of prestrips $\mathcal{S}'$ in the following way. Add all prestrips of $\mathcal{S} \cap \mathcal{Y}$ to $\mathcal{S}'$. Then, for all $(u, v) \in E^{\mathrm{C}}$, three cases are possible:

Case 1. If $||\mathcal{S} \cap \gamma_{uv}|| \leq 4$, add γ_{uv}^{00} to $\mathcal{S}'$.

Case 2. If $\mathcal{S} \cap \gamma_{uv}$ is γ_{uv}^{01}, γ_{uv}^{a1}, γ_{uv}^{b1} or γ_{uv}^{11}, add γ_{uv}^{01} to $\mathcal{S}'$.

Case 3. If $\mathcal{S} \cap \gamma_{uv}$ is γ_{uv}^{10}, γ_{uv}^{1a} or γ_{uv}^{1b}, add γ_{uv}^{10} to $\mathcal{S}'$.

Note that it is impossible to have overlapping prestrips in $\mathcal{S}'$ after these steps: in case 1, because $\gamma_{uv}^{00} = \langle x_{uv}^1, x_{uv}^2, x_{uv}^3, x_{uv}^4\rangle$ does not overlap with any prestrip except in γ_{uv}. In case 2, the prestrips in γ_{uv}^{01} overlap with Y_v^{A} and Y_v^{B}, but it is also the case of the sets γ_{uv}^{a1}, γ_{uv}^{b1} and γ_{uv}^{11}: neither Y_v^{A} nor Y_v^{B} belongs to $\mathcal{S}$ (and they do not belong to $\mathcal{S}'$). And in case 3, the prestrips in γ_{uv}^{10} overlap with Y_u^{A} and Y_u^{B}, but it is also the case for the sets γ_{uv}^{1a} and γ_{uv}^{1b}: neither Y_u^{A} nor Y_u^{B} belongs to $\mathcal{S}$ (nor $\mathcal{S}'$).

At this point, we have a set $\mathcal{S}'$ which is feasible and that satisfies $||\mathcal{S}'|| \geq ||\mathcal{S}||$: indeed, each time we do not directly include a subset of $\mathcal{S}$, we include a set of prestrips with greater or equal total size. Hence, $||\mathcal{S}'|| = \ell_1(\mathcal{M}_{1,2})$.

We now create a first set of vertices $X_1 \subseteq V$ from $\mathcal{S}'$ with the following construction procedure. Start with $X_1 = \emptyset$, and for all $(u, v) \in E^{\mathrm{C}}$:

- If $\mathcal{S}' \cap \gamma_{uv} = \gamma_{uv}^{00}$, do nothing.
- If $\mathcal{S}' \cap \gamma_{uv} = \gamma_{uv}^{10}$, add u to X_1.
- If $\mathcal{S}' \cap \gamma_{uv} = \gamma_{uv}^{01}$, add v to X_1.

Two interesting remarks can be made about X_1. The first one is about its cardinality: since $||\gamma_{uv}^{00}|| = 4$ and $||\gamma_{uv}^{01}|| = ||\gamma_{uv}^{10}|| = 6$, then

$$|X_1| = \sum_{(u,v)\in E^{\mathrm{C}}} \frac{||\mathcal{S}' \cap \gamma_{uv}|| - 4}{2} \ .$$

The other remark is that, if $u \in X_1$, then $Y_u^{\mathrm{A}}, Y_u^{\mathrm{B}} \notin \mathcal{S}'$: indeed, let v be the vertex such that $(u,v) \in E^{\mathrm{C}}$ (the case $(v,u) \in E^{\mathrm{C}}$ is similar). Since $u \in X_1$, $\gamma_{uv}^{10} \subseteq \mathcal{S}'$: the prestrips in γ_{uv}^{10} overlap Y_u^{A} and Y_u^{B}, so none of them is in $\mathcal{S}'$.

Note that X_1 is not necessarily independent (we only know that for every edge $(u,v) \in E^{\mathrm{C}}$, u and v cannot both be in X_1). If an edge $(u,v) \in E^{\mathrm{A}} \cup E^{\mathrm{B}}$ is such that $u, v \in X_1$, we call it a *bad* edge. We call n_b the number of bad edges, and for each bad edge we arbitrarily remove one of its end vertices from X_1. The result is an independent set X with cardinality

$$|X| = |X_1| - n_b \ .$$

By the previous remark about X_1, we know that if $(u,v) \in E^{\mathrm{A}}$ is a bad edge, then neither Y_u^{A} nor Y_v^{A} belongs to $\mathcal{S}'$. In any other case, at most one of Y_u^{A} and Y_v^{A} belongs to $\mathcal{S}'$, since they overlap in $\mathcal{M}_1$. And it can be seen that the same occurs with edges of E^{B}, thus the number of prestrips in $\mathcal{S}' \cap \mathcal{Y}$ is at most $|E^{\mathrm{A}}| + |E^{\mathrm{B}}| - n_b$.

We have:

$$\begin{aligned}
||\mathcal{S}'|| &= ||\mathcal{S}' \cap \mathcal{Y}|| + \sum_{(u,v)\in E^{\mathrm{C}}} ||\mathcal{S}' \cap \gamma_{uv}|| \\
&= 2|\mathcal{S}' \cap \mathcal{Y}| + \sum_{(u,v)\in E^{\mathrm{C}}} ||\mathcal{S}' \cap \gamma_{uv}|| \\
&\leq 2(|E^{\mathrm{A}}| + |E^{\mathrm{B}}| - n_b) + \sum_{(u,v)\in E^{\mathrm{C}}} ||\mathcal{S}' \cap \gamma_{uv}||
\end{aligned}$$

Hence,

$$\sum_{(u,v)\in E^{\mathrm{C}}} ||\mathcal{S}' \cap \gamma_{uv}|| \geq ||\mathcal{S}'|| - 2(n - n_b)$$

Finally,

$$\begin{aligned}
\alpha(G) &\geq |X| \\
&= \left(\sum_{(u,v)\in E^{\mathrm{C}}} \frac{||\mathcal{S}' \cap \gamma_{uv}|| - 4}{2} \right) - n_b \\
&\geq \frac{||\mathcal{S}'|| - 2(n - n_b) - 4|E^{\mathrm{C}}|}{2} - n_b \\
&= \frac{||\mathcal{S}'|| - 4n}{2} \\
&= \frac{\ell_1(\mathcal{M}_{1,2}) - 4n}{2}
\end{aligned}$$

This last inequality achieves the proof of Property 5.5. □

Proof (of Theorem 5.4). The above property directly implies that our construction (which can clearly be achieved in polynomial time) leads to a reduction from CP2C-MIS to 1-gap-MSR, which proves Theorem 5.4. □

5.2.3 δ-gap-MSR Is APX-hard for $\delta \geq 2$

The δ-gap-MSR problem is known to be APX-hard, by extending the APX-hardness proof for MSR [95], which uses a reduction from a variant of SAT. We present here an alternative proof, using a reduction based on a graph approach, which leads to a somewhat larger inapproximability lower bound: 1.0106382 instead of 1.000625.

Theorem 5.6. *δ-gap-MSR is APX-hard for any $\delta \geq 2$. More precisely, it is NP-hard to approximate within $95/94 \simeq 1.0106382$.*

To prove this theorem, we present an L-reduction to 2-gap-MSR from the variant of Maximum Independent Set restricted to cubic graphs, (3-MIS, see Section 5.1.2). Using Theorem 5.3, the APX-hardness is extended to δ-gap-MSR for any $\delta \geq 2$.

Given a cubic graph $G = (V, E)$, our reduction consists in constructing two genomic maps $\mathcal{M}_{1,2}$, having properties P1, P2 and P3 described below, where Ω denotes the set of all δ-prestrips of $\mathcal{M}_{1,2}$:

P1. There exists a bijection Φ between V and Ω

P2. Every prestrip in Ω has length 2

P3. Two prestrips σ_1 and σ_2 of Ω are overlapping iff $(\Phi^{-1}(\sigma_1), \Phi^{-1}(\sigma_2)) \in E$

Let P_k denote the path graph with k vertices.

Lemma 5.7. *Given a cubic graph $G = (V, E)$, one can compute in polynomial time a partition of E into two classes E^B and E^W (for "Black" and "White" edges), such that (1) each connected component of (V, E^B) (called "black component") is isomorphic to a path graph , and (2) each connected component of (V, E^W) (called "white component") is isomorphic to a path graph P_k for some $k \leq 4$.*

Proof. Given a cubic graph $G = (V, E)$, we can compute in polynomial time a bipartition of the edges $E = E^{\mathrm{B}} \cup E^{\mathrm{W}}$ such that both (V, E^{B}) and (V, E^{W}) are linear forests (i.e. acyclic graphs of maximum degree 2, see [2]). At this point every black and white component is isomorphic to a path.

Suppose there exist 5 vertices a, b, c, d, e such that edges (a, b), (b, c), (c, d), (d, e) are white. We deduce that b, c and d cannot belong to the same black component (they are three different degree-1 vertices of (V, E^{B}), and a path graph has only 2 vertices of degree 1). Then either b and c, or c and d, are in different black components. In the first case, we can switch the color of (b, c) from white to black, and we can switch (c, d) in the second case. The result is that (V, E^{B}) and (V, E^{W}) are still linear forests, and we have strictly reduced the size of a white component. We can apply this process until no white component is longer than P_4: Lemma 5.7 is proved. □

The first step of the reduction from 3-MIS to 2-gap-MSR is to compute a partition of E into two classes E^{B} and E^{W} according to Lemma 5.7. We then construct two genomic maps $\mathcal{M}_{1,2}$, satisfying properties P1, P2 and P3. Moreover, incompatibilities in $\mathcal{M}_1$ (resp. $\mathcal{M}_2$) will correspond to black (resp. white) edges. We begin by assigning a different pair of integers (x_a, x'_a) to every vertex $a \in V(G)$; we write $\Phi(a) = \langle x_a, x'_a \rangle$.

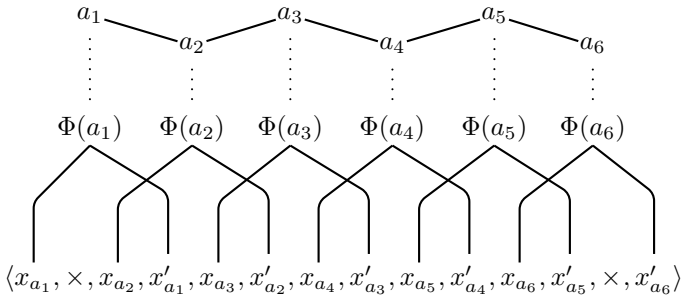


Figure 5.2: Transformation of a black component B_i (top) into the sequence I_i (bottom).

Then, for every black component B_i of order k, let $V(B_i) = \{a_h \mid 1 \leq h \leq k\}$ and let $(a_h, a_{h+1}) \in E^{\mathrm{B}}$ for $1 \leq h < k$; we construct the following sequence (see Figure 5.2):

$$I_i = \left\langle x_{a_1}, \times, x_{a_2}, x'_{a_1}, \ldots, x_{a_h}, x'_{a_{h-1}}, x_{a_{h+1}}, x'_{a_h}, \ldots, x_{a_k}, x'_{a_{k-1}}, \times, x'_{a_k} \right\rangle$$

The full genomic map $\mathcal{M}_1$ is given by $\mathcal{M}_1 = \left\langle I_1, \times^3, I_2, \times^3, \ldots \right\rangle$.

For $\mathcal{M}_2$, we use a similar construction, but we need to take the reversed opposite of some subsequences to avoid creating undesired prestrips. For a white component W_j having 4 vertices, say a, b, c and d with $(a, b), (b, c), (c, d) \in E^{\mathrm{W}}$, we create the following sequence:

$$J_j = \left\langle x_a, \times, x_b, x'_a, -x'_c, x'_b, -x'_d, -x_c, \times, -x_d \right\rangle.$$

If W_j is of order three (resp. two), we remove the extra elements from J_j, i.e. we obtain $J_j = \left\langle x_a, \times, x_b, x'_a, -x'_c, x'_b, \times, -x_c \right\rangle$ (resp. $J_j = \left\langle x_a, \times, x_b, x'_a, \times, x'_b \right\rangle$). Finally, $\mathcal{M}_2$ is created in the same way as $\mathcal{M}_1$: $\mathcal{M}_2 = \left\langle J_1, \times^3, J_2, \times^3, \ldots \right\rangle$.

Lemma 5.8. *The set Ω of the 2-prestrips of $\mathcal{M}_{1,2}$ is exactly $\{\Phi(a) \mid a \in V\}$. Moreover, $\Phi(a)$ and $\Phi(b)$ overlap in $\mathcal{M}_1$ iff $(a, b) \in E^B$, and $\Phi(a)$ and $\Phi(b)$ overlap in $\mathcal{M}_2$ iff $(a, b) \in E^W$.*

Proof. Suppose, by contradiction, that a prestrip of Ω contains markers corresponding to two different vertices u and v: then there exists $\sigma \in \Omega$ such that $\sigma = \left\langle \sigma_1, \sigma_2 \right\rangle$, with $\sigma_1 \in \{x_u, x'_u\}$ and $\sigma_2 \in \{x_v, x'_v\}$.

First note that σ_1 and σ_2 appear with the same orientation, since every element of $\mathcal{M}_1$ is positive. Because of the gap condition, both elements must appear in the same I_i in $\mathcal{M}_1$, and in the same J_j in $\mathcal{M}_2$. In J_j, the markers with a positive orientation come from the two prestrips associated to vertices (called a and b in the construction) linked by a white edge. Similarly, the negative markers come from two vertices c and d with $(c, d) \in E^{\mathrm{W}}$. So, whatever the orientation of σ in $\mathcal{M}_2$, there must be an edge $(u, v) \in E^{\mathrm{W}}$, and consequently this edge does not belong to E^{B}.

We look at the subsequences in I_i with gap at most 2 which do not contain any peg marker. Using the notations of the construction, they are of one of the following kinds:

1. $\left\langle x_{a_h}, x'_{a_{h-1}} \right\rangle$
2. $\left\langle x_{a_h}, x_{a_{h+1}} \right\rangle$
3. $\left\langle x_{a_h}, x'_{a_h} \right\rangle$

4. $\langle x'_{a_h}, x_{a_{h+2}} \rangle$
5. $\langle x'_{a_h}, x'_{a_{h+1}} \rangle$
6. $\langle x'_{a_h}, x_{a_{h+3}} \rangle$

If we write $u = a_h$, then v can be none of a_{h-1}, a_h or a_{h+1}, since $u \neq v$ and $(u, v) \notin E^{\text{B}}$. Only possibilities 4. and 6. remain, that is a prestrip of the form $\sigma = \langle x'_u, x_v \rangle$. However this kind of prestrip does not appear in J_j, thus we have proved that each prestrip of $\mathcal{M}_{1,2}$ is either of the kind $\langle x_u, x'_u \rangle$ or $\langle x'_u, x_u \rangle$. Moreover, for any $u \in V$, $\langle x'_u, x_u \rangle$ is not a subsequence of $\mathcal{M}_1$ nor $\mathcal{M}_2$, thus each prestrip of Ω can be written $\langle x_u, x'_u \rangle = \Phi(u)$ with $u \in V$.

Conversely, for every $a \in V$, $\langle x_a, x'_a \rangle$ is a subsequence of $\mathcal{M}_1$ with gap 2, and either $\langle x_a, x'_a \rangle$ or $\langle -x'_a, -x_a \rangle$ is a subsequence of $\mathcal{M}_2$ with gap 2. So $\Phi(a)$ is a 2-prestrip of $\mathcal{M}_{1,2}$: it belongs to Ω.

Finally, the second part of Lemma 5.8 is deduced from the construction of the sequences $\mathcal{M}_1$ and $\mathcal{M}_2$. □

The consequence of Lemma 5.8 is that $\mathcal{M}_{1,2}$ satisfies the three properties P1, P2 and P3 defined above. The reduction we have described is an L-reduction from 3-MIS to 2-gap-MSR (see Introduction, page 8), with coefficients $\alpha = 2$ and $\beta = 1/2$. Indeed, Φ transforms an independent set of cardinality k into a feasible set of prestrips of total size $\ell = 2k$, and Φ^{-1} does the reverse operation. We conclude that 2-gap-MSR and, by Theorem 5.3, δ-gap-MSR for $\delta \geq 2$ is APX-hard. More precisely, these problems are, like 3-MIS, NP-hard to approximate within 95/94. Thus Theorem 5.6 is proved.

5.2.4 δ-gap-MSR-DU Is APX-hard, for All δ

In this section, we focus on the variant of MSR allowing duplicates in the input sequences. The problem becomes much harder, even with a *0-gap* constraint. The 0-gap-MSR-DU shares similarities the Minimum Common String Partition problem (MCSP, see Chapter 4). Both problems deal with two sequences with duplicates, and aim at matching markers in order to reconstruct common strips. However, they differ both in the input sequences and in the optimization function. Indeed, each marker in an MCSP instance should have the same number of occurrences in both sequences, which is not necessary in MSR-DU. Moreover, in MCSP, one wants to create a minimum number of strips, using length-1 strips if necessary (and all elements are covered), while in MSR-DU the number of elements covered by the strips (each strip having length at least 2) has to be maximized.

Theorem 5.9. *δ-gap-MSR-DU is APX-hard for any $\delta \geq 0$. More precisely, it is NP-hard to approximate within $8649/8648 \simeq 1.000115$ for $\delta = 0, 1$, and within $95/94 \simeq 1.0106382$ for $\delta \geq 2$.*

We note that we need to consider only 0-gap-MSR-DU since APX-hardness of δ-gap-MSR-DU directly follows from APX-hardness of 0-gap-MSR-DU (see Theorem 5.3). Moreover, the inapproximability bound for $\delta \geq 2$ is directly deduced from Theorem 5.6.

As in the previous section, we use an L-reduction from 3-MIS, the variant of Maximum Independent Set restricted to cubic graphs.

This L-reduction is done in two steps. First, we transform the input graph such that it admits a partition of its edges and a labeling of its vertices with good properties (see below for the corresponding definitions). Then, using this partition and labeling, we can create an instance of 0-gap-MSR-DU which simulates the behavior of 3-MIS. Finally, Lemma 5.16 gives the whole L-reduction from the APX-hard problem 3-MIS [3] to 0-gap-MSR-DU, which achieves the proof of Theorem 5.9.

The first transformation of the reduction defines an oriented graph, for which we use the following definitions. If $G = (V, A)$ is a loopless oriented graph, $a = (u, v) \in A$ corresponds to an arc from u to v, of which u is the *source*, and v the *target*. The *degree* of a vertex $u \in V$ is the number of arcs $a \in A$ of which u is the source or the target. A subset X of V is *independent* if for all $(u, v) \in A$, X does not contain both u and v.

Definition 5.1 (Good partition)**.** *Let $G = (V, A)$ be a loopless directed graph. We say that $A = A_1 \cup A_2$ is a* good partition *of A if (i) $A_1 \cap A_2 = \emptyset$, (ii) for any $p \in \{1, 2\}$ and $a, b \in A_p$, a and b neither have the same source nor the same target, and (iii) (V, A_2) contains no cycle.*

Note that if $A = A_1 \cup A_2$ is a good partition of $G = (V, A)$, then every $u \in V$ has degree at most two in (V, A_1) and in (V, A_2) (using condition ii). Moreover, with condition iii, if $C \subseteq V$ is a connected component in the underlying undirected graph of (V, A_2), then we can write $C = \{u_0, u_1, \ldots, u_k\}$, such that the vertices $u_0, u_1, \ldots, u_k$ form a directed path: $(u_i, u_j) \in A_2 \Leftrightarrow j = i + 1$.

Definition 5.2 (Good labeling)**.** *Let $G = (V, A)$ be a loopless directed graph, Σ a set of labels, and $\phi : V \to \Sigma \times \Sigma$, where we write $\phi(u) = (u^1, u^2)$ the image of a vertex u.*

Then ϕ is said to be a good labeling *of G if*

1. *$u^1 \neq u^2$ for all $u \in V$,*
2. *$(v^1, v^2) \neq \phi(u)$ and $(v^2, v^1) \neq \phi(u)$ for any $u, v \in V$ such that $u \neq v$,*
3. *$u^2 = v^1$ for $(u, v) \in A$.*

Lemma 5.10. *Let $G = (V, E)$ be an undirected graph with maximum degree 3. Then we can compute, in polynomial time, a directed graph $G' = (V', A')$, a good partition $A' = A'_1 \cup A'_2$ of G' and a good labeling ϕ of G' with the following properties:*

a. $|V'| = |V| + 2|E|$, $|A'| = 3|E|$

b. the maximum degree of G' is 3

c. $\alpha(G') \geq \alpha(G) + |E|$

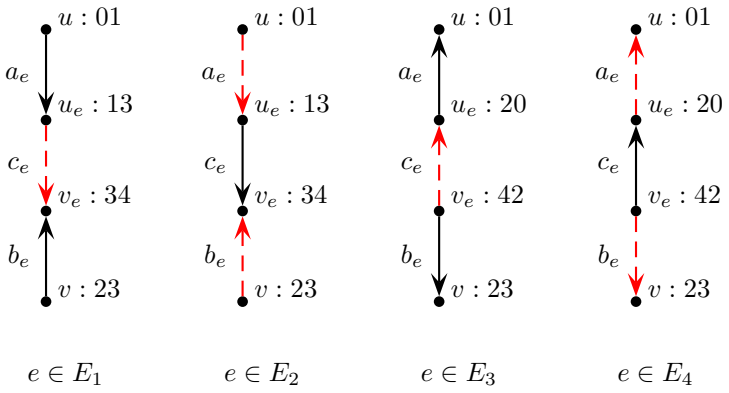
d. If X' is an independent set of G', there exists an independent set X of G such that $|X| \geq |X'| - |E|$ (and X can be computed in polynomial time).

Proof. We first use Vizing's theorem (see [106]) to obtain a 4-coloring of the edges of (V, E), that is a partition $E = E_1 \cup E_2 \cup E_3 \cup E_4$, such that two edges appearing in the same E_i are not incident.

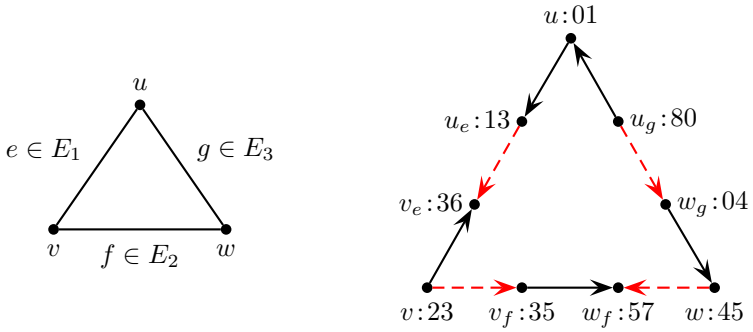
To create ϕ, we need a numbering of the vertices $y : V \to \{0, \ldots, |V| - 1\}$ and a numbering of the edges $x : E \to \{0, \ldots, |E| - 1\}$. For each $u \in V$, we choose $\phi(u) = (2y(u), 2y(u) + 1)$.

For each $e = \{u, v\} \in E$, we create two vertices u_e and v_e, and three arcs a_e, b_e, c_e such that (see Figure 5.3a, and an example in Figure 5.3b):

– If $e \in E_1 \cup E_2$, then $a_e = (u, u_e)$, $b_e = (v, v_e)$, $c_e = (u_e, v_e)$. Moreover, $\phi(u_e) = (u^2, v^2)$ and $\phi(v_e) = (v^2, 2|V| + x(e))$.



(a) Construction of vertices u_e, v_e, arcs a_e, b_e, c_e, and labeling for an edge $e = \{u, v\}$, and for each case $e \in E_1, E_2, E_3$ or E_4. We assume $y(u) = 0$, $y(v) = 1$ and $x(e) = 0$.



(b) Example on the triangle graph, with $y(u) = 0$, $y(v) = 1$, $y(w) = 2$ and $x(e) = 0$, $x(f) = 1$, $x(g) = 2$.

Figure 5.3: Construction of a good partition and a good labeling ϕ for an undirected graph of maximum degree 3. We write $u : u^1u^2$ for $\phi(u) = (u^1, u^2)$; arcs of A'_1 are solid, those of A'_2 are dashed.

– If $e \in E_3 \cup E_4$, then $a_e = (u_e, u)$, $b_e = (v_e, v)$, $c_e = (v_e, u_e)$. Moreover, $\phi(u_e) = (v^1, u^1)$, and $\phi(v_e) = (2|V| + x(e), v^1)$.

We add each arc a_e, b_e and c_e to either A'_1 or A'_2, according to the following rules:

– If $e \in E_1 \cup E_3$, then $a_e, b_e \in A'_1$ and $c_e \in A'_2$.
– If $e \in E_2 \cup E_4$, then $a_e, b_e \in A'_2$ and $c_e \in A'_1$.

Thus we have created a graph $G' = (V', A')$ with the set of vertices $V' = V \cup \{u_e, v_e \mid e \in E\}$ and the set of arcs $A' = A'_1 \cup A'_2$.

We show that $A' = A'_1 \cup A'_2$ is a good partition of G'. (i) $A'_1 \cap A'_2 = \emptyset$ by construction. (ii) Each vertex $u_e \in V' - V$ is adjacent to exactly one arc of A'_1 and one arc of A'_2. Each vertex $u \in V$ is adjacent in G to at most one edge in E_1 (resp. E_2), thus it is the source in G' of at most one arc in A'_1 (resp. A'_2). And it is also adjacent in G to at most one edge in E_3 (resp. E_4), so it is the target in G' of at most one arc in A'_1 (resp. A'_2). (iii) There are no cycles in (V', A'_2), since each connected component of this graph contains at most two arcs.

Moreover, ϕ is a good labeling of G'. First remark that for $u, v \in V$, $\{u^1, u^2\} \cap \{v^1, v^2\} = \emptyset$. It implies that condition 1. is true for all $u \in V'$, and that condition 2. is true for all $u \in V'$ and $v \in V$. For $u_e \in V' - V$ and $u'_{e'} \in V' - V$, with $e = \{u, v\}$ and $e' = \{u', v'\} \neq e$, we have $u' \notin \{u, v\}$ or $v' \notin \{u, v\}$. It implies that one element of $\phi(u'_{e'})$ does not appear in $\phi(u_e)$. Finally, condition 3. is verified by construction.

We now prove that G', the good partition and the good labeling have the required properties.

a. The cardinality conditions on V' and A' are satisfied by construction.

b. The degree of $u \in V$ in (V', A') is the same as in (V, E), thus it is at most 3. And the degree of $u_e \in V' - V$ in (V', A') is 2.

c. The independence number of the graph is increased by at least 1 each time we split an edge e into 3 arcs a_e, b_e, c_e (we can add either u_e or v_e to any independent set).

d. Let X' be an independent set of G'. We create X in the following way: start with $X = X'$. Then, consider each edge $e = (u, v)$ of G. Several cases are possible: if X' contains both u and v, then it contains neither u_e nor v_e, and we remove u from X. Otherwise, X' contains at most one element among $\{u_e, v_e\}$, and we remove this element. We obtain a set X which is a subset of V (it does not contain any vertex u_e or v_e) and is independent in G (it cannot contain both u and v for $(u, v) \in E$). Finally, we have removed at most one vertex per edge $e \in E$, so $|X| \geq |X'| - E$.

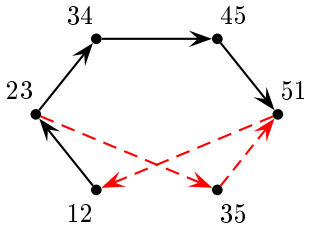
□

Let $G = (V, A)$ be a directed graph, with $A = A_1 \cup A_2$ a good partition of A (such that (V, A_2) is a degree-2 acyclic graph) and $\phi : V \to \Sigma \times \Sigma$ a good labeling of G. We give an arbitrary order over each set A_1, A_2 and V, and we construct two genomic maps $\mathcal{M}_{1,2}$ with the following procedure (see for example the directed graph in Figure 5.4a and the resulting maps in Figure 5.4b):

$\mathcal{M}_1 \leftarrow \langle\rangle$
For each $u \in V$
  $\mathcal{M}_1 \leftarrow \langle \mathcal{M}_1, \times, u^2, u^1 \rangle$
For each $(u, v) \in A_1$
  $\mathcal{M}_1 \leftarrow \langle \mathcal{M}_1, \times, u^1, u^2, v^2 \rangle$
For each $u \in V$ s.t. u has no incoming arc in A_1
  $\mathcal{M}_1 \leftarrow \langle \mathcal{M}_1, \times, u^1, u^2 \rangle$
For each $u \in V$ s.t. u has no outgoing arc in A_1
  $\mathcal{M}_1 \leftarrow \langle \mathcal{M}_1, \times, u^1, u^2 \rangle$
$\mathcal{M}_2 \leftarrow \langle\rangle$
For each connected component $\{u_0, u_1, \ldots, u_k\}$ in (V, A_2)
  / such that $u_0, u_1, \ldots, u_k$ is a path in (V, A_2), with $k \geq 0$ */*
  $\mathcal{M}_2 \leftarrow \langle \mathcal{M}_2, \times, u_0^1 \rangle$
  For $i \leftarrow 0$ to k
    $\mathcal{M}_2 \leftarrow \langle \mathcal{M}_2, u_i^2, u_i^1, u_i^2 \rangle$

The resulting maps have the property that, for all $u \in V$:
- there is exactly one occurrence of $\langle u^2, u^1 \rangle$ in $\mathcal{M}_1$, and exactly two occurrences of $\langle u^1, u^2 \rangle$ in $\mathcal{M}_1$ (recall that, for $(u, v) \in A_1$, $u^2 = v^1$). Moreover, these three subsequences are non-overlapping;
- there is exactly one occurrence of $\langle u^1, u^2, u^1, u^2 \rangle$ in $\mathcal{M}_2$, and no other occurrence of $\langle u^1, u^2 \rangle$ or $\langle u^2, u^1 \rangle$.

Moreover, the strip $\langle u^2, u^1 \rangle$ does not intersect any occurrence of $\langle v^1, v^2 \rangle$ or $\langle v^2, v^1 \rangle$ for $v \neq u$.

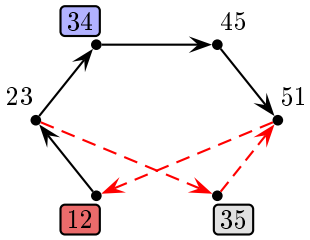


(a) Directed graph $G = (V, A)$ with a good partition of A and a good labeling of the vertices

$\mathcal{M}_1$: × 2 1 × 3 2 × 4 3 × 5 4 × 1 5 × 5 3
× 1 2 3 × 2 3 4 × 3 4 5 × 4 5 1
× 1 2 × 3 5 × 3 5 × 5 1

$\mathcal{M}_2$: × 2 3 2 3 5 3 5 1 5 1 2 1 2
× 3 4 3 4 × 4 5 4 5

(b) Maps created for the reduction



(c) Maximum independent set corresponding to a solution of 0-gap-MSR-DU($\mathcal{M}_{1,2}$)

Figure 5.4: Reduction from MIS to 0-gap-MSR-DU. In (a) and (c), each vertex of the graph is marked with the two integers of the good labeling. In (b), the prestrips of an optimal solution are highlighted. The prestrips corresponding to an independent set of G are framed and reported in (c).

Definition 5.3. *Let $\mathcal{M}_{1,2}$ be the maps constructed by the above procedure from a graph $G = (V, A)$, and let $\mathcal{O}$ be a solution of 0-gap-MSR-DU($\mathcal{M}_{1,2}$). We say that $u \in V$ is* selected *in $\mathcal{O}$ if both occurrences of $\langle u^1, u^2 \rangle$ appear in $\mathcal{O}$; we say it is unselected if only the strip $\langle u^2, u^1 \rangle$ appears in $\mathcal{O}$.*

Lemma 5.11. *All strips in a feasible solution have length 2. Moreover, each of the strips is of one of the following kinds: $\langle u^1, u^2 \rangle$ or $\langle u^2, u^1 \rangle$ for $u \in V$.*

Proof. Since the peg markers × cannot be selected in map $\mathcal{M}_1$, and a strip cannot overlap them (the gap constraint is $\delta = 0$), all strips are either length-2 strips of the kind $\langle u^1, u^2 \rangle$ or $\langle u^2, u^1 \rangle$, or length-3 strips of the kind $\langle u^1, u^2, v^2 \rangle$, with $(u, v) \in A_1$. We show that strips of this last kind are in fact impossible. Indeed, we have $v^2 \neq u^1$, and three different non-peg markers are never consecutive in map $\mathcal{M}_2$, except for the sequences $\langle u_i^1, u_i^2, u_{i+1}^2 \rangle$, where (u_i, u_{i+1}) is an arc of A_2. Hence if we have such a

length-3 strip, $u^1 = u_i^1$, $u^2 = u_i^2 = v^1 = u_{i+1}^1$, and $v^2 = u_{i+1}^2$. So $u = u_i$ and $v = u_{i+1}$, which implies that the arc (u, v) appears both in A_1 and in A_2, a contradiction. □

Lemma 5.12. *Given a feasible solution $\mathcal{S}$ of 0-gap-MSR-DU($\mathcal{M}_{1,2}$) of total size ℓ, we can create a feasible solution $\mathcal{S}'$ of total size at least ℓ where each vertex $u \in V$ is either selected or unselected.*

Proof. We start with $\mathcal{S}' = \mathcal{S}$. Remember that for all u, the strip $\langle u^2, u^1 \rangle$ only intersects the occurrences of $\langle u^1, u^2 \rangle$, which already implies that a vertex u cannot be both selected and unselected. If $\mathcal{S}'$ uses at most one strip among $\langle u^2, u^1 \rangle$ and the occurrences of $\langle u^1, u^2 \rangle$, then we can replace it by $\langle u^2, u^1 \rangle$ without creating conflicts, and the vertex u becomes unselected. Otherwise, $\mathcal{S}'$ uses two independent strips among $\langle u^2, u^1 \rangle$ and the occurrences of $\langle u^1, u^2 \rangle$, hence it cannot use $\langle u^2, u^1 \rangle$, and u is selected. □

Lemma 5.13. *If $(u, v) \in A$, then u and v cannot be both selected in a feasible solution.*

Proof. Two cases are possible: $(u, v) \in A_1$ and $(u, v) \in A_2$. In the first case, one occurrence of $\langle u^1, u^2 \rangle$ intersects an occurrence of $\langle v^1, v^2 \rangle$ in map $\mathcal{M}_1$ (in the sequence $\langle \times, u^1, u^2, v^2, \times \rangle$). So both occurrences of $\langle u^1, u^2 \rangle$ and both occurrences of $\langle v^1, v^2 \rangle$ cannot be all selected in the same solution. The situation is similar if $(u, v) \in A_2$, since an occurrence of $\langle u^1, u^2 \rangle$ intersects an occurrence of $\langle v^1, v^2 \rangle$ in the sequence $\langle u^1, u^2, u^1, u^2, v^2, v^1, v^2 \rangle$ of map $\mathcal{M}_2$. □

Lemma 5.14. *If $X \subset V$ is an independent set of $G = (V, A)$, then the set of strips selecting every vertex $u \in X$ and unselecting every $u \in V - X$ is a feasible solution of 0-gap-MSR-DU($\mathcal{M}_{1,2}$).*

Proof. We first create the feasible solution which keeps all the vertices $u \in V$ unselected (this solution contains all strips $\langle u^2, u^1 \rangle$, which are pairwise non-overlapping). For each $u \in X$, we replace $\langle u^2, u^1 \rangle$ by the two occurrences of $\langle u^1, u^2 \rangle$: these two strips do not overlap any v^2, v^1 for $v \in V$, nor any $\langle v^1, v^2 \rangle$ for $v \in X$ (this is because there is no arc linking u and v). Thus we end with a feasible set of strips, and every $u \in X$ is selected, while the other vertices are unselected. □

Lemma 5.15. *There exists an independent set X of G of cardinality k if, and only if, there exists a feasible solution $\mathcal{S}$ of 0-gap-MSR-DU ($\mathcal{M}_{1,2}$) of total size $\ell = 2(|V| + k)$. Moreover, given such an $\mathcal{S}$, the corresponding independent set X is computable in polynomial time.*

Proof. This is the corollary of the four previous lemmas: the "only if" part follows directly Lemma 5.14. For the "if" part, we start with $\mathcal{S}$ a feasible solution of 0-gap-MSR-DU($\mathcal{M}_{1,2}$) of total size ℓ. Using Lemma 5.12, we obtain a feasible solution $\mathcal{S}'$ of total size at least ℓ, such that, if X_1 is the set of selected vertices in $\mathcal{S}'$ and X_2 the set of unselected vertices, then $X_1 \cup X_2$ is a partition of V. Then $||\mathcal{S}'|| = 4|X_1| + 2|X_2|$, and X_1 is an independent set (by Lemma 5.13). Thus $\ell \leq 4|X_1| + 2|X_2| = 2(|V| + |X_1|)$. Then we can remove vertices from X_1 until we reach a set X such that $\ell = 2(|V| + |X|)$. □

Lemma 5.16. *There exists an L-reduction from 3-MIS to 0-gap-MSR-DU.*

Proof. We start with an instance $G = (V, E)$ of 3-MIS (G is a cubic graph). Using Lemma 5.10, we obtain a new directed graph $G' = (V', A')$, a good partition $A' = A'_1 \cup A'_2$ of G', and a good labeling ϕ of G'. Moreover, $|V'| = |V| + 2|E|$, and $\alpha(G') = \alpha(G) + |E|$. Since G is a cubic graph, we have $|E| = 3|V|/2$ and $|V| \leq 4\alpha(G)$, thus $|V'| = 4|V|$ and $\alpha(G') \leq 7\alpha(G)$. Next we create an instance $(\mathcal{M}_{1,2})$ of 0-gap-MSR-DU from $G' = (V', A')$ with the procedure described above. Let ℓ_0 be the optimal value of 0-gap-MSR-DU$(\mathcal{M}_{1,2})$. Applying Lemma 5.15 on the optimal solution we have,

$$\begin{aligned} \ell_0 &= 2(|V'| + \alpha(G')) \\ &\leq 2(4|V| + 7\alpha(G)) \\ &\leq 46\alpha(G). \end{aligned}$$

Hence we have the first inequality of the L-reduction. We now consider $\mathcal{S}$ a feasible solution of 0-gap-MSR-DU$(\mathcal{M}_{1,2})$. Using Lemma 5.15 we can construct an independent set X' of G' such that:

$$||\mathcal{S}|| = 2(|V'| + |X'|),$$

and using Lemma 5.10, we can deduce from X' an independent set X of G of cardinality $|X| \geq |X'| - |E|$. Hence,

$$\begin{aligned} \ell_0 - ||\mathcal{S}|| &= 2(|V'| + \alpha(G') - |V'| - |X'|) \\ &= 2(\alpha(G) + |E| - |X'|) \\ &\geq 2(\alpha(G) + |E| - |X| - |E|) \\ &= 2(\alpha(G) - |X|). \end{aligned}$$

This proves the second inequality of the L-reduction from 3-MIS to 0-gap-MSR-DU. Moreover, since 3-MIS is not approximable within 95/94 [52], 0-gap-MSR-DU is not approximable within $1 + 1/(94 \times 46 \times 2) = 8649/8648$. □

5.3 Polynomial-Time Algorithms

In this section we present polynomial-time algorithms for several variants of MSR. Algorithms in Sections 5.3.1, 5.3.2, 5.3.3 and 5.3.5 rely on literature algorithms – exact algorithms and approximations – for restrictions of the Maximum Independent Set problem presented in Section 5.1.2.

5.3.1 Reduction to Maximum Weight Independent Set

In this section we consider the variants of Maximum Weight Independent Set on the classes of interval and d-interval graphs. The following construction follows the one used by Chen et al. [47] to design a 4-approximation algorithm for MSR and MSR-DU and by Jiang [95]. We use this construction in order to extend the algorithm to the δ-gap variants of the problems using the approximation for d-Interval-MWIS (Theorem 5.18), and to design an exact polynomial-time algorithm for 0-gap-MSR using the exact algorithm for Interval-MWIS (Theorem 5.19).

Algorithm 5.1 Reduction from δ-gap-MSR-DU-d to d-Interval-MWIS.

Input: d genomic maps $\mathcal{M}_{1\ldots d}$, $\delta \in \mathbb{N} \cup \{\infty\}$

1: compute the weighted d-interval graph $G_\delta(\mathcal{M}_{1\ldots d})$
2: compute X, a $2d$-approximation for d-Interval-MWIS($G_\delta(\mathcal{M}_{1\ldots d})$)
3: **return** the feasible set of prestrips $\mathcal{S} = \{\sigma \mid I_\sigma \in X\}$

Given d genomic maps $\mathcal{M}_{1\ldots d}$ and a gap $\delta \in \mathbb{N} \cup \{\infty\}$, we construct a set of d-intervals in the following way. First, compute the set Ω of all prestrips of $\mathcal{M}_{1\ldots d}$ (possibly with the δ-gap constraint). Then, to each prestrip $\sigma \in \Omega$, assign the intervals $I_{\sigma,1} \ldots I_{\sigma,d}$ of $\mathbb{R}$ described below, the d-interval $I_\sigma = (I_{\sigma,1}, \ldots, I_{\sigma,d})$, and the weight $w(I_\sigma) = |\sigma|$. We write respectively $\min(\text{idx}(\sigma, \mathcal{M}))$ and $\max(\text{idx}(\sigma, \mathcal{M}))$ the indices of the first and last element of σ in a map $\mathcal{M}$, and $L = \max\{|\mathcal{M}_i| + 1 \mid 1 \leq i \leq d\}$.

$$I_{\sigma,i} = [iL + \min(\text{idx}(\sigma, \mathcal{M}_i));\ iL + \max(\text{idx}(\sigma, \mathcal{M}_i))],$$

We denote $G_\delta(\mathcal{M}_{1\ldots d})$ the weighted d-interval graph with vertex set $\{I_\sigma \mid \sigma \in \Omega\}$ and weight w. It has the following property:

Property 5.17. *Let $\mathcal{S} \subseteq \Omega$ and $X = \{I_\sigma \mid \sigma \in \mathcal{S}\}$. The set X is an independent set of $G_\delta(\mathcal{M}_{1\ldots d})$ with weight W iff $\mathcal{S}$ is feasible with total size W.*

Proof. Given two integers i and j and two prestrips σ and τ, intervals $I_{\sigma,i}$ and $I_{\tau,j}$ intersect iff $i = j$ and the subsequences of $\mathcal{M}_i$ σ and τ overlap. Thus we have the following equivalence:

$$I_\sigma \text{ and } I_\tau \text{ intersect } \Leftrightarrow \sigma \text{ and } \tau \text{ overlap.}$$

Hence, a set of d-intervals X is independent iff $\mathcal{S} = \{\sigma \mid I_\sigma \in X\}$ is feasible.

For the weight conservation, we have:

$$w(X) = \sum_{I_\sigma \in X} w(I_\sigma) = \sum_{\sigma \in \mathcal{S}} |\sigma| = ||\mathcal{S}||.$$

□

Theorem 5.18. *For any $d \geq 2$ and $\delta \in \mathbb{N} \cup \{\infty\}$, there exists a factor-$2d$ approximation algorithm for δ-gap-MSR-d and for δ-gap-MSR-DU-d.*

Proof. We use the construction described above to create Algorithm 5.1. Property 5.17 yields that the total size of $\mathcal{S}$ is the weight of X, and that δ-gap-MSR-$d(\mathcal{M}_{1\ldots d})$ and d-Interval-MWIS($G_\delta(\mathcal{M}_{1\ldots d})$) have the same optimal values. Consequently, $\mathcal{S}$ is a $2d$-approximation for δ-gap-MSR-d (when the input maps $\mathcal{M}_{1\ldots d}$ do not contain duplicates) and for δ-gap-MSR-DU-d. We thus have proved Theorem 5.18. □

From a theoretical point of view, this approximation algorithm is particularly interesting when d has high values, in which cases it has an almost tight ratio (compared to the asymptotic lower bound of $\Omega(d/\log d)$ proved by Jiang [95]).

Theorem 5.19. *There exists an exact polynomial-time algorithm for 0-gap-MSR-d and 0-gap-CMSR-d.*

Algorithm 5.2 $R(d, \delta)$-approximation algorithm for δ-gap-MSR-d.

Input: d genomic maps $\mathcal{M}_{1\ldots d}$ without duplicates, $\delta \in \mathbb{N}$

1: $\Omega_2 \leftarrow$ the set of all δ-prestrips of length 2
2: $E \leftarrow$ the subset of $\Omega_2 \times \Omega_2$ of overlapping prestrips
3: $G \leftarrow$ the $(p+1)$-claw-free graph (Ω_2, E), with $p = d \cdot (1 + \lfloor \frac{\delta}{2} \rfloor) + (\delta \bmod 2)$ by Lemma 5.21
4: **return** a $(p/2 + \epsilon)$-approximation of a Maximum Independent Set of G

Proof. We consider only the 0-gap-MSR-d problem, since the result extends trivially to 0-gap-CMSR-d.

Let $\mathcal{M}_{1\ldots d}$ be d genomic maps without duplicates (in fact, we only need to assume that $\mathcal{M}_1$ has no duplicates). The graph $G_0(\mathcal{M}_{1\ldots d})$ has the following property:

$$I_\sigma \text{ and } I_\tau \text{ intersect } \Leftrightarrow I_{\sigma,1} \cap I_{\tau,1} \neq \emptyset.$$

Indeed, if two prestrips overlap in $\mathcal{M}_i$ for some i, since they have gap zero, they must have a common marker m appearing in $\mathcal{M}_i$. But since m can appear only once in $\mathcal{M}_1$, they also overlap in $\mathcal{M}_1$. Thus $I_{\sigma,i} \cap I_{\tau,i} \neq \emptyset$ implies $I_{\sigma,1} \cap I_{\tau,1} \neq \emptyset$, which suffices to prove the claim. Using this property, we can see that $G_0(\mathcal{M}_{1\ldots d})$ is isomorphic to the interval graph with vertex set $\{I^1_\sigma \mid \sigma \in \Omega\}$. Hence, the reduction algorithm 5.1 can be used to obtain an optimal solution, by computing an optimal solution X of Interval-MWIS($G_0(\mathcal{M}_{1\ldots d})$) instead of a $2d$-approximation (line 2). This completes the proof of Theorem 5.19. □

Since the proof uses only the fact that $\mathcal{M}_1$ has no duplicates, we have in fact proved that the following problem, which is more general than 0-gap-MSR, is also in P:

Problem	Generalization of 0-gap-MSR-d
Input	d genomic maps $\mathcal{M}_{1\ldots d}$ each containing the same n markers, where $\mathcal{M}_1$ cannot have duplicates
Output	d subsequences $\mathcal{M}'_{1\ldots d}$ of $\mathcal{M}_{1\ldots d}$ respectively, each containing the same ℓ markers, such that the markers in $\mathcal{M}'_{1\ldots d}$ can be partitioned into strips, and such that each strip has gap 0
Maximize	the number ℓ of selected markers

5.3.2 Approximation Algorithm for δ-gap-MSR-d

In this section, we present an approximation algorithm for δ-gap-MSR-d. The approximation ratio being $O(d\delta)$, instead of $O(d)$, it is practical only for small values of δ.

Theorem 5.20. *Algorithm 5.2 computes an $R(d, \delta)$-approximation for δ-gap-MSR-d, where*

$$R(d, \delta) = \frac{3}{4}\left(d\left(1 + \left\lfloor \frac{\delta}{2} \right\rfloor\right) + (\delta \bmod 2)\right) + \epsilon.$$

Note that the values of $R(d,\delta)$ for small δ are the following:

$$R(d,\delta) = \begin{cases} 0.75d + 0.75 + \epsilon & \text{for } \delta = 1 \\ 1.5d + \epsilon & \text{for } \delta = 2 \\ 1.5d + 0.75 + \epsilon & \text{for } \delta = 3 \\ 2.25d + \epsilon & \text{for } \delta = 4. \end{cases}$$

Thus Algorithm 5.2 improves the $2d$-approximation (see Section 5.3.1) for $\delta \leq 3$, but will be improved by the 1.8-approximation in Section 5.3.3 for $d = 2$ and $\delta = 1$.

Lemma 5.21. *The size of any claw in the graph G created by Algorithm 5.2 is upper-bounded by $p = d(1 + \lfloor \frac{\delta}{2} \rfloor) + (\delta \bmod 2)$.*

Proof. Suppose there is a q-claw in $G = (\Omega_2, E)$. We denote by $c = \langle u, v \rangle$ its center, and by N the set of q neighbors of c in this claw: the prestrip c overlaps with each $n \in N$, but prestrips in N are pairwise non-overlapping. We partition N into $N = N_\cap \cup N_1 \cup \ldots \cup N_d$ as follows: if $n \in N$ shares a marker with c, then $n \in N_\cap$; otherwise, choose a map $\mathcal{M}_i$ for which $\{u, v\} \cup S^i(u, v)$ and $\{u', v'\} \cup S^i(u', v')$ intersect, and add n to N_i.

First note that $0 \leq |N_\cap| \leq 2$: a prestrip in $N_\cap$ either contains u or v, and non-overlapping prestrips cannot both contain u or v. Now for $1 \leq i \leq d$, we give an upper bound on $|N_i|$, depending on $|N_\cap|$. We can assume wlog. that u and v appear in positive form in $\mathcal{M}_i$, and that N_i consists of h prestrips $\langle x_1, y_1 \rangle, \ldots, \langle x_h, y_h \rangle$ appearing in this order in $\mathcal{M}_i$ (i.e., $\langle x_1, y_1, x_2, y_2, \ldots, x_h, y_h \rangle$ is a subsequence of G_i).

If $|N_\cap| = 2$, then x_1 must appear after u and y_h must appear before v in $\mathcal{M}_1$: otherwise $\langle x_1, y_1 \rangle$ (respectively $\langle x_h, y_h \rangle$) would overlap with some prestrip of $N_\cap$. Since there can be at most δ markers between u and v, we have $2h \leq \delta$.

If $|N_\cap| = 1$, suppose the prestrip in $N_\cap$ contains u: then x_1 must appear after u in $\mathcal{M}_i$. Also, x_h must appear before v, otherwise $\langle x_h, y_h \rangle$ would not overlap with $\langle u, v \rangle$. Hence we have $2h - 1 \leq \delta$.

Finally, if $|N_\cap| = 0$, then y_1 must appear after u and x_h before v, hence $2h - 2 \leq \delta$.

To summarize, we have the following bounds on $h = |N_i|$:

- If $|N_\cap| = 0$, then $|N_i| \leq 1 + \lfloor \frac{\delta}{2} \rfloor$.
- If $|N_\cap| = 1$, and δ is odd, then $|N_i| \leq 1 + \lfloor \frac{\delta}{2} \rfloor$.
- If $|N_\cap| = 1$, and δ is even, then $|N_i| \leq \lfloor \frac{\delta}{2} \rfloor$.
- If $|N_\cap| = 2$, then $|N_i| \leq \lfloor \frac{\delta}{2} \rfloor$.

Summing over all N_i and $N_\cap$, we have

$$\begin{aligned} q = |N| &\leq \max \begin{cases} 0 + d \cdot (1 + \lfloor \frac{\delta}{2} \rfloor) \\ 1 + d \cdot ((\delta \bmod 2) + \lfloor \frac{\delta}{2} \rfloor) \\ 2 + d \cdot \lfloor \frac{\delta}{2} \rfloor \end{cases} \\ &= \max \left\{ d, 1 + d \cdot (\delta \bmod 2), 2 \right\} + d \cdot \lfloor \tfrac{\delta}{2} \rfloor \\ &= d + (\delta \bmod 2) + d \cdot \lfloor \tfrac{\delta}{2} \rfloor \\ &= d \cdot (1 + \lfloor \tfrac{\delta}{2} \rfloor) + (\delta \bmod 2) = p, \end{aligned}$$

as desired. □

We now bound the approximation ratio of Algorithm 5.2. If $\mathcal{O}$ is an optimal solution of size ℓ, there is a solution of size $\frac{2}{3}\ell$ in Ω_2: all prestrips in $\mathcal{O}$ can be decomposed into smaller prestrips of length 2 or 3. If we remove the last element

Algorithm 5.3 1.8-approximation algorithm for 1-gap-MSR-2.

Input: d genomic maps $\mathcal{M}_{1...d}$ without duplicates

$T \leftarrow \{(0,1,2),(1,2,3),(2,3,4),(0,2),(1,2),(1,3),(2,3),(2,4),$
$(5,6),(5,7),(6,7),(6,8),(7,8)\}$
$\Omega \leftarrow$ set of all 1-prestrips of $\mathcal{M}_{1,2}$ of length 2 or 3
for $\lambda \leftarrow 1$ **to** 9 **do**
  $V^\lambda \leftarrow \{\sigma \mid \sigma \in \Omega,\ \pi_9(\text{idx}(\sigma, \mathcal{M}_2) - \lambda) \in T\}$
  $E^\lambda \leftarrow \{(\sigma_1, \sigma_2) \mid \sigma_1, \sigma_2$ overlapping prestrips of $V^\lambda\}$
  $w(\sigma) \leftarrow |\sigma|$ (for all $\sigma \in V^\lambda$)
  $V^\lambda_{Ind} \leftarrow$Claw-Free-MWIS of graph (V^λ, E^λ) with weight w
end for
return $\max\{||V^\lambda_{Ind}|| \mid 1 \leq \lambda \leq 9\}$

of each length-3 prestrip of $\mathcal{O}$, we obtain a feasible solution $\mathcal{O}'$ of size at least $\frac{2}{3}\ell$, included in Ω_2, which moreover forms an independent set of $G = (\Omega_2, E)$. Using the $(p/2+\epsilon)$-approximation of Maximum Independent Set on $(p+1)$-claw-free-graphs given in [81], we obtain an independent set of G corresponding to a set of prestrips of total length $\frac{1}{p/2+\epsilon}\frac{2}{3}\ell$. This leads to an approximation ratio of $\frac{3}{4}p + \epsilon$, where $p = d(1 + \lfloor \frac{\delta}{2} \rfloor) + (\delta \bmod 2)$.

5.3.3 Approximation Algorithm for 1-gap-MSR-2

In this section, we prove the following result.

Theorem 5.22. *Algorithm 5.3 is a factor-1.8 approximation algorithm for 1-gap-MSR-2.*

Proof. Algorithm 5.3 works as follows. Given two genomic maps $\mathcal{M}_{1,2}$, compute the set Ω of all prestrips with length 2 or 3 (and gap at most 1). Longer prestrips are ignored, since they can be split into smaller ones appearing in Ω. Select a subset $V^\lambda \subseteq \Omega$ (according to some parameter λ: see the selection process described below), and create E^λ, the set of all overlapping pairs of prestrips of V^λ. The pair (V^λ, E^λ) forms a graph which is claw-free (see Lemma 5.23). It is thus possible to compute, in polynomial time, a maximum independent set for this graph (see Section 5.1.2), which yields a feasible set of prestrips V^λ_{Ind}.

The selection of V^λ among Ω is done as follows: given a prestrip σ of $\mathcal{M}_{1,2}$, take the values of $\text{idx}(\sigma, \mathcal{M}_2) - \lambda$ modulo 9. This is done by the arithmetic function π_9, which takes the values of a list modulo 9: for example, if σ has indices $(30, 32, 33)$ in $\mathcal{M}_2$, and $\lambda = 5$, then $\text{idx}(\sigma, \mathcal{M}_2) - \lambda = (25, 27, 28)$, and $\pi_9(\text{idx}(\sigma, \mathcal{M}_2) - \lambda) = (7, 0, 1)$. If the result of $\pi_9(\text{idx}(\sigma, \mathcal{M}_2) - \lambda)$ belongs to some set (the vector T in Algorithm 5.3), add σ to V^λ. We only need to test the 9 different values of λ to obtain 9 different feasible sets of prestrips.

Finally, Lemma 5.30 proves that there exists some λ for which the total size of the corresponding V^λ_{Ind} is at least $5/9^{th}$ of a maximum feasible set of prestrips of $\mathcal{M}_{1,2}$. Thus, Algorithm 5.3 is a polynomial-time algorithm giving a 1.8-approximation to 1-gap-MSR, and Theorem 5.22 is proved. □

In the following lemma, we prove that the subset of prestrips selected by T is small enough (and, most importantly, well-structured) so that the corresponding graph is claw-free.

Lemma 5.23. *For each λ, the graph (V^λ, E^λ) created by Algorithm 5.3 is claw-free.*

Proof. Although it is not necessary for the algorithm, we assume here, wlog., that $\mathcal{M}_1$ is the identity permutation $\langle 1, 2, \ldots, |\mathcal{M}_1| \rangle$. This assumption simplifies somehow the notations: $\mathrm{idx}(\sigma, \mathcal{M}_1) = \sigma$.

Consider a prestrip $\sigma \in V^\lambda$. Since all elements of T have values included either in $\{0, 1, 2, 3, 4\}$ or in $\{5, 6, 7, 8\}$, and by construction $\pi_9(\mathrm{idx}(\sigma, \mathcal{M}_2) - \lambda) \in T$, we have

$$\begin{aligned} \pi_9(\mathrm{idx}(\sigma, \mathcal{M}_2) - \lambda) &\subseteq [\![0\,;\,4]\!] \\ \text{or } \pi_9(\mathrm{idx}(\sigma, \mathcal{M}_2) - \lambda) &\subseteq [\![5\,;\,8]\!]\,. \end{aligned}$$

Thanks to the gap condition, there exists some integer k such that the indices of σ in $\mathcal{M}_2$ are all in one of the following size-5 or size-4 intervals:

$$\begin{aligned} \mathrm{idx}(\sigma, \mathcal{M}_2) &\subseteq [\![0 + 9k + \lambda\,;\,4 + 9k + \lambda]\!] \\ \text{or } \mathrm{idx}(\sigma, \mathcal{M}_2) &\subseteq [\![5 + 9k + \lambda\,;\,8 + 9k + \lambda]\!]]. \end{aligned}$$

Let $K^\lambda(\sigma)$ be the size-5 or size-4 interval of $\mathcal{M}_2$ containing σ, such that:

$$\begin{aligned} K^\lambda(\sigma) &= \langle \mathcal{M}_2[0 + 9k + \lambda], \ldots, \mathcal{M}_2[4 + 9k + \lambda] \rangle \\ \text{or } K^\lambda(\sigma) &= \langle \mathcal{M}_2[5 + 9k + \lambda], \ldots, \mathcal{M}_2[8 + 9k + \lambda] \rangle. \end{aligned}$$

The last notations we use are for edges of E^λ: for $\sigma, \tau \in V^\lambda$, if $(\sigma, \tau) \in E^\lambda$, we write $\sigma \text{ — } \tau$. If σ and τ have a common element, we say that they *intersect*, and we write $\sigma \overset{\cap}{\text{—}} \tau$. Otherwise, they must overlap in $\mathcal{M}_1$ or $\mathcal{M}_2$ (possibly both): we write respectively $\sigma \overset{\mathcal{M}_1}{\text{—}} \tau$ or $\sigma \overset{\mathcal{M}_2}{\text{—}} \tau$.

Before proving that (V^λ, E^λ) is claw-free, we first give a series of properties over this graph.

Property 5.24. *Let σ, τ be distinct prestrips of V^λ. If σ and τ overlap in $\mathcal{M}_i$ for some $i \in \{1, 2\}$, without intersecting each other, then they both have gap exactly 1 in $\mathcal{M}_i$.*

Proof. All the prestrips we consider have gap at most 1. If one of them (say σ) has gap 0, then it would contain two consecutive elements in $\mathcal{M}_i$ which τ overlaps: consequently, τ would have gap at least 2, a contradiction. □

Property 5.25. *Let σ, τ be distinct prestrips of V^λ. If $\sigma \overset{\cap}{\text{—}} \tau$ or $\sigma \overset{\mathcal{M}_2}{\text{—}} \tau$, then $K^\lambda(\sigma) = K^\lambda(\tau)$.*

Proof. Both cases imply $K^\lambda(\sigma) \cap K^\lambda(\tau) \neq \emptyset$. Moreover, the indices over $\mathcal{M}_2$ are partitioned by the intervals $[\![0 + 9k + \lambda\,;\,4 + 9k + \lambda]\!]$ and $[\![5 + 9k + \lambda\,;\,8 + 9k + \lambda]\!]$, and this genomic map does not contain duplicates. Hence Property 5.25 follows. □

Property 5.26. *Let σ, τ be distinct prestrips of V^λ. If $K^\lambda(\sigma) = K^\lambda(\tau) (= K)$ and $|K| = 5$, then $\sigma \overset{\cap}{\text{—}} \tau$ or $\sigma \overset{\mathcal{M}_2}{\text{—}} \tau$.*

Proof. This property is deduced from the vector T of Algorithm 5.3: every element in T which is included in $[\![0\,;\,4]\!]$ either contains 2, or is $(1, 3)$. If σ and τ do not intersect, then, writing $K = \langle K[0], K[1], K[2], K[3], K[4] \rangle$, one of the two prestrips contains $K[2]$ and the other is $\langle K[1], K[3] \rangle$, in which case they overlap in $\mathcal{M}_2$. Consequently, $\sigma \overset{\cap}{\text{—}} \tau$ or $\sigma \overset{\mathcal{M}_2}{\text{—}} \tau$. □

Property 5.27. *Let σ, τ_1, τ_2 be distinct prestrips of V^λ. If $\sigma \overset{\mathcal{M}_2}{—} \tau_1$ and either $\sigma \overset{\mathcal{M}_2}{—} \tau_2$ or $\sigma \overset{\cap}{—} \tau_2$, then $\tau_1 \overset{\cap}{—} \tau_2$ or $\tau_1 \overset{\mathcal{M}_2}{—} \tau_2$.*

Proof. Let $K = K^\lambda(\sigma)$. Using Property 5.25 both on (σ, τ_1) and (σ, τ_2), we have $K = K^\lambda(\tau_1)$ and $K = K^\lambda(\tau_2)$. If σ, τ_1 and τ_2 do not intersect, they correspond to 3 disjoint subsets (each of cardinality 2 or 3) of K (which is of cardinality 4 or 5): a contradiction. Since σ and τ_1 do not intersect, either $\tau_1 \overset{\cap}{—} \tau_2$, (in which case the property is proved), or $\sigma \overset{\cap}{—} \tau_2$.

If K has size 5, since $K^\lambda(\tau_1) = K^\lambda(\tau_2)$, Property 5.26 applies directly on τ_1, τ_2.

If K has size 4, then σ, τ_1 and τ_2 have length 2. Since σ and τ_1 do not intersect, and $|K| = |\sigma| + |\tau_1|$, every element of K appears either in σ or τ_1. Now τ_2 is a subsequence of K, and there is at least one element of τ_2 which does not appear in σ, so τ_2 and τ_1 intersect. □

Property 5.28. *Let σ, τ be distinct prestrips of V^λ, with $|\sigma| = 2$. If $\sigma \overset{\mathcal{M}_1}{—} \tau$ then there exists $x \in \{1, \dots, |\mathcal{M}_1| - 2\}$ such that $\sigma = \langle x, x+2 \rangle$, and τ contains as sub-prestrip $\langle x-1, x+1 \rangle$ or $\langle x+1, x+3 \rangle$.*

Proof. By Property 5.24, σ has gap 1 in $\mathcal{M}_1$, so there exists x such that $\sigma = \langle x, x+2 \rangle$. The only two subsequences of size 2 overlapping $\langle x, x+2 \rangle$ without intersecting it are $\langle x-1, x+1 \rangle$ and $\langle x+1, x+3 \rangle$: τ must contain one of those as sub-prestrip. □

Property 5.29. *Let σ, τ_1, τ_2 be distinct prestrips of V^λ. If $\sigma \overset{\mathcal{M}_1}{—} \tau_1$ and $\sigma \overset{\mathcal{M}_1}{—} \tau_2$ then either $\tau_1 \overset{\cap}{—} \tau_2$, or there exists x such that $\sigma = \langle x, x+2, x+4 \rangle$, one of $\{\tau_1, \tau_2\}$ contains $\langle x-1, x+1 \rangle$, and the other contains $\langle x+3, x+5 \rangle$.*

Proof. If $|\sigma| = 2$ we can use Property 5.28 twice: there exists x such that $\sigma = \langle x, x+2 \rangle$, and both τ_1 and τ_2 contain $x+1$, hence $\tau_1 \overset{\cap}{—} \tau_2$.

Suppose now that τ_1 and τ_2 do not intersect, which implies $|\sigma| = 3$. Since σ and τ_1 are overlapping in $\mathcal{M}_1$, there exists an element $x \in \sigma$ appearing between the first and the last elements of τ_1, that is, $\min \tau_1 \leq x \leq \max \tau_1$. With the same arguments as previously, τ_1 contains $\langle x-1, x+1 \rangle$. There also exists $x' \in \sigma$ such that τ_2 contains $\langle x'-1, x'+1 \rangle$.

We can assume wlog that $x' \geq x$. Since the three prestrips do not intersect, $x' \notin \{x, x+1, x+2\}$, and x, x' cannot be consecutive in σ (otherwise the gap would be at least 2). Hence there exists x'' such that $x < x'' < x'$ and $\sigma = \langle x, x'', x' \rangle$. Now with the gap condition,

$$x'' \in \{x+1, x+2\} \cap \{x'-2, x'-1\},$$

and with the non intersecting condition,

$$x'' \notin \{x+1, x'-1\}.$$

Only one possibility remains:

$$x'' = x+2 = x'-2 \text{ and } \sigma = \langle x, x+2, x+4 \rangle.$$

This proves Property 5.29. □

We are now ready to prove Lemma 5.23: assume there exist four prestrips $\sigma, \tau_1, \tau_2, \tau_3$ forming a claw in (V^λ, E^λ), that is

$$\begin{array}{ccc} \sigma \text{ — } \tau_1, & \sigma \text{ — } \tau_2, & \sigma \text{ — } \tau_3, \\ (\tau_1, \tau_2) \notin E^\lambda, & (\tau_2, \tau_3) \notin E^\lambda, & (\tau_3, \tau_1) \notin E^\lambda. \end{array}$$

Let $n_{\mathcal{M}_1}$ be the number of prestrips in $\{\tau_1, \tau_2, \tau_3\}$ overlapping σ in $\mathcal{M}_1$ without intersecting it.

If $n_{\mathcal{M}_1} = 0$, then for all $j \in \{1, 2, 3\}$, either $\sigma \overset{\cap}{—} \tau_j$ or $\sigma \overset{\mathcal{M}_2}{—} \tau_j$. Hence we can use Property 5.25 to show that $K^\lambda(\sigma) = K^\lambda(\tau_1) = K^\lambda(\tau_2) = K^\lambda(\tau_3)$. In that case, τ_1, τ_2 and τ_3 are 3 prestrips of length 2 or 3 included in a set of size 4 or 5, so two of them must intersect, a contradiction.

The other trivial case is $n_{\mathcal{M}_1} = 3$. Using Property 5.29, we deduce that σ can be written $\sigma = \langle x, x+2, x+4 \rangle$ and that each one of τ_1, τ_2, τ_3 contains either $\langle x-1, x+1 \rangle$ or $\langle x+3, x+5 \rangle$. Again two of them must intersect, a contradiction.

Now we consider $n_{\mathcal{M}_1} = 2$: wlog, we can assume that $\sigma \overset{\mathcal{M}_1}{—} \tau_1$, $\sigma \overset{\mathcal{M}_1}{—} \tau_2$, and $\sigma \overset{\cap}{—} \tau_3$ or $\sigma \overset{\mathcal{M}_2}{—} \tau_3$. By Property 5.29, σ can be written $\sigma = \langle x, x+2, x+4 \rangle$, τ_1 contains $\langle x-1, x+1 \rangle$, and τ_2 contains $\langle x+3, x+5 \rangle$. Since σ has length 3, by definition of the vector T, see Algorithm 5.3, σ has gap 0 in $\mathcal{M}_2$. Hence by Property 5.24, the case $\sigma \overset{\mathcal{M}_2}{—} \tau_3$ is impossible, which implies $\sigma \overset{\cap}{—} \tau_3$. Moreover τ_3 does not overlap with τ_1 nor τ_2, so it contains neither x nor $x+4$. Necessarily, the common element between σ and τ_3 is $x+2$. Since $|\tau_3| \geq 2$ and since it has gap 0 or 1, it must contain an element among $\{x, x+1, x+3, x+4\}$ and thus overlap with τ_1 or τ_2: a contradiction.

Now, consider the last possible case, that is $n_{\mathcal{M}_1} = 1$ (wlog, assume that $\sigma \overset{\mathcal{M}_1}{—} \tau_1$). Property 5.27 applied to σ, τ_2 and τ_3, eliminates the cases where $\sigma \overset{\mathcal{M}_2}{—} \tau_2$ or $\sigma \overset{\mathcal{M}_2}{—} \tau_3$, since there can be no edge between τ_2 and τ_3. Hence we have $\sigma \overset{\cap}{—} \tau_2$ and $\sigma \overset{\cap}{—} \tau_3$. With Property 5.25, there exists a length-4 or length-5 sequence K such that $K = K^\lambda(\sigma) = K^\lambda(\tau_2) = K^\lambda(\tau_3)$.

If $|K| = 5$, Property 5.26 applies with τ_2 and τ_3, and we conclude that $\tau_2 \overset{\cap}{—} \tau_3$ or $\tau_2 \overset{\mathcal{M}_2}{—} \tau_3$, a contradiction.

Otherwise, $|K| = 4$. Since Algorithm 5.3 considers only length-2 prestrips in size-4 intervals $[\![5 + 9k + \lambda\,;\, 8 + 9k + \lambda]\!]$, we have $|\sigma| = 2$. With Property 5.28, there exists an x such that $\sigma = \langle x, x+2 \rangle$, and τ_1 contains either $\langle x-1, x+1 \rangle$ or $\langle x+1, x+3 \rangle$ as sub-prestrip. We only consider the case where τ_1 contains $\langle x-1, x+1 \rangle$, the other being similar. Let $j \in \{2, 3\}$, since τ_j intersects σ without overlapping with τ_1, τ_j contains $x+2$. Hence τ_2 and τ_3 have a common element, $x+2$, a contradiction.

Altogether, we have shown that the graph (V^λ, E^λ) cannot contain any claw, and Lemma 5.23 is proved. □

Lemma 5.30. *Let $\mathcal{O}$ be a maximal feasible set of prestrips for 1-gap-MSR($\mathcal{M}_{1,2}$). Then Algorithm 5.3 provides a solution of total size at least $5||\mathcal{O}||/9$.*

Proof. The proof relies on the construction of nine feasible sets of prestrips, denoted $\mathcal{O}^1, \ldots, \mathcal{O}^9$, such that each prestrip in $\mathcal{O}^\lambda$ appears both in $\mathcal{O}$ (possibly as a sub-prestrip) and in V^λ. We also require that

$$\sum_{\lambda=1}^{9} ||\mathcal{O}^\lambda|| \geq 5||\mathcal{O}||.$$

First note that we can assume that each prestrip in $\mathcal{O}$ has length 2 or 3: a prestrip cannot be shorter, and we can split longer ones. The approach is as follows: we start with nine empty sets $\mathcal{O}^1, \ldots, \mathcal{O}^9$. Then, for each prestrip $\sigma \in \mathcal{O}$, we enumerate the values of λ for which V^λ contains σ (or a sub-prestrip of σ), we add σ to the corresponding sets $\mathcal{O}^\lambda$, and we measure the increase of the sum $\sum_{\lambda=1}^{9} ||\mathcal{O}^\lambda||$. Examples are given in Figure 5.5.

For a prestrip σ of length 2 (see Figure 5.5b), we can add σ to $\mathcal{O}^\lambda$ only if we have $\pi_9(\text{idx}(\sigma, \mathcal{M}_2) - \lambda) \in T$. If $\text{idx}(\sigma, \mathcal{M}_2) = (y, y+1)$, then $\pi_9(\text{idx}(\sigma, \mathcal{M}_2) - \lambda)$ takes the values (0,1), (1,2), ..., (7,8), (8,0). Five of those nine pairs appear in the vector T of Algorithm 5.3: (1,2), (2,3), (5,6), (6,7) and (7,8). So we add σ to $\mathcal{O}^\lambda$ for 5 different values of λ: the total size $\sum_{\lambda=1}^{9} ||\mathcal{O}^\lambda||$ is increased by $10 = 5|\sigma|$.

The same goes for a prestrip of length 2 with indices $(y, y+2)$: it appears in V^λ for 5 different values of λ, with indices $(0,2), (1,3), (2,4), (5,7), (6,8)$. When added to the corresponding $\mathcal{O}^\lambda$, the total size is again increased by $10 = 5|\sigma|$.

Now we consider a prestrip σ of length 3 with indices $(y, y+1, y+2)$. There are three values of λ for which $\pi_9(\text{idx}(\sigma, \mathcal{M}_2) - \lambda) \in T$ (because $(0,1,2)$, $(1,2,3)$, $(2,3,4)$ are in T): we add σ to $\mathcal{O}^\lambda$ in those three cases. We now consider the two sub-prestrips of σ: σ_1 with indices $(y, y+1)$, and σ_2 with indices $(y+1, y+2)$. Amongst the 6 remaining values of λ for which $\pi_9(\text{idx}(\sigma, \mathcal{M}_2) - \lambda) \notin T$, there are 3 for which σ_1 is selected (corresponding to pairs (5,6), (6,7) and (7,8) in T), and one more for which only σ_2 is selected (corresponding to the pair (5,6) in T). The total size of $\sum_{\lambda=1}^{9} ||\mathcal{O}^\lambda||$ is increased by $3 \times 3 + 4 \times 2 = 17$, which is greater than $5|\sigma| = 15$.

We use similar arguments for other prestrips of length 3. If $\text{idx}(\sigma, \mathcal{M}_2) = (y, y+2, y+3)$ (see Figure 5.5c), we use pairs (0,2), (1,3), (2,4), (5,7) and (6,8) of T for σ_1, and (0,2), (5,6), (6,7) for σ_2. The quantity $\sum_{\lambda=1}^{9} ||\mathcal{O}^\lambda||$ is increased by 16.

If $\text{idx}(\sigma, \mathcal{M}_2) = (y, y+1, y+3)$, we use pairs (1,2), (2,3), (5,6), (6,7) and (7,8) of T for σ_1, and (1,3), (2,4), (6,8) for σ_2. Again $\sum_{\lambda=1}^{9} ||\mathcal{O}^\lambda||$ is increased by 16.

Finally, if $\text{idx}(\sigma, \mathcal{M}_2) = (y, y+2, y+4)$, we use pairs (0,2), (1,3), (2,4), (5,7) and (6,8) of T for σ_1, and (0,2),(1,3), (5,7), (6,8) for σ_2. In that case, $\sum_{\lambda=1}^{9} ||\mathcal{O}^\lambda||$ is increased by 18.

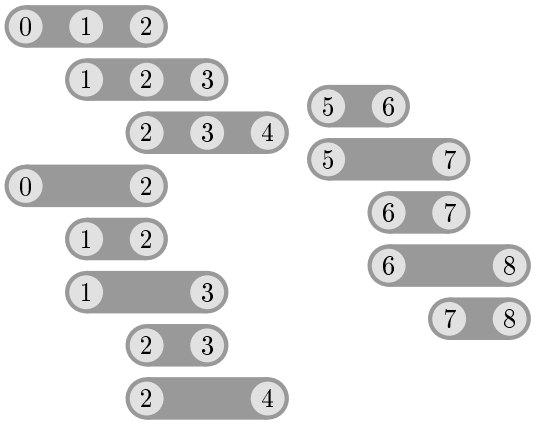
For each strip σ of $\mathcal{O}$, we have succeeded in adding σ, or sub-prestrips of σ, in several $\mathcal{O}^\lambda$ such that the total size is increased by at least $5|\sigma|$: we have 9 feasible sets (since $\mathcal{O}$ is feasible) satisfying the condition:

$$\sum_{\lambda=1}^{9} ||\mathcal{O}^\lambda|| \geq 5||\mathcal{O}||.$$

For each $\lambda \in \{1, \ldots, 9\}$, the prestrips of $\mathcal{O}^\lambda$, being taken from a feasible set $\mathcal{O}$, are pairwise non-overlapping and form an independent set of (V^λ, E^λ). Thus we have $||V^\lambda_{Ind}|| \geq ||\mathcal{O}^\lambda||$, and

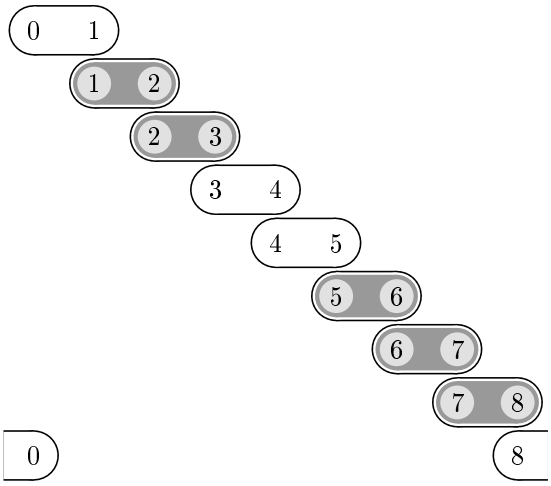
$$\begin{aligned}
\max\{||V^\lambda_{Ind}|| \mid 1 \leq \lambda \leq 9\} &\geq \frac{1}{9} \sum_{\lambda=1}^{9} ||V^\lambda_{Ind}|| \\
&\geq \frac{1}{9} \sum_{\lambda=1}^{9} ||\mathcal{O}^\lambda|| \\
&\geq \frac{5}{9} ||\mathcal{O}||.
\end{aligned}$$

Hence the solution returned by the algorithm is at least $5||\mathcal{O}||/9$. □



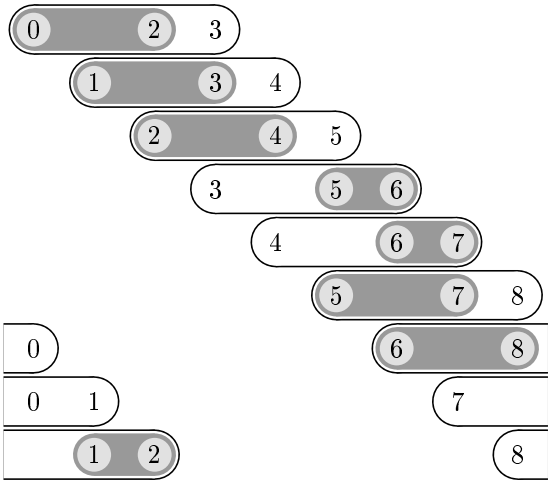
(a) Vector T defined in Algorithm 5.3

λ	$\pi_9(idx(\sigma, \mathcal{M}_2) - \lambda)$	Matching element in T
9	$(0, 1)$	$\emptyset$
8	$(1, 2)$	$(1, 2)$
7	$(2, 3)$	$(2, 3)$
6	$(3, 4)$	$\emptyset$
5	$(4, 5)$	$\emptyset$
4	$(5, 6)$	$(5, 6)$
3	$(6, 7)$	$(6, 7)$
2	$(7, 8)$	$(7, 8)$
1	$(8, 0)$	$\emptyset$



(b) Enumeration for $\text{idx}(\sigma, \mathcal{M}_2) = (y, y + 1)$. We assume wlog that $\pi_9(y) = 0$.

λ	$\pi_9(idx(\sigma, \mathcal{M}_2) - \lambda)$	Matching element in T
9	$(0, 2, 3)$	$(0, 2)$
8	$(1, 3, 4)$	$(1, 3)$
7	$(2, 4, 5)$	$(2, 4)$
6	$(3, 5, 6)$	$(5, 6)$
5	$(4, 6, 7)$	$(6, 7)$
4	$(5, 7, 8)$	$(5, 7)$
3	$(6, 8, 0)$	$(6, 8)$
2	$(7, 0, 1)$	$\emptyset$
1	$(8, 1, 2)$	$(1, 2)$



(c) Enumeration for $\text{idx}(\sigma, \mathcal{M}_2) = (y, y + 2, y + 3)$. We assume wlog that $\pi_9(y) = 0$.

Figure 5.5: Enumeration of the prestrips of V^λ matching a prestrip $\sigma \in \mathcal{O}$, for $\lambda \in \{1, \ldots, 9\}$. Note that V^λ contains all the prestrips whose indices taken modulo 9 are in T.

As a final remark, Algorithm 5.3 has been written to be as simple as possible with this approximation ratio. However, it can be easily improved (even if the theoretical ratio remains unchanged) by implementing – e.g., using heuristics – the following two steps.

- Before computing the independent set, add a maximal number of prestrips to the graph (V^λ, E^λ), with the constraint that the graph must remain claw-free.
- After computing the independent set, reinsert a maximal number of prestrips from $\Omega - V^\lambda$ to $\mathcal{O}^\lambda$, with the constraint that it must remain a feasible set.

5.3.4 Approximation Algorithm for 1-gap-CMSR-2

In this section, we give a method to transform approximation algorithms for 1-gap-MSR-d into approximation algorithms for 1-gap-CMSR-d, then apply this method to obtain a 2.778-approximation algorithm for 1-gap-CMSR-2.

Note that this transformation requires the algorithm to select all super-markers. This constraint is easily satisfied, since any solution of 1-gap-MSR-d can only be increased by adding super-markers (this is guaranteed by the gap-1 constraint: a super-marker can only make an existing selected prestrip grow, it cannot conflict with it).

Proposition 5.31. *If an algorithm $\mathcal{A}$ for* 1-gap-MSR-d *selects all super-markers and selects at least $1/r$ times the maximum number of single-markers selected in an optimal solution, then $\mathcal{A}$ is a $(1 + (1 - 1/r)2d)$-approximation for* 1-gap-CMSR-d.

Proof. We denote by s the total number of single-markers in the input maps $\mathcal{M}_{1\ldots d}$, by s^* the number of single-markers selected in an optimal solution, and by $s_\mathcal{A}$ the number of single-markers selected by algorithm $\mathcal{A}$. Then $s_\mathcal{A} \geq s^*/r$. Since $\mathcal{A}$ does not delete any super-marker, the number $k_\mathcal{A}$ of markers deleted by $\mathcal{A}$ is equal to $s - s_\mathcal{A}$. For 1-gap-CMSR-d, no optimal solution deletes a super-marker (otherwise the super-marker can be added back without breaking any strip since all strips have gap at most 1), so the number k^* of markers deleted by an optimal solution is equal to $s - s^*$. Since in any feasible solution, every selected single-marker must be adjacent to a deleted single-marker in some map, it follows that $s^* \leq 2dk^*$. The approximation ratio of $\mathcal{A}$ for 1-gap-CMSR-d is thus at most

$$\frac{k_\mathcal{A}}{k^*} = \frac{s - s_\mathcal{A}}{k^*} \leq \frac{(s^* + k^*) - (s^*/r)}{k^*} = 1 + \frac{(1 - 1/r)s^*}{k^*} \leq 1 + (1 - 1/r)2d. \quad \square$$

Theorem 5.32. *There exists a* 2.778*-approximation for* 1-gap-CMSR-2.

Algorithm 5.3 is a 1.8-approximation algorithm for 1-gap-MSR-2 and can be made to select all super-markers. With $r = 1.8 = 9/5$ and $d = 2$, we have $1 + (1 - 1/r)2d = 25/9 < 2.778$. By the above proposition, this algorithm for 1-gap-MSR-2 finds a 2.778-approximation for 1-gap-CMSR-2.

5.3.5 Approximation Algorithm for 0-gap-MSR-DU

In this section, we prove the following result.

Theorem 5.33. *Algorithm 5.4 is a factor-2.25 approximation algorithm for 0-gap-MSR-DU.*

Algorithm 5.4 2.25-approximation algorithm for 0-gap-MSR-DU.

Input: Two genomic maps $\mathcal{M}_{1,2}$ (possibly with duplicates)

$T \leftarrow \{(0,1,2),(0,1),(1,2)\}$
$\Omega \leftarrow$ set of all strips of $\mathcal{M}_{1,2}$ of length 2 or 3
for $\lambda_1 \leftarrow 0$ **to** 2 **do**
  for $\lambda_2 \leftarrow 0$ **to** 2 **do**
    $\lambda \leftarrow 3\lambda_1 + \lambda_2$
    $V^\lambda \leftarrow \{\sigma \mid \sigma \in \Omega,\ \pi_3(\text{idx}(\sigma, \mathcal{M}_1) - \lambda_1) \in T \text{ and } \pi_3(\text{idx}(\sigma, \mathcal{M}_2) - \lambda_2) \in T\}$
    $E^\lambda \leftarrow \{(\sigma_1, \sigma_2) \mid \sigma_1, \sigma_2 \text{ intersecting strips of } V^\lambda\}$
    $w(\sigma) \leftarrow |\sigma|$ (for all $\sigma \in V^\lambda$)
    $V^\lambda_{Ind} \leftarrow$MAXIMUM WEIGHT INDEPENDENT SET of (V^λ, E^λ) with weight w
  end for
end for
return $\max\{||V^\lambda_{Ind}|| \mid 0 \leq \lambda \leq 8\}$

Proof. Algorithm 5.4 follows the same lines as Algorithm 5.3 does for 1-gap-MSR, that is it computes an exact maximum weight independent set of a subgraph (V^λ, E^λ) of the graph representing the possible strips and the overlapping relation. Due to the possibility of having duplicates in the input genomes, the considered graph can be significantly more complex, and thus Algorithm 5.4 uses a more selective condition to create the set V^λ: the condition now bears on both $\text{idx}(\sigma, \mathcal{M}_1)$ and $\text{idx}(\sigma, \mathcal{M}_2)$. Lemma 5.34 proves that the subgraph (V^λ, E^λ) is indeed claw-free, which enables us to use a polynomial-time algorithm to find a maximum weight independent set of it, which corresponds to a feasible set of strips. The approximation ratio of $2.25 = 9/4$ is given by Lemma 5.35. □

Lemma 5.34. *For each λ, the graph (V^λ, E^λ) created by Algorithm 5.4 is claw-free.*

Proof. Let $\lambda = 3\lambda_1 + \lambda_2$. For each $\sigma \in V^\lambda$, from the definition of T, there exist two integers $k_1 = k_1(\sigma)$ and $k_2 = k_2(\sigma)$ such that:

$$\begin{aligned} idx(\sigma, \mathcal{M}_1) &\subseteq [\![3k_1 + \lambda_1\,;\, 3k_1 + \lambda_1 + 2]\!]\,, \\ idx(\sigma, \mathcal{M}_2) &\subseteq [\![3k_2 + \lambda_2\,;\, 3k_2 + \lambda_2 + 2]\!]\,. \end{aligned}$$

Moreover, σ contains the elements $\mathcal{M}_1[3k_1 + \lambda_1 + 1]$ and $\mathcal{M}_2[3k_2 + \lambda_2 + 1]$.

If σ and τ are two intersecting strips of V^λ, then they can intersect in $\mathcal{M}_1$ or in $\mathcal{M}_2$, which leads respectively to $k_1(\tau) = k_1(\sigma)$ and $k_2(\tau) = k_2(\sigma)$. Hence if σ has at least three neighbors in (V^λ, E^λ), then two of them, written τ_1 and τ_2, are such that $k_1(\tau_1) = k_1(\tau_2)$ or $k_2(\tau_1) = k_2(\tau_2)$. So τ_1 and τ_2 share a common element, namely $\mathcal{M}_1[3k_1(\tau_1) + \lambda_1 + 1]$ or $\mathcal{M}_2[3k_2(\tau_1) + \lambda_2 + 1]$ respectively, and there is an edge between them in (V^λ, E^λ). □

Lemma 5.35. *If $\mathcal{O}$ is an optimal solution of 0-gap-MSR-DU($\mathcal{M}_{1,2}$), Algorithm 5.4 provides a solution of total size at least $4||\mathcal{O}||/9$.*

Proof. We can assume, wlog, that all strips in $\mathcal{O}$ have length 2 or 3. We now create nine sets of strips $\mathcal{O}^0, \ldots, \mathcal{O}^8$ such that each strip in $\mathcal{O}^\lambda$ appears both in V^λ and in $\mathcal{O}$ (possibly as a substrip), and such that

$$\sum_{\lambda=0}^{8} ||\mathcal{O}^\lambda|| \geq 4||\mathcal{O}||.$$

Let σ be a strip of $\mathcal{O}$, and r_1, r_2 two integers in $\{0, 1, 2\}$ such that σ starts at position r_1 modulo 3 in $\mathcal{M}_1$ and r_2 modulo 3 in $\mathcal{M}_2$.

First suppose σ has length 2. Then for $\lambda_1 \in \{r_1 - 1, r_1\}$ modulo 3 and for $\lambda_2 \in \{r_2 - 1, r_2\}$ modulo 3, we have $\pi_3(\text{idx}(\sigma, \mathcal{M}_1) - \lambda_1) \in T$ and $\pi_3(\text{idx}(\sigma, \mathcal{M}_2) - \lambda_2) \in T$, thus we have $\sigma \in V^\lambda$ for four different values of λ. We add σ to the corresponding sets $\mathcal{O}^\lambda$, which increases the total size $\sum_{\lambda=0}^{8} ||\mathcal{O}^\lambda||$ by $8 = 4|\sigma|$.

Now suppose that σ has length 3. For $\lambda_1 = r_1$ and $\lambda_2 = r_2$, $\pi_3(\text{idx}(\sigma, \mathcal{M}_1) - \lambda_1) = \pi_3(\text{idx}(\sigma, \mathcal{M}_2) - \lambda_1) = (0, 1, 2)$, which is in T, thus we have $\sigma \in V^\lambda$. Moreover for $\lambda_1 \in \{r_1 - 1, r_1\}$ and $\lambda_2 \in \{r_2 - 1, r_2\}$, the beginning of σ, $\langle \sigma[1], \sigma[2] \rangle$, forms a length-2 strip appearing in V^λ. And for $\lambda_1 \in \{r_1, r_1 + 1\}$ and $\lambda_2 \in \{r_2, r_2 + 1\}$, the end of σ, $\langle \sigma[2], \sigma[3] \rangle$, forms a length-2 strip appearing in V^λ. Then for one value of λ (namely $3r_1 + r_2$), we add the length-3 strip σ to $\mathcal{O}^\lambda$ and for six other values of λ, we add one of the length-2 strips $\langle \sigma[1], \sigma[2] \rangle$ or $\langle \sigma[2], \sigma[3] \rangle$ to $\mathcal{O}^\lambda$. Thus the total size $\sum_{\lambda=0}^{8} ||\mathcal{O}^\lambda||$ is increased by $3 + 6 \times 2 = 15 > 4|\sigma|$.

Thus we indeed have

$$\sum_{\lambda=0}^{8} ||\mathcal{O}^\lambda|| \geq \sum_{\sigma \in \mathcal{O}} 4|\sigma| = 4||\mathcal{O}||.$$

Hence there exists some λ such that $||\mathcal{O}^\lambda|| \geq \frac{4}{9}||\mathcal{O}||$, and $\mathcal{O}^\lambda$ forms an independent set of (V^λ, E^λ) since a set of strips or substrips of $\mathcal{O}$ is necessarily feasible. Thus the size of the corresponding V^λ_{Ind} is at least $||\mathcal{O}^\lambda||$ and Algorithm 5.4 gives a solution of size at least $\frac{4}{9}||\mathcal{O}||$: it is indeed a $\frac{9}{4} = 2.25$-approximation. □

5.3.6 Approximation Algorithm for CMSR-d and δ-gap-CMSR-d

In this section, we present a $(d + 1.5)$-approximation algorithm for the two minimization problems CMSR-d and δ-gap-CMSR-d.

Theorem 5.36. *Algorithm 5.5 finds a $(d + 1.5)$-approximation for* CMSR-d *and* δ-gap-CMSR-d *for any $d \geq 2$ and $\delta \geq 1$.*

Let k be the number of deleted markers in an optimal solution. Then the number of single-markers in the input maps is at most $(2d + 1)k$ because each single-marker is either deleted or adjacent to a deleted marker. This immediately yields a $(2d + 1)$-approximation algorithm: simply delete all single-markers. The following is a tight example for this algorithm.

Example 5.2. *The optimal solution of CMSR-d over $\mathcal{M}_{1\ldots d}$ below is to delete the single-marker x, instead of all $2d + 1$ single-markers (for the naive algorithm):*

$$\begin{array}{lccccc}
\mathcal{M}_1 = \langle & z_d, y_d, & \cdots & z_3, y_3, & z_2, y_2, & z_1, x, y_1 \rangle \\
\mathcal{M}_2 = \langle & z_1, y_1, & z_2, x, y_2, & z_3, y_3, & \cdots & z_d, y_d \rangle \\
\mathcal{M}_3 = \langle & z_1, y_1, & z_2, y_2, & z_3, x, y_3, & \cdots & z_d, y_d \rangle \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\mathcal{M}_d = \langle & z_1, y_1, & z_2, y_2, & z_3, y_3, & \cdots & z_d, x, y_d \rangle
\end{array}$$

Algorithm 5.5 $(d+1.5)$-approximation for CMSR-d and δ-gap-CMSR-d.

Input: d genomic maps $\mathcal{M}_{1...d}$ without duplicates, $\delta \in \mathbb{N} \cup \{\infty\}$

1: $X \leftarrow \{$ triples of markers $(z, x, y) \mid z \prec y$ and $\mathsf{gap}(z, y) = \{x\}\}$
2: partition the markers into single-super-markers
3: **for all** $(z, x, y) \in X$ **do**
4: **if** x, y and z are not deleted **and** y or z is a single-marker **then**
5: delete x
6: re-create all super-markers
7: **end if**
8: **end for**
9: delete all remaining single-markers
10: **return** the resulting genomic map

As we can see from the above example, after one single-marker is deleted, many other single-markers may be merged into strips. Algorithm 5.5 first identifies (line 1) all triples of markers (z, x, y) such that z and y can be merged into a strip $\langle z, y \rangle$ after x is deleted. Such markers x are considered to be "cost-efficient" when z and/or y is a single-marker, since they allow, if deleted, to merge at least one single-marker into a super-marker. Thus the algorithm successively deletes (lines 2–8) those cost-efficient single-markers (each time reducing the total number of single-markers by at least 2), and finally removes (line 9) the remaining single-markers.

The approximation ratio analysis is non-trivial. We first give a number of definitions and easy remarks (inequalities (5.1) and (5.2)). We then bound on one hand the number of cost-efficient single-markers which are deleted by the algorithm but not by an optimal solution ($|D - \mathcal{K}|$ in (5.3)), and on the other hand the number of single-markers which have been cost-efficient for the optimal solution but are not deleted as such by the algorithm ($|R_1|$ in (5.4)). Finally, combining inequalities (5.1) to (5.4), we obtain a lower bound on the approximation ratio.

Lemma 5.37. *For each triple (z, x, y) in the set X in Algorithm 5.5, at least one of the three markers x, y, z must be deleted in any feasible solution.*

Proof. We prove the lemma by contradiction. Suppose that all three markers x, y, z are selected in a solution. Assume wlog. that the sequence $\langle z, x, y \rangle$ appears in some map. Then x must be in the same strip as one of z or y. Assume wlog. that $\langle z, x \rangle$ is part of some strip. Then $z \prec x$. Recall that $z \prec y$. Thus x and y are both candidate successors of z. By Lemma 5.1a, we have $y \in \mathsf{gap}(z, x)$, thus y must be deleted: a contradiction. □

We next prove the approximation ratio of Algorithm 5.5. Let $\mathcal{K}$ be the set of deleted markers in an optimal solution $\mathcal{O}$; $|\mathcal{K}| = k$. For each marker $x \notin \mathcal{K}$, we define two sets $\Gamma_{succ}(x)$ and $\Gamma_{pred}(x)$ as follows. If x is followed by a marker y in a strip of $\mathcal{O}$, $\Gamma_{succ}(x) = \mathsf{gap}(x, y)$; otherwise x is the last marker of its strip, $\Gamma_{succ}(x) = \emptyset$. If x is preceded by a marker z in a strip of $\mathcal{O}$, $\Gamma_{pred}(x) = \mathsf{gap}(z, x)$; otherwise x is the first marker of its strip, $\Gamma_{pred}(x) = \emptyset$. Then, for each marker $x \notin \mathcal{K}$, define $\gamma(x) = |\Gamma_{succ}(x)| + |\Gamma_{pred}(x)|$, and for each marker $x \in \mathcal{K}$, define $\gamma(x) = 0$. Intuitively, function γ can be seen as the *cost* that the optimal solution needed to pay to keep a single-marker into a strip. Note that if x is a single-marker of the input maps, then $\gamma(x) = 0$ iff $x \in \mathcal{K}$.

First, we can see that each marker $y \in \mathcal{K}$ is counted by γ at most twice in each map: at most once in some $\Gamma_{pred}(x_1)$, and at most once in some $\Gamma_{succ}(x_2)$. Thus we have the following inequality:

$$\sum_{x \text{ single-marker}} \gamma(x) \leq 2dk. \tag{5.1}$$

Refer now to Algorithm 5.5. Let D be the set of markers deleted in line 5, let S be the set of single-markers that are merged into super-markers in line 6, and let R be the set of markers deleted in line 9. Let $R_1 = \{r \in R \mid \gamma(r) = 1\}$ and $R_2 = \{r \in R \mid \gamma(r) \geq 2\}$. Thus we have a partition of R given by $R = (R \cap \mathcal{K}) \cup R_1 \cup R_2$.

Each marker $x \in D$ has a corresponding triple $(z, x, y) \in X$, where z or y is a single-marker. After x is deleted in line 5, z and y are merged into the same super-marker in line 6. Thus we have the following inequality:

$$|D| \leq |S|. \tag{5.2}$$

For each marker $x \in D - \mathcal{K}$, let $\phi(x)$ be an arbitrary marker in the non-empty set $\{z, x, y\} \cap \mathcal{K}$; see Lemma 5.37. Obviously $\phi(x) \neq x$, thus $\phi(x) \in \mathcal{K} - D$. We show that at most two markers in $D - \mathcal{K}$ can have the same image by ϕ. Suppose that $\phi(x_1) = \phi(x_2) = \phi$ for two different markers $x_1, x_2 \in D - \mathcal{K}$, where x_1 is deleted before x_2 in Algorithm 5.5. Then the marker ϕ is merged into a super-marker after x_1 is deleted, and again merged into a larger super-marker after x_2 is deleted. Since a marker has at most two neighbors in a super-marker, ϕ is necessarily a single-marker before x_1 is deleted, so it belongs to S, indeed $S \cap \mathcal{K}$. Moreover, after x_2 is deleted and ϕ is merged into a larger super-marker, ϕ cannot be adjacent to any other single-marker, say x_3. Therefore

$$|D - \mathcal{K}| \leq |\mathcal{K} - D| + |S \cap \mathcal{K}|. \tag{5.3}$$

Let u be a marker such that $\gamma(u) = 1$. Then by definition of γ, u belongs to some strip in the optimal solution, and it has a neighbor $v = \psi(u)$ in the same strip such that $\mathsf{gap}(u, v)$ contains only one marker, say x. Note that $u, v \notin \mathcal{K}$ and $x \in \mathcal{K}$. We claim that if u is a single-marker at the beginning of the algorithm, then either $u \in D \cup S$ or $v \in D$. This claim is clearly true if one of u or v is deleted by the algorithm in line 5. Otherwise, with $(v, x, u) \in X$ or $(u, x, v) \in X$, either x is not deleted because u is merged into a super-marker, or x is deleted: in both cases $u \in S$. This proves the claim. So for each $u \in R_1$, we have $v = \psi(u) \in D$, indeed $v \in D - \mathcal{K}$. Note that there can be at most two markers u_1 and u_2 with the same image v by ψ: the two neighbors of v in some strip in the optimal solution. Thus we have $|R_1| \leq 2|D - \mathcal{K}|$. Moreover, if there are two markers u_1 and u_2 with the same image v, then $\gamma(v) \geq 2$. Therefore

$$|R_1| \leq \sum_{v \in D - \mathcal{K}} \gamma(v). \tag{5.4}$$

Combining inequalities (5.1), (5.2), (5.3), and (5.4), the calculation in the following shows that the number of deleted markers, $|D| + |R|$, is at most $(d + 1.5)k$. Thus Algorithm 5.5 indeed finds a $(d + 1.5)$-approximation for δ-gap-CMSR-d and CMSR-d.

$$2dk \geq \sum_{x \text{ single-marker}} \gamma(x) \quad \text{by (5.1)}$$

$$
\begin{aligned}
&= \sum_{x\in D-\mathcal{K}} \gamma(x) + \sum_{x\in S-\mathcal{K}} \gamma(x) + \sum_{x\in R_1} \gamma(x) + \sum_{x\in R_2} \gamma(x) \\
&\geq \sum_{x\in D-\mathcal{K}} \gamma(x) + |S-\mathcal{K}| + |R_1| + 2|R_2| \\
&\geq |S-\mathcal{K}| + 2|R_1| + 2|R_2| \quad \text{by (5.4)}.
\end{aligned}
$$

Hence, $|R_1| + 2|R_2| \leq \frac{1}{2}|S-\mathcal{K}|$.

$$
\begin{aligned}
|D| + |R| &= |D| + |R_1| + |R_2| + |R\cap\mathcal{K}| \\
&\leq |D| + dk - \tfrac{1}{2}|S-\mathcal{K}| + |R\cap\mathcal{K}| \\
&= |D| + dk - \tfrac{1}{2}(|S| - |S\cap\mathcal{K}|) + |R\cap\mathcal{K}| \\
&\leq |D| + dk - \tfrac{1}{2}|D| + \tfrac{1}{2}|S\cap\mathcal{K}| + |R\cap\mathcal{K}| \quad \text{by (5.2)} \\
&= \tfrac{1}{2}(|D| + |S\cap\mathcal{K}|) + |R\cap\mathcal{K}| + dk \\
&= \tfrac{1}{2}(|D\cap\mathcal{K}| + |D-\mathcal{K}| + |S\cap\mathcal{K}|) + |R\cap\mathcal{K}| + dk \\
&\leq \tfrac{1}{2}(|D\cap\mathcal{K}| + (|\mathcal{K}-D| + |S\cap\mathcal{K}|) + |S\cap\mathcal{K}|) + |R\cap\mathcal{K}| + dk \quad \text{by (5.3)} \\
&= \tfrac{1}{2}|\mathcal{K}| + (|S\cap\mathcal{K}| + |R\cap\mathcal{K}|) + dk \\
&\leq \tfrac{1}{2}k + k + dk \\
&= \left(d + \tfrac{3}{2}\right)k.
\end{aligned}
$$

With the following two examples, we show that the approximation ratio of Algorithm 5.5 cannot be better than $d+1$ (Example 5.3). Moreover, no algorithm deleting only single-markers can achieve an approximation ratio better than d (Example 5.4).

Example 5.3. *The optimal solution of CMSR-d over $\mathcal{M}_{1\ldots d}$ below is to delete the two single-markers u and v, instead of all $2d+2$ single-markers (for Algorithm 5.5):*

$$
\begin{array}{lccccc}
\mathcal{M}_1 = \langle & z_d, y_d, & \cdots & z_3, y_3, & z_2, y_2, & z_1,\ v, u,\ y_1 \rangle \\
\mathcal{M}_2 = \langle & z_1, y_1, & z_2,\ u, v,\ y_2, & z_3, y_3, & \cdots & z_d, y_d \rangle \\
\mathcal{M}_3 = \langle & z_1, y_1, & z_2, y_2, & z_3,\ u, v,\ y_3, & \cdots & z_d, y_d \rangle \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\mathcal{M}_d = \langle & z_1, y_1, & z_2, y_2, & z_3, y_3, & \cdots & z_d,\ u, v,\ y_d \rangle
\end{array}
$$

Example 5.4. *The optimal solution of CMSR-d over $\mathcal{M}_{1\ldots d}$ below is to delete the super-marker $\langle u, v\rangle$, instead of $2d$ single-markers z_i and y_i, $1 \leq i \leq d$ (for any algorithm deleting only single-markers):*

$$
\begin{array}{lccccc}
\mathcal{M}_1 = \langle & z_d, y_d, & \cdots & z_3, y_3, & z_2, y_2, & z_1,\ -v, -u,\ y_1 \rangle \\
\mathcal{M}_2 = \langle & z_1, y_1, & z_2,\ u, v,\ y_2, & z_3, y_3, & \cdots & z_d, y_d \rangle \\
\mathcal{M}_3 = \langle & z_1, y_1, & z_2, y_2, & z_3,\ u, v,\ y_3, & \cdots & z_d, y_d \rangle \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\mathcal{M}_d = \langle & z_1, y_1, & z_2, y_2, & z_3, y_3, & \cdots & z_d,\ u, v,\ y_d \rangle
\end{array}
$$

Compared to the approximation upper bound of $2d$ (Section 5.1) for the maximization problem MSR-d which almost matches – at least asymptotically – the current best lower bound of $\Omega(d/\log d)$ [95], our upper bound of $d+1.5$ for the two minimization problems CMSR-d and δ-gap-CMSR-d is still far away from the constant lower bound in [95]. It is an intriguing question whether CMSR-d and δ-gap-CMSR-d admit approximation algorithms with constant ratios independent of d.

Algorithm 5.6 $2^{O(\ell d\delta)}n$ FPT algorithm for δ-gap-MSR-d.

Input: d genomic maps $\mathcal{M}_{1\ldots d}$ without duplicates, $\delta \in \mathbb{N}$

1: $\Omega_2 \leftarrow$ the set of length-2 prestrips
2: **for all** marker u **do**
3: create a boolean variable x_u
4: **end for**
5: **for all** $\langle u, v \rangle \in \Omega_2$ **do**
6: $g_1, \ldots, g_s \leftarrow$ the markers in $\mathsf{gap}(u, v)$
7: create a boolean formula $f_{u,v} = x_u \wedge x_v \wedge \neg x_{g_1} \wedge \ldots \wedge \neg x_{g_s}$
8: **end for**
9: Delete the variables that do not appear in any formula or appear only in negative form in the formulas.
10: Enumerate all possible assignments to the remaining variables to find an optimal assignment that maximizes the number of variables appearing in positive form in at least one satisfied formula. Delete all markers whose variables are not assigned true values.
11: **return** the resulting genomic maps.

5.4 Fixed-Parameter Tractable Algorithms

5.4.1 FPT Algorithm for δ-gap-MSR-d

In this section, we present the first FPT algorithm for δ-gap-MSR-d with the parameter ℓ. Recall that without the gap constraint, MSR-d with the parameter ℓ is W[1]-hard for any $d \geq 4$. In sharp contrast to the W[1]-hardness of MSR-d, we obtain a somewhat surprising result that δ-gap-MSR-d is in FPT, where ℓ is the parameter, and δ and d are constants. In fact, our FPT algorithm for δ-gap-MSR-d works even if d and δ are not constants: δ-gap-MSR-d is in FPT even with three combined parameters d, δ and ℓ.

Theorem 5.38. *Algorithm 5.6 finds an optimal solution for* δ-gap-MSR-d *for any* $d \geq 2$ *and* $\delta \geq 1$, *in time* $O(2^t td\delta^2 + nd\delta)$, *where* $t = \ell(1 + \frac{3}{2}d\delta)$.

Our algorithm is based on a simple idea: create a boolean variable for each marker (where true means the marker is selected in a solution, false that it is unselected), then test all possible assignments to find an optimal solution. To reduce the time complexity of this brute-force approach, we add a pruning step (line 9) to delete certain variables whose markers cannot appear in any optimal solution. The remaining variables form a "core" on which we can find an optimal solution in FPT time.

The correctness of the algorithm is deduced from the fact that each marker selected in a solution corresponds to a variable appearing in positive form in at least one formula, thus all optimal solutions are kept during the pruning step (line 9), and are discovered during the exhaustive enumeration (line 10).

Given an optimal solution, which selects ℓ markers, we call a marker *active* if it appears within distance at most δ from a selected marker in some map. Then

each map contains at most $\ell\delta + \frac{\ell}{2}\delta$ unselected active markers: at most δ after each selected marker, and at most δ before the first marker of each strip (note that the number of strips of this optimal solution is at most $\ell/2$). The total number of active markers is at most $\ell + d(\ell\delta + \frac{\ell}{2}\delta) = \ell(1 + \frac{3}{2}d\delta)$.

The pruning step in line 9 depends on the crucial observation that a non-active marker can never appear in positive form. Suppose for contradiction that a non-active marker u appears in a prestrip with some marker v. Then u is at distance at most $\delta+1$ from v in each map. Since u, as a non-active marker, must be at distance at least $\delta+1$ from the selected markers in all maps, no selected markers can appear between u and v in any map, thus we can extend the optimal solution by selecting both u and v, a contradiction.

Note that in line 9 the variables appearing at least once in positive form are never deleted, hence no formula becomes empty after deleting the variables that appear only in negative form. After line 9, the number of remaining variables is at most the number of active markers, which is at most $t = \ell(1 + \frac{3}{2}d\delta)$. Correspondingly, the number of formulas is at most $t(\delta + 1)$, because any candidate pair consists of an active marker and one of the $\delta + 1$ markers immediately following it in the first map. Each formula contains at most $d\delta + 2$ variables.

The time complexity of line 1 is $O(nd\delta)$. In lines 2–8, the variables can be created in time $O(n)$, and the formulas can be created in time $O(t(\delta+1)(d\delta+2)) = O(td\delta^2)$. Similarly, line 9 can be executed in time $O(n + td\delta^2)$. Finally, line 10 can be executed in time $O(2^t t(\delta + 1)(d\delta + 2)) = O(2^t td\delta^2)$, so the overall time complexity is $O(2^t td\delta^2 + nd\delta)$.

5.4.2 FPT Algorithm for CMSR-d and δ-gap-CMSR-d

In this section, we design an FPT algorithm for CMSR-d and δ-gap-CMSR-d, where the parameter is k, the number of deleted markers in the optimal solution.

Since super-markers are already strips in the input genomic maps, one may naturally be tempted to come up with the following algorithm. First, find all super-markers, and add them to the solution. Then, delete a subset of single-markers until all markers in the resulting maps can be partitioned into strips. The correctness of this algorithm for finding an exact solution, however, depends on the assumption that in some optimal solution no super-marker needs to be deleted, which is false as can be seen in the following counter-example:

$$\begin{aligned} \mathcal{M}_1 &= \langle 4, \quad 1, \quad 2, \quad 3, \quad 5, \quad 6, \quad 7 \rangle \\ \mathcal{M}_2 &= \langle 6, \ -3, \ -2, \ -1, \quad 7, \quad 4, \quad 5 \rangle \end{aligned}$$

Here $\langle 1, 2, 3 \rangle$ forms a super-marker, but the optimal solution deletes $\langle 1, 2, 3 \rangle$ and selects $\langle 4, 5 \rangle$ and $\langle 6, 7 \rangle$ instead. An easy generalization of this counter-example shows that any super-marker of size strictly less than $2d$ is not guaranteed to be always selected in some optimal solution. Note that on the other hand, longer super-markers, of size at least $2d$, are always selected in some optimal solution, see e.g. [89, Lemma 1].

We observe that an FPT algorithm for CMSR-d and δ-gap-CMSR-d can be easily obtained using the bounded search tree method. In any feasible solution for the two problems, a single-marker x must be either deleted or selected. If x is selected, then at least one of its neighbors must be deleted. Since x has at most $2d$ neighbors (at most two in each map), this leads to a very simple algorithm running in time

$O((2d+1)^k \operatorname{poly}(nd))$. Parallel to our work, Jiang et al. [89] presented an FPT algorithm running in time $O(3^k \operatorname{poly}(nd))$. We next describe a carefully tuned FPT algorithm running in time $O(2.36^k \operatorname{poly}(nd))$. Jiang & Zhu [90] recently proposed a kernelization for CMSR-2, that is, a set of reduction rules transforming any instance with optimal value k into an instance of size at most $84k$ with equal optimal value. This kernelization yields – using Algorithm 5.7 – an FPT algorithm running in time $O(2.36^k k^2 + n^2)$.

For convenience, we consider the decision problem associated with CMSR-d and δ-gap-CMSR-d, for which the parameter k is part of the input.

Theorem 5.39. *Algorithm 5.7 finds an exact solution for the decision problem associated with δ-gap-CMSR-d, for any $\delta \in \mathbb{N} \cup \{\infty\}$ and $d \geq 2$, in time $O(c^k \operatorname{poly}(nd))$, where $c < 2.36$ is the unique real root of the equation $2c^{-1} + 2c^{-3} = 1$.*

It is interesting to note that although the two problems MSR-d and δ-gap-MSR-d have very different complexities when parameterized by ℓ, their complements CMSR-d and δ-gap-CMSR-d are both tractable when parameterized by k.

We describe the intuition behind Algorithm 5.7. As already noted, we will explore a bounded search tree as follows: in each node we consider a single-marker x, and we explore the branches corresponding to the cases where x is deleted and where it is selected in a strip with each possible candidate successor or predecessor. This search tree has bounded depth (in each branch we delete at least one marker, and we stop after deleting k markers) and degree (each single-marker has at most $2d$ candidate successors or predecessors). In order to improve the complexity of this algorithm, we aim at (1) choosing x so that we may delete a maximum number of markers in the subsequent recursive calls (thus reducing the depth of the subtree), and (2) pointing out special cases where we may ignore some branches of the search tree without losing an optimal solution (thus reducing the degree). For objective (1) we choose x as the first single-marker in the first map, hence, the gap between x and a candidate predecessor consists mostly of super-markers, thus increasing the number of markers to be deleted in the corresponding branches. For objective (2), we provide a number of technical lemmas (Lemma 5.40 to Lemma 5.43) which allow us to reduce the degree of some "worst-case" nodes. In some situations, we find a marker which is necessarily deleted. In others, we identify a good candidate predecessor or successor, which we may select to generate a solution at least as good as with any other candidate.

Some technical lemmas

The efficiency of Algorithm 5.7 is made possible by several optimizations justified by the following four lemmas. These lemmas are all based on very simple observations. Note that although we consider the decision problem for simplicity, Algorithm 5.7 can be adapted to directly return the actual solution, instead of "true", when the input instance indeed has a solution of size at most k. Recall that the relation $\prec$ in lines 16–17 is defined for markers in the original maps — it remains unchanged through recursive calls, and can be precomputed.

Lemma 5.40. *Let x be a single-marker and w a super-marker. If x is selected in an optimal solution, and w is a candidate successor or predecessor of x with exactly one marker in $\mathsf{gap}(x, w)$, then there is an optimal solution where the marker in $\mathsf{gap}(x, w)$ is deleted.*

Algorithm 5.7 $O(2.36^k \operatorname{poly}(nd))$ FPT algorithm for δ-gap-CMSR-d and CMSR-d.

Input: d genomic maps $\mathcal{M}_{1\ldots d}$ without duplicates, parameters $k \in \mathbb{N}$, $\delta \in \mathbb{N} \cup \{\infty\}$

1: **return** recurse($\mathcal{M}_{1\ldots d}, k, \delta$, false)

Function recurse($\mathcal{M}_{1\ldots d}, k, \delta$, skip_step_2b): boolean

1: Partition the markers into single-super-markers.
2: $x \leftarrow$ the left-most single-marker in $\mathcal{M}_1$ (if it exists)
3: **if** $k < 0$ **or** ($k = 0$ **and** x exists) **then**
4: **return** false
5: **end if**
6: **if** x does not exist **then**
7: **return** true
8: **end if**
9: $s \leftarrow$ the first single-super-marker following x in $\mathcal{M}_1$ (if it exists)
10: *// 1: Assume x is deleted in the optimal solution*
11: Create $\mathcal{M}'_{1\ldots d}$ by removing x from $\mathcal{M}_{1\ldots d}$.
12: **if** recurse($\mathcal{M}'_{1\ldots d}, k-1, \delta$, false) **then**
13: **return** true
14: **end if**
15: *// 2: Assume x is part of a strip in the optimal solution*
16: $Y \leftarrow \{$ single-super-marker $y \mid x \prec y\}$ *// the set of candidate successors*
17: $Z \leftarrow \{$ super-marker $z \mid z \prec x\}$ *// the set of candidate predecessors*
18: **if** $\exists w_0 \in Y \cup Z$ a super-marker s.t. (x, w_0) satisfies the conditions of Lemma 5.40 **then**
19: Create $\mathcal{M}'_{1\ldots d}$ by removing the marker in gap(x, w_0) from $\mathcal{M}_{1\ldots d}$.
20: **return** recurse($\mathcal{M}'_{1\ldots d}, k-1, \delta$, false)
21: **end if**
22: **if** $\exists s_0$ a single-marker s.t. (x, s_0) satisfies the conditions of Lemma 5.41 **then**
23: Create $\mathcal{M}'_{1\ldots d}$ by removing s_0 from $\mathcal{M}_{1\ldots d}$.
24: **return** recurse($\mathcal{M}'_{1\ldots d}, k-1, \delta$, false)
25: **end if**
26: *// 2.a: Assume x is not at the end of its strip*
27: **if** $Y \neq \emptyset$ **then**
28: **if** recurse_2a($Y, x, \mathcal{M}_{1\ldots d}, k, \delta$) **then**
29: **return** true
30: **end if**
31: **end if**
32: *// 2.b: Assume x is at the end of its strip*
33: **if** $Z \neq \emptyset$ **and** skip_step_2b=false **then**
34: **if** recurse_2b($Z, x, s, \mathcal{M}_{1\ldots d}, k, \delta$) **then**
35: **return** true
36: **end if**
37: **end if**
38: **return** false

Algorithm 5.7, continued

Function recurse_2a($Y, x, \mathcal{M}_{1\ldots d}, k, \delta$): boolean

if $\exists y_0 \in Y$ s.t. y_0 satisfies the conditions of Lemma 5.42 **then**
  if $\delta \in \mathbb{N}$ **and** y_0 is a single-marker **then**
    Replace y_0 by the unspecified marker $[y_0 \mid Y]$.
  end if
  $Y_0 \leftarrow \{y_0\}$
else
  $Y_0 \leftarrow Y$
end if
for all $y \in Y_0$ **do**
  Create $\mathcal{M}'_{1\ldots d}$ by removing all markers in $\mathsf{gap}(x, y)$ from $\mathcal{M}_{1\ldots d}$.
  if recurse($\mathcal{M}'_{1\ldots d}, k - |\mathsf{gap}(x, y)|, \delta$, false) **then**
    return true
  end if
end for
return false

Function recurse_2b($Z, x, s, \mathcal{M}_{1\ldots d}, k, \delta$): boolean

if $\exists z_0 \in Z$ s.t. z_0 satisfies the conditions of Lemma 5.43 **then**
  $Z_0 \leftarrow \{z_0\}$
else
  $Z_0 \leftarrow Z$
end if
for all $z \in Z_0$ **do**
  if z ends with an unspecified marker $[y_0 \mid Y]$ **and** $\exists y_1 \in Y$ s.t. $y_1 \prec x$ **then**
    Replace the unspecified marker $[y_0 \mid Y]$ by y_1.
  end if
  Create $\mathcal{M}'_{1\ldots d}$ by removing all markers in $\mathsf{gap}(x, z)$ from $\mathcal{M}_{1\ldots d}$.
  skip_next_step_2b $\leftarrow$ s exists **and** s is a single-marker **and** $s \notin \mathsf{gap}(x, z)$
  if recurse($\mathcal{M}'_{1\ldots d}, k - |\mathsf{gap}(x, z)|, \delta$, skip_next_step_2b) **then**
    return true
  end if
end for
return false

Proof. Assume that w is a candidate successor of x, the case where it is a candidate predecessor being symmetric.

Let v be the single-marker such that $\mathsf{gap}(x, w) = \{v\}$, i.e., in some map $\mathcal{M}_i$, one of $\langle +x, \pm v, +w \rangle$ or $\langle -w, \pm v, -x \rangle$ appears. In the case where no optimal solution selects both x and v, the lemma is obviously true since in any solution where x is selected, v must be deleted. It remains to consider the cases where an optimal solution $\mathcal{O}$ exists such that both x and v are selected. Since any strip of length $p \geq 4$ can be split into two shorter strips of lengths 2 and $(p-2)$, we can assume wlog. that all strips have lengths 2 or 3.

First case: x and v appear in the same strip. Then v is a candidate successor of x and w is not selected in $\mathcal{O}$, since by Lemma 5.1a, $w \in \mathsf{gap}(x, v)$. Create $\mathcal{O}'$ by removing this strip from $\mathcal{O}$, the total size decreases by at most 3. Then no marker in $\{x, w\} \cup \mathsf{gap}(x, w)$ is selected in $\mathcal{O}'$: we can add the strip $\langle x, w \rangle$ to obtain a feasible solution of size greater than or equal to that of $\mathcal{O}$, since w is a super-marker, where v is deleted.

Second case: x and v appear in different strips. Looking at map $\mathcal{M}_i$, we see that v is at one end of its strip, and either (a) w is deleted or (b) w is in the same strip as v, and v precedes w. In case (a) we delete v (plus a second marker if v is in a length-2 strip), and add w at the end of the strip containing x: we again have an optimal solution where v is deleted. We now show that case (b) is absurd: since w is a candidate successor of both x and v, then x and v are candidate predecessors of w. However, by Lemma 5.1a, $x \in \mathsf{gap}(w, v)$, so this contradicts the fact that x is selected and w, v are in the same strip. □

Lemma 5.41. *Let x be a single-marker and s a single-super-marker. If s appears in $\mathsf{gap}(x, w)$ for each w that is a candidate successor or predecessor of x, then s itself cannot be a candidate successor or predecessor of x, and any solution selecting x deletes s.*

Proof. First, if s was a candidate successor or predecessor of x, we would have $s \in \mathsf{gap}(x, s)$, which is impossible. Consider now the strip containing x in any feasible solution. If x is at the end of this strip, it is preceded by a candidate predecessor z, $z \neq s$, and all markers in $\mathsf{gap}(x, z)$, including s, are deleted. Otherwise x is followed in its strip by a candidate successor y, and again $s \in \mathsf{gap}(x, y)$ is deleted. □

Lemma 5.42. *(In this lemma we assume there is no gap constraint.) Let x be a single-marker and y a candidate successor of x such that all markers in $\mathsf{gap}(x, y)$ are single-markers and candidate successors of x. If x is part of some strip in an optimal solution, but not at the end of this strip, then there is an optimal solution where $\langle x, y \rangle$ is part of some strip.*

Proof. Let y_0 be the single-super-marker following x in the strip of the optimal solution, and y_1 be the successor of y_0 if it exists. If $y = y_0$, then the lemma is proved. Otherwise, $y_0 \in \mathsf{gap}(x, y)$ (by Lemma 5.1a) and y_0 is a single-marker.

If y_1 does not exist, we can replace y_0 by y if we delete all markers in $\mathsf{gap}(x, y) - \{y_0\}$. But since all these markers are candidate successors of x, they also appear in $\mathsf{gap}(x, y_0)$ and are already deleted, hence the total size of the solution is unchanged.

Assume now that y_1 exists, we prove that y_1 is a candidate successor of y. First of all, x, y, y_0, y_1 appear in the same sequence of gene markers (in the same chromosome) in each map. Moreover, x, y, y_1 appear in this order in all maps: y and y_1 both appear after x, and y_1 cannot appear in any $S^i(x, y)$, otherwise $y_1 \in \mathsf{gap}(x, y)$

and y_1 would be a candidate successor of x (which is absurd, since $y_0 \in S^j(x, y_1)$ for all j). Since there is no gap constraint, y_1 is a candidate successor of y. We can replace y_0 by y if we delete all markers in $\Gamma = (\mathsf{gap}(x, y) \cup \mathsf{gap}(y, y_1)) - \{y_0\}$:

$$\begin{aligned}\Gamma &= \left(\bigcup_i S^i(x, y) \cup S^i(y, y_1)\right) - \{y_0\} \\ &= \left(\bigcup_i S^i(x, y_1)\right) - \{y, y_0\} \\ &= \left(\bigcup_i S^i(x, y_0) \cup S^i(y_0, y_1)\right) - \{y\} \\ &= (\mathsf{gap}(x, y_0) \cup \mathsf{gap}(y_0, y_1)) - \{y\}.\end{aligned}$$

Then all markers in Γ are already deleted: we can replace y_0 by y without changing the solution size. □

Lemma 5.43. *Let x be the first single-marker in $\mathcal{M}'_1$. Let z be a candidate predecessor of x such that all markers in $\mathsf{gap}(x, z)$ are size-2 super-markers and candidate predecessors of x. If x appears at the end of a strip in an optimal solution, then there is an optimal solution where $\langle z, x \rangle$ is at the end of some strip.*

Proof. Let z_0 be the single-super-marker preceding x in the strip of the optimal solution, and z_1 be the one preceding z_0. If $z = z_0$, the lemma is proved. Otherwise, $z_0 \in \mathsf{gap}(x, z)$, hence it is a size-2 super-marker.

If z_1 exists, then it is also a super-marker, since x is the first single-marker in $\mathcal{M}'_1$, and we can split the strip between z_1 and z_0: hence we can assume that the strip containing x in the optimal solution is $z_0 x$.

We can replace z_0 by z in $z_0 x$ by deleting all markers in $\mathsf{gap}(x, z) - \{z_0\}$. Since all these markers appear in $\mathsf{gap}(x, z_0)$, they are already deleted in the optimal solution. Moreover, $|z| \geq 2 = |z_0|$, so replacing z_0 by z does not reduce the solution size. □

In addition to these four optimizations, we also use a "delayed commitment" optimization which is the equivalent of Lemma 5.42 when we need to observe a gap constraint. We consider the case where x is part, but not at the end, of some strip in the optimal solution, and where y is a single-marker and a candidate successor of x such that all markers in $\mathsf{gap}(x, y)$ are single-markers and candidate successors of x. In this case we delete all markers in $\mathsf{gap}(x, y)$ to make $\langle x, y \rangle$ a strip, but keep the possibility of replacing y by any marker $y_1 \in \mathsf{gap}(x, y)$, should necessity arise. We denote this unspecified marker by $[y \mid \mathsf{gap}(x, y)]$.

Correctness of Algorithm 5.7

To prove the correctness of Algorithm 5.7, we use the following lemma from [128]. We provide an easy proof for completeness.

Lemma 5.44. [128, Proposition 2] *We can decompose the strips of any optimal solution in such a way that (1) each strip contains at most 3 single-super-markers and (2) each strip containing 3 single-super-markers starts and ends with a single-marker.*

Proof. Let s be a strip containing h single-super-markers: $s = \langle s_1, s_2, \ldots, s_h \rangle$. If $h \geq 4$, we can split s into two strips: $\langle s_1, s_2 \rangle$ and $\langle s_3, s_4, \ldots, s_h \rangle$. We apply this until condition (1) is true. If $h = 3$ and s_1 (respectively s_3) is a super-marker, then we can split s into $\langle s_1 \rangle$ and $\langle s_2, s_3 \rangle$ (respectively $\langle s_1, s_2 \rangle$ and $\langle s_3 \rangle$). We can do this operation until condition (2) also becomes true. □

Let $\mathcal{O}$ be any optimal solution. Decompose the strips of $\mathcal{O}$ as in the above lemma. We show by induction that the solution found by Algorithm 5.7 has the same size as $\mathcal{O}$. Lines 1–8 of function recurse deal with the trivial cases where $k \leq 0$ or there are no more single-markers (in $\mathcal{M}_1$ or in any other map): in this case, a solution exists iff $k \geq 0$ and there are no more single-markers. We suppose now that $k \geq 1$ and that there exists a left-most single-marker, x, in $\mathcal{M}_1$. Then exactly one of the following three cases is true:

1: x is deleted in $\mathcal{O}$,

2.a: There exists a single-super-marker y such that $\langle x, y \rangle$ is part of a strip in $\mathcal{O}$,

2.b: There exists a super-marker z such that $\langle z, x \rangle$ is a strip in $\mathcal{O}$.

Note that in case 2.b, z cannot be a single-marker since it is to the left of x in $\mathcal{M}_1$. By our choice of x, case 2.a can be split into the following two subcases:

2.a.i: There exists a single-super-marker y such that $\langle x, y \rangle$ is a strip in $\mathcal{O}$,

2.a.ii: There exists a single-super-marker y and a single-marker y' such that $\langle x, y, y' \rangle$ is a strip in $\mathcal{O}$.

Refer to Algorithm 5.7. In case 1, a solution is found in lines 10–14 of the function recurse. In case 2, i.e. in the case where x is part of an optimal solution, if either Lemma 5.40 or Lemma 5.41 can be applied, then again a solution is found. Otherwise, we are in case 2.a or 2.b.

Suppose we are in case 2.a. If $y \in Y_0$, then the function recurse_2a tests a branch in which $\langle x, y \rangle$ becomes part of some strip. Otherwise, there exists some $y_0 \in Y$ satisfying the conditions of Lemma 5.42. If there is no gap constraint, y is replaced by y_0, which does not change the size of the solution. If there is a gap constraint, y is replaced by the unspecified marker $u = [y_0 \mid Y]$, and we look further in case 2.a.i or 2.a.ii.

In case 2.a.i, we can replace y by y_0 since $\mathsf{gap}(x, y_0)$ has no more markers than $\mathsf{gap}(x, y)$. In case 2.a.ii, we can replace y by any y_1 such that $x \prec y_1 \prec y'$, since $\mathsf{gap}(x, y) \cup \{y\} \cup \mathsf{gap}(y, y')$ is the same set as $\mathsf{gap}(x, y_1) \cup \{y_1\} \cup \mathsf{gap}(y_1, y')$. This is what happens in case 2.b of a subsequent recursive call in which y' becomes the left-most single-marker in $\mathcal{M}_1$.

Suppose we are in case 2.b. If $z \in Z_0$, then the function recurse_2b tests a branch in which $\langle z, x \rangle$ becomes a strip. Otherwise, Lemma 5.43 can be applied, which leaves the size of the optimal solution unchanged. In line 11 of recurse_2b, if s becomes the left-most single-marker in $\mathcal{M}_1$ in the next recursive call of recurse, it cannot be at the end of a strip because x is already at the end of a strip.

This completes the correctness proof.

An example on the behavior of the unspecified markers

We run Algorithm 5.7 on the following three maps, with the gap constraint $\delta = 3$:

$$\mathcal{M}_1 = \langle 1, 2, a, 4, 3, r, b \rangle$$
$$\mathcal{M}_2 = \langle 1, 3, 2, a, 4, b, r \rangle$$
$$\mathcal{M}_3 = \langle 1, 4, 3, 2, x, b, a, r \rangle$$

In these maps, 1 has three candidate successors: 2, 3 and 4. Moreover, $\mathsf{gap}(1, 2) = \{3, 4\}$. Thus, in part (2.a) of Algorithm 5.7, only one branch is considered: 3 and 4 are deleted, and 2 is replaced by the unspecified marker $[2 \mid 2, 3, 4]$. In the subsequent recursive call, the three maps start with a size-2 super-marker.

$$\mathcal{M}_1 = \langle (1, [2 \mid 2, 3, 4]), a, r, b \rangle$$
$$\mathcal{M}_2 = \langle (1, [2 \mid 2, 3, 4]), a, b, r \rangle$$
$$\mathcal{M}_3 = \langle (1, [2 \mid 2, 3, 4]), x, b, a\ r \rangle$$

The new first single-marker is a. In part (2.b), $(1,\ [2 \mid 2, 3, 4])$ is a candidate predecessor of a with 2, and the set $\mathsf{gap}(a, [2 \mid 2, 3, 4])$ is $\{x, b\}$. In this branch of the search tree, we obtain

$$\mathcal{M}_1 = \mathcal{M}_2 = \mathcal{M}_3 = \langle (1\ 2\ a)\ r \rangle,$$

where r is a candidate successor of a. Hence the algorithm finds the solution consisting of the length-4 strip $\langle 1, 2, a, r \rangle$.

In another branch of the search tree, where a and r are deleted, we obtain the following maps:

$$\mathcal{M}_1 = \langle (1, [2 \mid 2, 3, 4]), b \rangle$$
$$\mathcal{M}_2 = \langle (1, [2 \mid 2, 3, 4]), b \rangle$$
$$\mathcal{M}_3 = \langle (1, [2 \mid 2, 3, 4]), x, b \rangle$$

Here b is the left-most single-marker in $\mathcal{M}_1$, and $(1\ [2 \mid 2, 3, 4])$ is a candidate predecessor of b with 3 and 4 (not with 2, since the gap between 2 and b in the original maps is $4 > \delta$). Hence part (2.a) of Algorithm 5.7 deletes the markers in $\mathsf{gap}(b, [2 \mid 2, 3, 4]) = \{x\}$, and replaces the unspecified marker $[2 \mid 2, 3, 4]$, e.g. by 3. Thus Algorithm 5.7 finds in another branch of the search tree the length-3 strip $\langle 1, 3, b \rangle$.

Complexity analysis of Algorithm 5.7

Let $T(k)$ be the complexity of the function recurse of Algorithm 5.7 with parameters k and skip_step_2b=false, and $T_{skip}(k)$ the complexity of this function with parameters k and skip_step_2b=true (the complexity here being the number of leaves in the search tree). The complexity of several parts of the algorithm depends on whether the single-super-marker s defined at line 9 is a single-marker: so we define a boolean variable s_single, which is true if s exists and is a single-marker, and false otherwise. We now compute the complexity of each part of the algorithm.

Part 1: The complexity from line 10 to 14 is $T(k-1)$.

Part 2 (lines 15 to 37): if one of the conditions from lines 18 and 22 is true, then the complexity here is $T(k-1)$. Otherwise, we need to analyze the complexity of parts 2.a (lines 26 to 31) and 2.b (lines 32 to 37).

Part 2.a: We write r for the number of single-super-markers in Y_0, and r' for the minimum size of $\mathsf{gap}(x,y)$ for $y \in Y_0$, and y_0 the single-super-marker reaching this bound; then the complexity is at most $rT(k-r')$. We now bound r and r': first, by Lemma 5.1a, we already have $r' \geq r-1$. We now prove by contradiction that $r' > r-1$. Assume that $r' = r-1$, then the candidate successors of $Y - \{y_0\}$ are the only markers appearing in $\mathsf{gap}(x,y_0)$, and they are all single-markers. Thus y_0 satisfies the conditions of Lemma 5.42, and $Y_0 = \{y_0\}$, $r=1$ and $r'=0$ (even if y_0 is replaced by an unspecified marker in the meantime). This is absurd, since $r' = |\mathsf{gap}(x,y_0)|$ and $\mathsf{gap}(x,y_0)$ is not empty by Lemma 5.1b. Thus $r' \geq r$ and the complexity of part 2.a is upper bounded by $r'T(k-r')$ with $r' \geq 1$.

Moreover, if s_single is false, then we show that we cannot have $r'=1$. By contradiction again, suppose $r'=1$. The super-marker s exists (otherwise no marker follows x in $\mathcal{M}_1$, so $Y=\emptyset$), and it appears in $\mathsf{gap}(x,y)$ for all $y \in Y - \{s\}$, then we necessarily have $y_0 = s$ and $|\mathsf{gap}(x,y_0)| = 1$. This would mean that (x,y_0) satisfies the conditions of Lemma 5.40, a contradiction.

Thus the complexity of part 2.a is at most $\max\{r'T(k-r') \mid r' \geq 1\}$ if s_single is true, and $\max\{r'T(k-r') \mid r' \geq 2\}$ otherwise.

Part 2.b: First note that all $z \in Z$ are super-markers. We denote by t the number of super-markers in Z, and by t' the minimum size of $\mathsf{gap}(x,z)$ for $z \in Z_0$ (we write z_0 for the super-marker reaching this bound). By Lemma 5.1a, $\mathsf{gap}(x,z_0)$ contains at least $t-1$ super-markers, thus $t' \geq 2(t-1)$. Moreover $t' \neq 0$ (Lemma 5.1b) and $t' \neq 1$ (otherwise (x,z_0) would satisfy the conditions of Lemma 5.40); and one cannot have $t' = 2(t-1)$ for $t \geq 2$: if $t' = 2(t-1)$, then z_0 satisfies the conditions of Lemma 5.43, so $Z_0 = \{z_0\}$ and $t=1$.

Hence the complexity of part 2.b is at most $\max\{T(k-2), \max\{tT(k-2t+1) \mid t \geq 2\}\}$. This is the best bound we obtain when s_single is false, but it can be improved when s_single is true.

Indeed, if s_single is true, we consider Z_1 the set of $z \in Z_0$ such that $s \in \mathsf{gap}(x,z)$ and $Z_2 = Z_0 - Z_1$. We can see that Z_2 is not empty: otherwise (x,s) would satisfy the conditions of Lemma 5.41. Several cases are possible:

- $t=1$, then $Z_0 = Z_2$ contains only one super-marker z_0, and the complexity is $T_{skip}(k-2)$.
- $t \geq 2$: For each $z \in Z_1$, $\mathsf{gap}(x,z)$ contains at least $(t-1)$ super-markers from $Z - \{z\}$ and the single-marker s, so the complexity is $T(k-2(t-1)-1)$. For $z \in Z_2$, it is $T_{skip}(k-2(t-1)-1)$.

Overall, the complexity of part 2.b in the case where s_single is true is at most:

$$\max \begin{cases} T_{skip}(k-2), \\ \max\{T_{skip}(k-t') + (t-1)\max\{T(k-t'), T_{skip}(k-t')\} \mid t \geq 2, t' = 2t-1\} \end{cases}$$

We can now show by induction over k that $T(k) \leq c^k$ and $T_{skip}(k) \leq \mu c^k$, with $c \approx 2.3593$ (c is the real positive solution of $1 = 2c^{-1} + 2c^{-3}$) and $\mu = 2/c \approx 0.8477$. We apply the induction hypothesis on the upper bound obtained for the complexity of each part.

- For part 1,

$$T(k-1) \leq c^{k-1},$$

– for part 2.a, with s_single = false,

$$\max\{r'T(k-r') \mid r' \geq 2\} \leq 2c^{k-2},$$

– for part 2.a, with s_single = true,

$$\max\{r'T(k-r') \mid r' \geq 1\} \leq c^{k-1},$$

– for part 2.b, with s_single = false,

$$\max\{T(k-2), \max\{tT(k-2t+1) \mid t \geq 2\}\} \leq c^{k-2},$$

– for part 2.b, with s_single = true,

$$\begin{aligned} \max\{T(k-t'), T_{skip}(k-t')\} &\leq c^{k-t'} \text{ for all } t' \\ \max\{T_{skip}(k-t') + (t-1)c^{k-t'} \mid t \geq 2, t' = 2t-1\} &\leq 2c^{k-4} + c^{k-3} \\ \max\{T_{skip}(k-2), 2c^{k-4} + c^{k-3}\} &\leq 2c^{k-3}. \end{aligned}$$

We sum up the bounds, first for the case where s_single = false (remember that we need to count the complexity of part 2.b only for T, not for T_{skip}):

$$\begin{aligned} T_{skip}(k)c^{-k} &\leq c^{-1} + \max\{c^{-1}, 2c^{-2}\} = 2c^{-1} \;=\; \mu \\ T(k)c^{-k} &\leq c^{-1} + \max\{c^{-1}, 2c^{-2} + c^{-2}\} = 0.962\ldots < 1 \end{aligned}$$

Next for s_single = true:

$$\begin{aligned} T_{skip}(k)c^{-k} &\leq c^{-1} + \max\{c^{-1}, c^{-1}\} = 2c^{-1} \;=\; \mu \\ T(k)c^{-k} &\leq c^{-1} + \max\{c^{-1}, c^{-1} + 2c^{-3}\} = 1 \end{aligned}$$

Thus we have $T(k) \leq c^k$ and $T_{skip}(k) \leq \mu c^k$. This proves that the size of the search tree is bounded by $O(c^k)$, and each recursive call is done in polynomial time in n and d, so, altogether, the running time of Algorithm 5.7 is $O(c^k \operatorname{poly}(dn))$.

5.4.3 FPT Algorithm for 1-gap-CMSR-d

In this section we present an improvement of Algorithm 5.7 for the problem 1-gap-CMSR-d.

Theorem 5.45. *Algorithm 5.8 finds an exact solution for the decision problem associated with* 1-gap-CMSR-d*, for any* $d \geq 2$*, in time* $O(2^k \operatorname{poly}(nd))$.

Note that, as for Algorithm 5.7, Algorithm 5.8 can be easily adapted to produce a full solution instead of simply returning "true", when the instance indeed has a solution of the right size.

We first prove the following two lemmas.

Lemma 5.46. *(This lemma uses the gap constraint* $\delta = 1$*.) Let* x *be a single-marker, and* z *a super-marker candidate predecessor of* x*. Then if an optimal solution selects* x*, it also deletes all markers in* $\mathsf{gap}(x, z)$.

Algorithm 5.8 $O(2^k \operatorname{poly}(nd))$ FPT algorithm for 1-gap-CMSR-d.

Function 1-gap-CMSR($\mathcal{M}_{1...d}, k$): boolean

if $k < 0$ **then**
  return false
end if
Partition the markers into single-super-markers.
if there exists at least one single-marker in $\mathcal{M}_1$ **then**
  $x \leftarrow$ the left-most single-marker in $\mathcal{M}_1$
else
  return true
end if
// 1: Assume x is deleted in the optimal solution
Create $\mathcal{M}'_{1...d}$ by removing x from $\mathcal{M}_{1...d}$.
if 1-gap-CMSR($\mathcal{M}'_{1...d}$, $k-1$) **then**
  return true
end if
// 2: Assume x is selected in the optimal solution
if $\exists z_0 \prec x$ **then**
  Create $\mathcal{M}'_{1...d}$ by removing all markers in $\mathsf{gap}(x, z_0)$ from $\mathcal{M}_{1...d}$.
  return 1-gap-CMSR($\mathcal{M}'_{1...d}, k - |\mathsf{gap}(x, z_0)|$)
else if $\exists a \succ x$ **then**
  if $\exists b \succ x$ s.t. $b \neq a$ **then**
    $y_0 \leftarrow \mathsf{choose}(\mathcal{M}'_{1...d}, a, b)$
  else
    $y_0 \leftarrow a$
  end if
  Create $\mathcal{M}'_{1...d}$ by removing all markers in $\mathsf{gap}(x, y_0)$ from $\mathcal{M}_{1...d}$.
  return 1-gap-CMSR($\mathcal{M}'_{1...d}, k - |\mathsf{gap}(x, y_0)|$)
else
  return false
end if

Function $\mathsf{choose}(\mathcal{M}_{1...d}, a, b)$: single-marker

if $\exists a' \succ a$ **then**
  if $\exists b' \succ b$ **and** $b' \neq a'$ **then**
    if $\mathsf{choose}(\mathcal{M}_{1...d}, a', b') = a'$ **then**
      return a
    else
      return b
    end if
  else
    return a
  end if
else
  return b
end if

Proof. With the gap constraint, z is the only candidate predecessor of x, and it is selected in the optimal solution (like all super-markers).

Take $u \in \mathsf{gap}(x, z)$, then u cannot be in the same strip as x (it is neither a candidate successor nor predecessor of x). Hence if u is selected in the optimal solution, then it is in the same strip as z and all markers of $\mathsf{gap}(z, u)$, including x (see Lemma 5.1a), are deleted: a contradiction. So all markers in $\mathsf{gap}(x, z)$ are deleted in the optimal solution. □

Lemma 5.47. *(This lemma uses the gap constraint $\delta = 1$.) Let x be a single-marker with two candidate successors a and b. If x appears in an optimal solution, but not at the end of its strip, then there is an optimal solution where $\langle x, c\rangle$ is part of some strip, with $c = \mathsf{choose}(\mathcal{M}_{1\ldots d}, a, b)$ (see Algorithm 5.8).*

Proof. For simplicity, we assume wlog. that x has a positive sign in all maps. Otherwise, if x has a negative sign in some map $\mathcal{M}_i$, we can replace this map by its reversed opposite.

If $\delta = 1$ and x has two candidate successors a_1 and b_1, then in each map we have the sequence $\langle x, a_1, b_1\rangle$ or $\langle x, b_1, a_1\rangle$. Moreover, if both a_1 and b_1 have at least one candidate successor (respectively a_2 and b_2) with $a_2 \neq b_2$, then again only two patterns are possible in all maps: $\langle x, a_1, b_1, a_2, b_2\rangle$ or $\langle x, b_1, a_1, b_2, a_2\rangle$. We proceed with this construction recursively, until we reach a pair (a_h, b_h) such that a_h and b_h do not have different candidate successors.

Assume that x is selected in a strip of an optimal solution, followed by h' markers in this strip, with $1 \leq h' \leq h$. Then these markers are either $\langle a_1, \ldots, a_{h'}\rangle$ or $\langle b_1, \ldots, b_{h'}\rangle$, and we can replace one sequence by the other without creating overlapping strips, so there are optimal solutions selecting $\langle x, c\rangle$ for $c = a_1$ and for $c = b_1$.

If $h' > h$, let u be the $h + 1^{\text{st}}$ marker following x in the strip (and assume wlog that the first h selected markers are $\langle b_1, \ldots b_h\rangle$). Then u is a candidate successor of b_h, and either a_h has no candidate successor, or it has only u. In the first case, $\mathsf{choose}(a_h, b_h) = b_h$, and $\mathsf{choose}(a_1, b_1) = b_1$: this is the choice made in the optimal solution. In the second case, $\mathsf{choose}(a_1, b_1) = a_1$, but in the strip of the optimal solution, we can replace $\langle x, b_1, \ldots, b_h, u\rangle$ by $\langle x, a_1, \ldots, a_h, u\rangle$ without creating incompatibilities, since we have $\mathsf{gap}(a_h, u) = \{b_h\}$ and $\mathsf{gap}(b_h, u) = \{a_h\}$. Thus there is also an optimal solution selecting $a_1, \ldots, a_h, u$ after x: this proves the lemma. □

We now turn to the proof of Theorem 5.45. Algorithm 5.8 is based on the following observation. Let x be the left-most single-marker in $\mathcal{M}_1$, and assume it appears in an optimal solution, then there are two cases:

(1) x has a candidate predecessor z_0 (it is necessarily a super-marker). Then, by Lemma 5.46, we can delete all markers between x and z_0 in all maps, regardless of whether z_0 and x are in the same strip in the optimal solution. At least one such marker must exist.

(2) x has no candidate predecessor, then it must be in the same strip as a successor. With the gap constraint, x can have at most two successors. Using Lemma 5.47, we can choose one of them (y_0, in Algorithm 5.8).

This proves the correctness of the algorithm. Moreover, the complexity of the 1-gap-CMSR function with parameter k is at most $O(2^k \operatorname{poly}(nd))$: it is polynomial except for at most two recursive calls, each with a parameter $k' < k$. Thus Theorem 5.45 is proved.

Conclusion

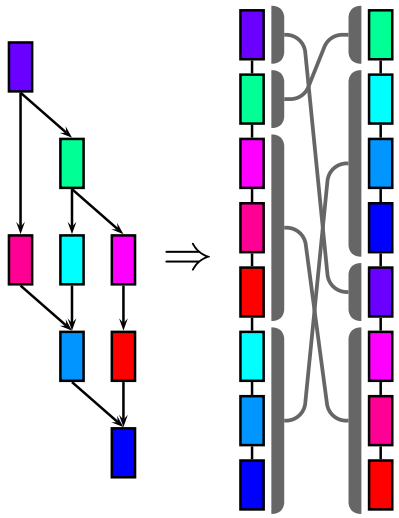
In this chapter, we have extensively studied the variants of Maximal Strip Recovery; it appears that this problem is intractable even for very constrained parameters. However, we are able to provide a number of efficient algorithms, either in the form of approximations or FPT algorithms, in order to come as close as possible to the intractability bounds. We have given a more specific focus on the δ-gap variant which usually allows for more efficient algorithms (see the specific algorithms for 1-gap-MSR and 1-gap-CMSR), while being relevant for biological applications. A key algorithm we have presented is the carefully-tuned FPT algorithm for δ-gap-CMSR-d and CMSR-d.

In terms of approximability, there remain important gaps between the hardness results and the precision of existing algorithms. For example, do 1-gap-MSR and/or 1-gap-CMSR admit polynomial time approximation schemes? Do CMSR-d and δ-gap-CMSR-d admit approximation algorithms with a ratio independent of d? We know that the answer is no for MSR-d and δ-gap-CMSR-d.

Another challenge is to design more efficient algorithms capable of processing sequences with duplicates. For example, creating an efficient FPT algorithms for (δ-gap-) CMSR-DU would be an important positive result for the Maximal Strip Recovery problem.

6

Minimum Breakpoint Linearization



The NP-hard Minimum Breakpoint Linearization problem aims at reconstructing a linear genome from partially ordered data, using the genome of a close species for reference. Formally, the objective is, given a partial order Π, to produce a permutation compatible with Π that minimizes the breakpoint distance to a reference permutation.

In this chapter, we present an approximation algorithm for this problem, and three algorithms obtained from a reduction to Subset-FVS.

The results in this chapter have been presented at the *7th Annual Conference on Theory and Applications of Models of Computation* (TAMC 2010, Prague [32]), and accepted for publication in *Theoretical Computer Science* (TCS 2013 [38]).

Introduction

In a number of comparative genomics algorithms, a full knowledge of the order of the genes on the chromosomes for the species under study is required: this is the case for all the problems studied in the previous chapters. However, despite the rapid advances in DNA sequencing, we have a perfect knowledge of the full gene sequence only for a limited number of species, and for other species, we only have genetic maps, where some uncertainties in the gene order may remain. Hence, the problem of inferring a total order, compatible with the partial knowledge on these genetic maps and optimizing a relevant objective function, is a first step to study nonetheless all genomes. In the past few years, growing attention has been given to this problem, in which the objective function is an evolutionary distance to a reference genome (*e.g.* number of rearrangements [127], reversal [126, 73], breakpoint [73, 23, 44], or common intervals [23] distance).

In this chapter, we focus on the Minimum Breakpoint Linearization (MBL) problem, which aims at finding a linearization of a partial order while minimizing the breakpoint distance to a reference genome. An approach for solving MBL is the construction of an *adjacency-order graph* proposed in [73] and used in [44] to design a heuristic and an approximation algorithm (whose ratio depends on m, the number of genetic maps used to construct the studied genome). However, the construction has a flaw which makes both above mentioned algorithms invalid on general data [38]. Thus, we introduce in this chapter a new type of adjacency-order graphs, and prove that it is effective to solve the MBL problem. This renewed approach allows us to use general graph theory results [67, 49] to obtain new approximation and fixed-parameter tractable algorithms for MBL. Moreover, we also achieve an $O(m^2)$-approximation, in the same spirit as was done in [44].

To describe the MBL problem, let a partial order Π over a given set $\Sigma = [\![1\,;\,n]\!]$ of markers represent the incomplete genomic data at hand. A *linearization* of Π is a permutation (or a total order) $\pi = \langle \pi[1], \pi[2], \ldots, \pi[n] \rangle$ on Σ, such that, for all markers i, j, if $i <_\Pi j$, then $i <_\pi j$ (alternatively, i precedes j in π). In that case, π is said to be *compatible* with Π. An *adjacency* in the permutation π is a factor of length 2. The *breakpoint distance* $d_B(\pi_1, \pi_2)$ between two permutations π_1 and π_2 (over the same set Σ) is defined by the total number of adjacencies in π_1 which are not adjacencies in π_2. Finally, $\mathcal{I}_n$ denotes the identity permutation of size n.

The Minimum Breakpoint Linearization problem is defined as follows:

Problem	MBL
Input	A partial order Π
Output	A linearization π of Π
Minimize	$k = d_B(\pi, \mathcal{I}_n)$

Example 6.1. *Consider the partial order* Π *defined by* $1 <_\Pi 2 <_\Pi 3$*;* $4 <_\Pi 1 <_\Pi 5$ *and* $6 <_\Pi 3$ *(see Figure 6.1). This partial order yields four linearizations which are optimal solutions of* MBL *with* $d_B(\pi, \mathcal{I}_n) = 3$*. They are the following (common adjacencies with* $\mathcal{I}_n$ *are underlined):*

$$\langle 4,\ 1,\ \underline{5,\ 6},\ \underline{2,\ 3} \rangle$$

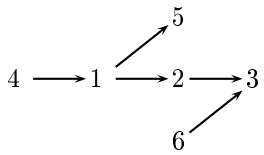


Figure 6.1: Directed Acyclic Graph representing the partial order Π of Example 6.1.

$$\langle 4,\ \underline{1,\ 2},\ \underline{5,\ 6},\ 3\rangle$$
$$\langle 6,\ 4,\ \underline{1,\ 2,\ 3},\ 5\rangle$$
$$\langle 4,\ 6,\ \underline{1,\ 2,\ 3},\ 5\rangle$$

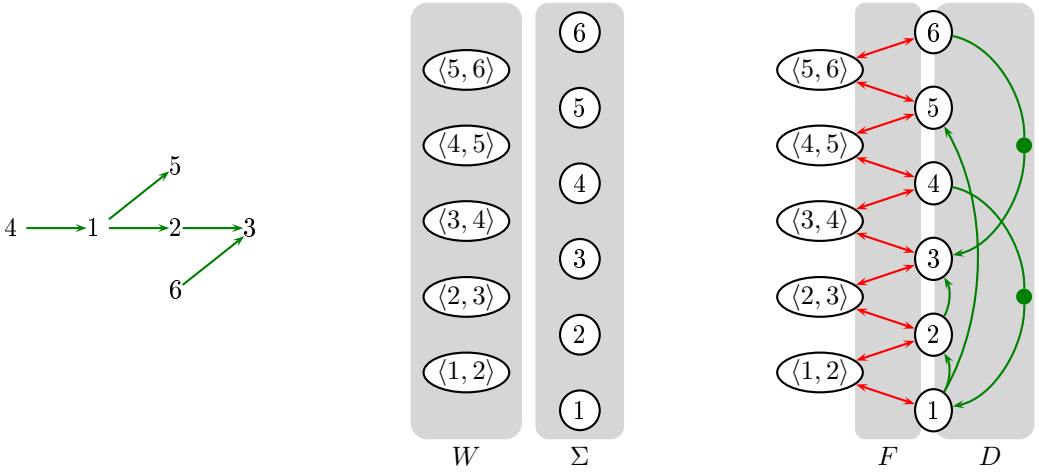
It is worth noting that the input partial order, in practice, is sometimes obtained by combining a limited number m of *genetic maps* [125, 127]. A genetic map consists of an ordered list of blocks $B_1, B_2, \ldots, B_q$, each of which is an unordered list of markers, i.e. any two markers from the same block are incomparable. The blocks $B_1, B_2, \ldots, B_q$ induce a partial order Π as follows: for any $a \in B_i$ and $b \in B_j$, $a <_\Pi b$ iff $i < j$. Note that it is not required for all maps to contain all markers, and we assume that combining two or more genetic maps never creates conflicts. The instance Π of Example 6.1 is induced by e.g. the following two maps.

$$\{4\} \longrightarrow \{1\} \longrightarrow \{2, 5\},$$
$$\{2, 6\} \longrightarrow \{3\}.$$

The Minimum Breakpoint Linearization problem, based on the genome rearrangement problem defined by Zheng and Sankoff [127], was studied independently in [23] and [73] (we note that in the latter, the problem is denoted as PBD, and deals with two partial orders instead of one partial order and one total order). In [23], Blin et al. prove that MBL is NP-hard and give two types of algorithms for solving MBL: (i) a heuristic and (ii) an exact, thus exponential-time, algorithm based on dynamic programming. Moreover, this last algorithm is efficient in the specific case where input genomes are created from a bounded number m of gene maps, in which the blocks have a bounded size. It is noted in [38] that the NP-hardness proof from [23] is actually an APX-hardness proof.

In [73], Fu and Jiang give an (independent) NP-hardness proof, and present the construction of the adjacency-order graph $\mathcal{G}_\Pi$ of a partial order Π in order to obtain heuristics and approximation algorithms [44]. However the main theorem in [73] is false (Example 6.1 is actually a counter-example [38]), which leads us to defining a new structure in order to solve MBL.

The chapter is organized as follows. Section 6.1 is devoted to the definition of a new adjacency-order graph, which is used in Section 6.2 to show that solving MBL may be reduced to solving a variant of the well-known Feedback Vertex Set problem, namely, Subset-FVS [67]. Using literature algorithms for Subset-FVS, Section 6.3 presents three algorithms, two approximations and one fixed-parameter tractable, for MBL. Finally, Section 6.4 presents an approximation algorithm which is specific to partial orders created from genomic maps, and has a ratio depending on the number m of those genomic maps.



(a) Original DAG Π (b) Vertices of G_Π (c) Arcs of G_Π

Figure 6.2: Construction of an adjacency-order graph. The symmetric arcs in F are represented as double arrows. The arcs in X are marked with a large dot.

6.1 Defining a New Adjacency-Order Graph G_Π

We present here the construction of a new structure to solve MBL whose main property will lead us to several different algorithms. This new adjacency-order graph G_Π contains both the features of the partial order Π and of the identity permutation $\mathcal{I}_n$. Some of the cycles in this graph will express the incompatibilities between the order Π and the permutation $\mathcal{I}_n$. In order to count or to bound the breakpoint distance between a linearization π of Π and $\mathcal{I}_n$, one has to identify the vertices in the adjacency-order graph needed to break all these conflict-cycles, and to count or bound their number. The MBL problem thus becomes a graph theory problem, which allows us either to use existing algorithms or to build up new graph-based algorithms.

Adjacency-order graph. Let $\Pi = (\Sigma, D)$ be a directed acyclic graph (DAG) representing a partial order over $\Sigma = [\![1\,;\,n]\!]$ (see Figure 6.2a for the instance given in Example 6.1), i.e. we write $i <_\Pi j$ iff there is a directed path from i to j in Π. We create a set W of vertices representing the adjacencies of the identity permutation $\mathcal{I}_n$ by $W = \{\langle i, i{+}1\rangle \mid 1 \leq i < n\}$. Finally, let $V = \Sigma \cup W$ (Figure 6.2b). Note that, in the following, we will not distinguish the vertices of Σ and their corresponding integers (this will always be clear from the context). Moreover, the natural order $<$ over the integers is also used as an order over Σ. We now construct a set of arcs F (denoted by an arrow $\to$) in the following way:

$$\begin{aligned} F = \quad & \{\langle i, i{+}1\rangle \to i \mid 1 \leq i < n\} \cup \{\langle i, i{+}1\rangle \to i+1 \mid 1 \leq i < n\} \\ \cup \quad & \{i \to \langle i, i{+}1\rangle \mid 1 \leq i < n\} \cup \{i+1 \to \langle i, i{+}1\rangle \mid 1 \leq i < n\} \end{aligned}$$

Each arc in F has one end in W and one end in Σ. We write $E = D \cup F$ (Figure 6.2c) and we define the *adjacency-order* graph G_Π of Π by $G_\Pi = (V, E)$.

In G_Π, the arcs of D that go top-down (see Figure 6.2c) intuitively show incompatibilities between the order in Π and the order in $\mathcal{I}_n$. We call such an arc *conflict-arc*, that is, an arc $i \to j \in D$ such that $i > j$, and we note $X[G_\Pi]$ (or only X, if there

is no ambiguity) the set containing all conflict-arcs: $X[G_\Pi] = \{i \to j \in D \mid i > j\}$. Now, every cycle containing a conflict-arc is called a *conflict-cycle*. In Theorem 6.3, we prove that the adjacencies involved in conflict-cycles are incompatible, so that we need to remove at least one adjacency from each of those cycles to obtain a linearization of Π. We also define a weight map $w[G_\Pi]$ on the vertices of G_Π, which associates 1 to each $u \in W$, and ∞ to $u \in \Sigma$.

Notations. An arc between u and v is written $u \to v$, or $u \to_{E'} v$ if it belongs to some subset E'. A *path* P is a (possibly empty) sequence of arcs written $u \xrightarrow{P}{}^* v$, or $u \xrightarrow{P}{}^*_{E'} v$ if P uses only arcs from E'. A non-empty path Q is written with a $+$ sign: $u \xrightarrow{Q}{}^+ v$. A *cycle* is a non-empty path $u \xrightarrow{\mathcal{C}}{}^+ v$ with $v = u$.

Given in G_Π a path $P = v_0 \to v_1 \to \ldots \to v_\ell$, we use the following notations: $\ell(P) = \ell$ is the length of P, $V(P) = \{v_h \mid 0 \leq h \leq \ell\}$, $W(P) = V(P) \cap W$, $\Sigma(P) = V(P) \cap \Sigma$, $E(P) = \{v_h \to v_{h+1} \mid 0 \leq h < \ell\}$, $F(P) = E(P) \cap F$, $D(P) = E(P) \cap D$, $X(P) = E(P) \cap X$. A cycle $\mathcal{C}$ is said to be *simple* if all vertices v_h are distinct (except $v_0 = v_\ell$), which implies $\ell(\mathcal{C}) = |V(\mathcal{C})| = |E(\mathcal{C})|$.

First Properties. The following lemma gives a way to exhibit conflict-arcs in some cycles of G_Π. It is used in Property 6.2 to give an insight on how conflict-cycles can appear in the adjacency-order graph.

Lemma 6.1. *Let $\mathcal{C}$ be a (not necessarily simple) cycle of G_Π. Let $c \in \Sigma$, such that there exists $a, b \in \Sigma(\mathcal{C})$ with $a \leq c < b$. Then at least one of the following propositions is true:*

(i) $\mathcal{C}$ contains an arc $u \to_X v$ with $v \leq c < u$

(ii) $\mathcal{C}$ contains both arcs $c+1 \to_F \langle c, c{+}1\rangle$ and $\langle c, c{+}1\rangle \to_F c$

Proof. Define $c^+ = \{d \mid d > c\} \cup \{\langle d, d{+}1\rangle \mid d > c\}$ and $c^- = \{d \mid d \leq c\} \cup \{\langle d, d{+}1\rangle \mid d < c\}$. Then $c^+ \cup \{\langle c, c{+}1\rangle\} \cup c^-$ is a partition of V. We show that when proposition (i) is false, proposition (ii) is necessarily true. Assume that proposition (i) is false. Since $\mathcal{C}$ contains vertices in both $c^+ \cup \{\langle c, c{+}1\rangle\}$ and c^- (resp. b and a), it thus contains an arc $u \to v$ with $u \in c^+ \cup \{\langle c, c{+}1\rangle\}$ and $v \in c^-$. We must have $u \to v \in F$, otherwise $u \to v \in D$ implies $u \to v \in X$ (since $u > v$), and proposition (i) would be true, a contradiction. Necessarily $u = \langle c, c{+}1\rangle$ and $v = c$ (there is no arc in F going out of c^+ into c^-). So $\mathcal{C}$ contains the arc $\langle c, c{+}1\rangle \to c$. Using the same argument, we can show that there is an arc $u' \to v'$ in $\mathcal{C}$ with $u' \in c^+$ and $v' \in \{\langle c, c{+}1\rangle\} \cup c^-$. Since $u' \to v'$ cannot be in X (since proposition (i) is false) nor in $D - X$ (these arcs go from c^- to c^+), then it is in F, and we can only have $u' = c + 1$ and $v' = \langle c, c{+}1\rangle$. So $\mathcal{C}$ also uses the arc $\langle c, c{+}1\rangle \to_F c$, and thus proposition (ii) is true.

□

Property 6.2. *Let $\mathcal{C}$ be a simple cycle with $|D(\mathcal{C})| \geq 2$. Then $\mathcal{C}$ is a conflict-cycle.*

Proof. Let $a = \min(\Sigma(\mathcal{C}))$, $b = \max(\Sigma(\mathcal{C}))$, and $u \to_D v$ and $u' \to_D v'$ two arcs of D appearing in $\mathcal{C}$. By contradiction, we suppose $\mathcal{C}$ is not a conflict-cycle (i.e., $\mathcal{C}$ contains no arc of X). For each $c \in \Sigma$ with $a \leq c < b$, we can use Lemma 6.1: only proposition (ii) can be true for each c, and there is an arc of F going out of $c + 1$ and another going into c appearing in cycle $\mathcal{C}$. Since this cycle is simple, we have

$u \neq c+1$ and $v \neq c$ for all $a \leq c < b$ (and similarly, $u' \neq c+1$ and $v' \neq c$). By definition of a and b, u, u', v, v' cannot be out of the interval $[\![a\,;\,b]\!]$, so $u = u' = a$ and $v = v' = b$. This implies that the simple cycle $\mathcal{C}$ uses twice the same arc $a \to_D b$, a contradiction.

□

6.2 Cutting All Conflict-Cycles in G_Π Is Enough

Now that we have defined how to construct G_Π starting from the input partial order Π, we turn to proving the main structural result of our paper: conflict-cycles contain all the conflicts between the partial order Π and the identity permutation $\mathcal{I}_n$ (see Theorem 6.3). More precisely, when appropriate adjacencies in $\mathcal{I}_n$ (identified as vertices in W) are removed, the remaining adjacency-order graph has no conflict-cycle and this condition is necessary and sufficient to obtain a linearization of Π that preserves all the remaining adjacencies in $\mathcal{I}_n$.

Theorem 6.3. *Let Π be a partial order, $G_\Pi = (V, E)$ its adjacency-order graph (with $V = \Sigma \cup W$ and $E = D \cup F$), and $W' \subseteq W$. Then there exists a permutation π over Σ, compatible with Π, and containing every adjacency from W' iff $G_\Pi[W' \cup \Sigma]$ has no conflict-cycle.*

Proof. ($\Rightarrow$) Let π be a linearization of Π containing every adjacency of W'. The following lemma will allow us to conclude by contradiction.

Lemma 6.4. *In this Lemma, we use the preconditions of Theorem 6.3 and the fact that π is a linearization of Π containing every adjacency of W'. Let $P = v_1 \to v_2 \to \ldots \to v_\ell$ be a path with vertices in $W' \cup \Sigma$ such that (H1) the vertices v_i, $1 \leq i \leq \ell$ are pairwise distinct, (H2) $\ell \geq 2$, (H3) $v_1, v_\ell \in \Sigma$, and (H4) for any $1 \leq i < \ell$, $v_i \to v_{i+1} \in F$.*
Let $a = \min(v_1, v_\ell)$ and $b = \max(v_1, v_\ell)$. Then the sequence $\langle a, a{+}1, a{+}2, \ldots, b\rangle$ is a factor of π. Moreover, $\Sigma(P) = [\![a\,;\,b]\!]$ and $W(P) = \{\langle c, c{+}1\rangle \mid a \leq c < b\}$.

Proof. Using H4, we can consider only the bipartite graph $(W' \cup \Sigma, F)$. With H3 we obtain that ℓ is odd, and $v_{2i-1} \in \Sigma$ (for any $1 \leq i \leq \frac{\ell+1}{2}$) whereas $v_{2i} \in W'$ (for any $1 \leq i \leq \frac{\ell-1}{2}$). Moreover, H2 implies that $\ell \geq 3$. The proof is by induction on ℓ:
For $\ell = 3$, since $v_1 \to_F v_2$ and $v_1 \in \Sigma$, there are two possible cases: (i) $v_2 = \langle v_1, v_1{+}1\rangle$ and (ii) $v_2 = \langle v_1{-}1, v_1\rangle$.

(i) $v_2 = \langle v_1, v_1{+}1\rangle$. Then $v_3 = v_1 + 1$ (since $v_2 \to_F v_3$ and $v_3 \neq v_1$, due to H1). In this case, $a = v_1$, $b = a + 1$ and $\langle a, b\rangle$ is a factor of π (indeed, it corresponds to the adjacency v_2 which belongs to W').

(ii) $v_2 = \langle v_1{-}1, v_1\rangle$. Likewise, $v_3 = v_1 - 1$, $a = v_3$, $b = a + 1$, and $\langle a, b\rangle$ is a factor of π.

Finally, in both cases we have $\Sigma(P) = \{v_1, v_3\} = [\![a\,;\,b]\!]$ and $W(P) = \{v_2\} = \{\langle a, a{+}1\rangle\}$: the lemma is true for $\ell = 3$.
For $\ell = \ell' + 2$, $\ell' \geq 3$, by induction, the lemma is true for the path $P' = v_1 \to \ldots \to v_{\ell'}$. Again, we need to consider two different cases (see Fig. 6.3a and 6.3b): (i) $v_1 < v_{\ell'}$ and (ii) $v_1 > v_{\ell'}$.

(i) In this case, $v_{\ell'-1} = \langle v_{\ell'}{-}1, v_{\ell'}\rangle$, and by H1, $v_{\ell-1} = v_{\ell'+1} = \langle v_{\ell'}, v_{\ell'}{+}1\rangle$, and $v_\ell = v_{\ell'+2} = v_{\ell'} + 1$. If we write $a' = v_1$ and $b' = v_{\ell'}$, then by induction the sequence

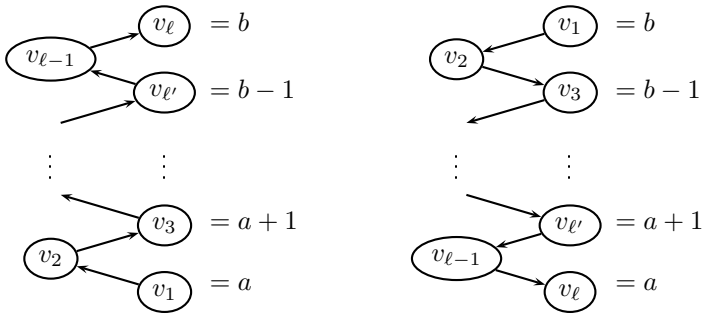


Figure 6.3: Two types of paths are possible in $(W' \cup \Sigma, F)$.

$\langle a', a'+1, \ldots, b' \rangle$ is a factor of π. And the sequence $\langle b', b'+1 \rangle$ is also a factor of π, since it corresponds to vertex $v_{\ell'+1} \in W'$. So the full sequence $\langle a', a'+1, \ldots, b', b'+1 \rangle$ is a factor of π. This proves the first part of the lemma, since $a = v_1 = a'$ and $b = v_\ell = b' + 1$. The second part is also true with $\Sigma(P) = \Sigma(P') \cup \{b\}$ and $W(P) = W(P') \cup \{\langle b-1, b \rangle\}$.

(ii) This case is symmetric to the previous one, with $a = a' - 1 = v_{\ell'} - 1$, $b = b' = v_1$ and $v_\ell = a$. We link together the sequences $\langle a'-1, a' \rangle$ and $\langle a', a'+1, \ldots, b' \rangle$ to prove that $\langle a', a'+1, \ldots, b' \rangle$ is a factor of π. Moreover, $\Sigma(P) = \Sigma(P') \cup \{a\}$ and $W(P) = W(P') \cup \{\langle a, a+1 \rangle\}$.

□

Proof of Theorem 6.3 (continued): We suppose, by contradiction, that there exists in $G_\Pi[W' \cup \Sigma]$ a cycle $\mathcal{C} = v_0 \to v_1 \to v_2 \to \ldots \to v_\ell = v_0$ containing an arc from X Wlog, assume that this arc is $v_0 \to_X v_1$, and that $\mathcal{C}$ is simple (otherwise, there exists a simple sub-cycle of $\mathcal{C}$ that contains an arc from X). We distinguish two cases, depending on whether $v_0 \to_X v_1$ is the only arc in $D(\mathcal{C})$.

First case: $v_0 \to v_1 \in X$ and for all i, $1 \leq i < \ell$, $v_i \to v_{i+1} \in F$ holds. In that case, we can directly use Lemma 6.4. Indeed, the path $v_1 \to v_2 \to \ldots \to v_\ell = v_0$ satisfies hypothesis H1 (by simplicity of $\mathcal{C}$), H2 (otherwise there would be a loop $v_0 \to v_0$ in X), H3 (since $v_\ell \to v_1 \in D$) and H4 (this is assumed in this first case). We also know, due to the fact that $v_0 \to_X v_1$, that $v_1 < v_\ell$. We can conclude that $\langle v_1, v_1+1, \ldots, v_\ell \rangle$ is a factor of π, so $v_1 <_\pi v_\ell = v_0$. This contradicts the fact that π is compatible with Π, since $v_0 <_\Pi v_1$.

Second case: Let $i_0 = 0, i_1, \ldots, i_{h-1}, i_h = \ell$ be the increasing sequence of indices such that $v_{i_j} \to v_{i_j+1} \in D$ for all j such that $0 \leq j < h$. Note that $h \geq 2$ and for all j, we have $v_{i_j} \in \Sigma$. Let us prove that for all $j < h$, the relation $v_{i_j} <_\pi v_{i_{j+1}}$ holds. The case where $i_{j+1} = i_j + 1$ is easy, since the arc $v_{i_j} \to_D v_{i_j+1}$ implies $v_{i_j} <_\Pi v_{i_{j+1}}$ (by construction of G_Π) and thus $v_{i_j} <_\pi v_{i_{j+1}}$, since π is compatible with Π. Now, assume there are several arcs between v_{i_j} and $v_{i_{j+1}}$, i.e. $i_{j+1} = i_j + m$ with $m \geq 2$. We use Lemma 6.4 with the path P in F given by $v_{i_j+1} \to v_{i_j+2} \to \ldots \to v_{i_j+m}$. Path P satisfies the hypotheses H1, H2, H3 and H4 of the lemma, thus one of the sequences $\langle v_{i_j+1}, v_{i_j+1}+1, \ldots, v_{i_j+m} \rangle$ and $\langle v_{i_j+m}, v_{i_j+m}+1, \ldots, v_{i_j+1} \rangle$ is a factor of π. Note that v_{i_j} is a distinct vertex from $v_{i_{j+1}}$ (since $h \geq 2$), and from other vertices in the set $\Sigma(P)$ as well (since each of them is the source of an arc from F in $\mathcal{C}$, whereas

v_{i_j} is the source of an arc from D in $\mathcal{C}$). Consequently, v_{i_j} cannot appear in either of the factors $\langle v_{i_j+1}, v_{i_j+1}{+}1, \ldots, v_{i_j+m}\rangle$ and $\langle v_{i_j+m}, v_{i_j+m}{+}1, \ldots, v_{i_j+1}\rangle$ of π. As v_{i_j} precedes v_{i_j+1} in Π (and thus in π), we have $v_{i_j} <_\pi v_{i'}$ for all $i' \in [\![i_j + 1\,;\, i_j + m]\!]$, and particularly, $v_{i_j} <_\pi v_{i_{j+1}}$.

In conclusion, we have $v_{i_j} <_\pi v_{i_{j+1}}$ for all $j < h$ and $v_{i_h} = v_{i_0}$, a contradiction since there can be no cycle in the relation $<_\pi$. Hence, the subgraph $G_\Pi[W' \cup \Sigma]$ does not contain any conflict-cycle.

($\Leftarrow$) *(constructive proof)* We use the following method to construct a linearization π of Π containing all adjacencies of W', where the subgraph $G' = G_\Pi[W' \cup \Sigma]$ is assumed to contain no conflict-cycle. We denote by $V_1, \ldots V_k$ the strongly connected components of G', ordered by topological order (i.e., if $u, v \in V_i$, there exists a path from u to v ; moreover, if $u \in V_i$ and $v \in V_j$ and there exists a path $u \to^* v$ in G', then $i \leq j$). We sort the elements of each set $V_i \cap \Sigma$ in ascending order of integers, and obtain a sequence μ_i. The concatenation $\langle \mu_1, \mu_2, \ldots\rangle$ gives π, a permutation of Σ. We now check that π contains every adjacency in W' and is compatible with Π.

Let $\langle a, a{+}1\rangle \in W'$. Vertices a and $a + 1$ are in the same strong connected component V_i, because of the arcs $a \leftrightarrow \langle a, a{+}1\rangle \leftrightarrow a + 1$. Those two elements are obviously consecutive in the corresponding μ_i, and appear as an adjacency in π. By contradiction, assume now that there exist two distinct elements $a, b \in \Sigma$ such that $a <_\pi b$ and $b <_\Pi a$. We denote by i and j the indices such that $a \in V_i$ and $b \in V_j$. Since $a <_\pi b$, we have $i \leq j$, and since $b <_\Pi a$, there exists a path $b \overset{P_1}{\to}{}^+_D a$ in (Σ, D). Therefore, in G', we have $i \geq j$. We thus deduce that $i = j$, and therefore a and b share the same strong connected component. This means that there also exists a path P_2 from a to b in G'. Hence, we have a cycle $b \overset{P_1}{\to}{}^+ a \overset{P_2}{\to}{}^+ b$, which cannot be a conflict-cycle, thus those paths do not use any arc from X. The latter is in particular true along P_1, which implies $b < a$, since each arc $u \to v$ in $D - X$ is such that $u < v$. On the other hand, a appears before b in π, and therefore in μ_i, so $a < b$, a contradiction. Finally π is a feasible solution for MBL(Π), with at least $|W'|$ common adjacencies with the identity permutation $\mathcal{I}_n$.

Since all vertices in $W - W'$ count for unconserved adjacencies (and thus define $d_B(\pi, \mathcal{I}_n)$), from Theorem 6.3 we directly get the following corollary.

Corollary 6.5. *The value k of an optimal solution of* MBL(Π) *is the minimum number of vertices one needs to delete in W to remove all conflict-cycles from G_Π.*

6.3 Algorithms Based on Subset-FVS

The structural property of the adjacency-order graph presented in the previous section (Theorem 6.3 and Corollary 6.5) entails a reduction of the problem MBL to a generalization of the well studied Feedback Vertex Set (FVS) problem, where only the conflict-cycles must be cut. This generalization, named Subset-FVS and studied by Even et al. [67], is defined as follows:

Problem	Subset-FVS
Input	A directed graph $G = (V, E)$, a set $Y \subseteq V \cup E$, a weight map $w : V \to \mathbb{R}$
Output	A set $V'' \subseteq V$ such that, with $V' = V - V''$, no cycle in $G[V']$ uses a vertex or an arc from Y
Minimize	The weight of V''

Algorithm 6.1 Reduction from MBL to AOG-SUBSET-FVS.

Input: A directed acyclic graph $\Pi = (\Sigma, D)$

1: Create $G_\Pi = (V, E)$ the adjacency-order graph of Π
2: $W'' \leftarrow$ AOG-SUBSET-FVS$(G_\Pi, X[G_\Pi], w[G_\Pi])$ /* *using e.g. [67, 49]* */
3: $W' \leftarrow W - W''$
4: $(V_1, V_2, \ldots, V_h) \leftarrow$ `SCC-sort`$(G_\Pi[W' \cup \Sigma])$
5: **for** $i \leftarrow 1$ **to** h **do**
6: $\quad \mu_i \leftarrow$ `sort`$(V_i \cap \Sigma)$
7: **end for**
8: $\pi \leftarrow \langle \mu_1, \mu_2, \ldots, \mu_h \rangle$
9: **return** π

In our approach, we are only interested in the restriction of SUBSET-FVS to adjacency-order graphs, where Y is the set of conflict arcs and w is such that only vertices in W can be deleted:

Problem	AOG-SUBSET-FVS
Input	An adjacency-order graph G_Π, $Y = X[G_\Pi]$, $w = w[G_\Pi]$
Output	A set W'' solution of SUBSET-FVS(G_Π, Y, w)
Minimize	The weight of W''

We note that any algorithm for SUBSET-FVS is also valid for AOG-SUBSET-FVS with the same approximation ratio and running time. Also, as the weights of all vertices are either 1 or ∞, it is possible to use an algorithm for the unweighted variant of SUBSET-FVS by replacing each vertex with infinite weight by $|V|+1$ vertices with the same incoming and outgoing arcs (the resulting graph has a quadratic number of vertices and the same feedback vertex sets).

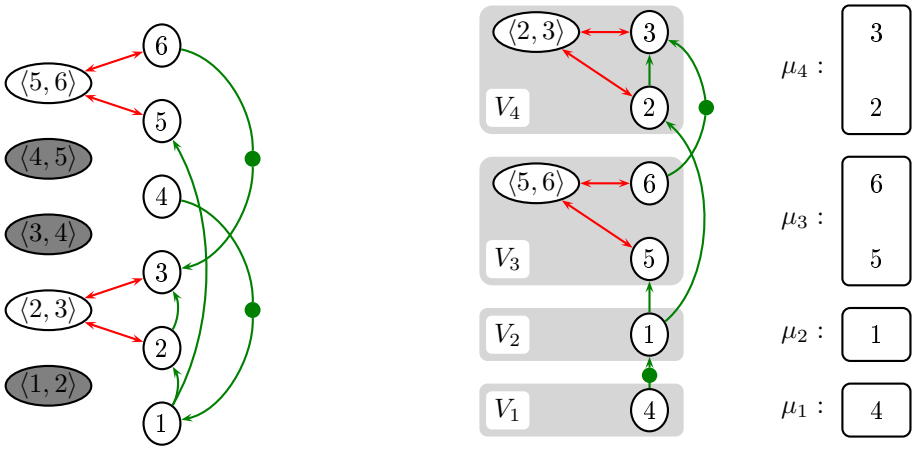
Two approximation algorithms are given in [67] for SUBSET-FVS. The first one achieves an approximation ratio of $O(\log^2 |Y|)$, while the second algorithm achieves a ratio of $O\left(\min(\log(\tau^*) \log\log(\tau^*), \log(n) \log\log(n))\right)$, where τ^* is the value of the optimal *fractional* solution for the corresponding linear programming problem (thus τ^* is upper bounded by the optimal solution of SUBSET-FVS). Also, Chitnis et al. [49] recently proposed an FPT algorithm for SUBSET-FVS, where the parameter is $|V'|$. More precisely, they designed a $2^{2^{O(|V'|)}} n^{O(1)}$ exact algorithm for the unweighted version of SUBSET-FVS.

Algorithm 6.1 is a reduction of MBL to AOG-SUBSET-FVS which allows us to use these algorithms (see Figure 6.4 for an example). We denote by `SCC-sort()` a function that decomposes a graph into its strongly connected components, and then topological sorts these components. Also, let `sort()` be a function that sorts a set of integers according to the increasing order of its elements. Algorithm 6.1 is derived from the constructive proof of Theorem 6.3, and its correctness follows from Theorem 6.3 itself.

Corollary 6.6. *Depending on the algorithm used for* AOG-SUBSET-FVS*, Algorithm 6.1 can be either*

- *an exponential-time exact algorithm (with running time $O(2^n)$ by brute-force),*
- *an FPT algorithm with running time $2^{2^{O(k)}} n^{O(1)}$,*
- *an $O\left(\log^2 |X|\right)$-approximation,*
- *an $O\left(\log(k) \log\log(k)\right)$-approximation,*

where $|X|$ is the number of conflict-arcs in Π and k the optimal value of our problem.



(a) Result of AOG-Subset-FVS: $W'' = \{\langle 1, 2\rangle, \langle 3, 4\rangle, \langle 4, 5\rangle\}$

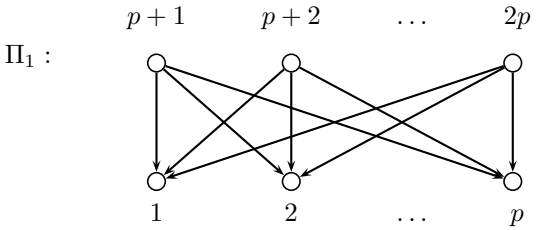
(b) `SCC-sort` yields four components V_1, V_2, V_3 and V_4, and Algorithm 6.1 returns permutation $\pi = \langle 4, 1, 5, 6, 2, 3\rangle$

Figure 6.4: Key steps of Algorithm 6.1 on the instance of MBL of Example 6.1.

Note that the two approximation ratios are incomparable, since we may have $|X| \approx nk$ or $k \approx n|X|$, as can be seen in Figures 6.5a and 6.5b.

6.4 An $(m^2 + 4m - 4)$-approximation Algorithm

In this section, we assume that the partial order Π is generated from m gene maps. Recall that a gene map is a totally ordered sequence of blocks, each of which is an unordered set of markers. We exploit this supplementary information to obtain an $(m^2 + 4m - 4)$-approximation algorithm for AOG-Subset-FVS, and therefore a new approximation algorithm, having the same ratio, for MBL. Before giving the algorithm and analyzing its approximation ratio, we first introduce a few definitions



(a) With Π_1: $|X| = p^2 >> k = 1$

$\Pi_2 : 1 \to 3 \to \ldots \to (2p-1) \to 2 \to 4 \to \ldots \to (2p)$

(b) With Π_2: $|X| = 1 << k = 2p - 1$

Figure 6.5: Comparing $|X|$ (number of conflict-arcs in Π) with k (the optimal breakpoint distance to the identity).

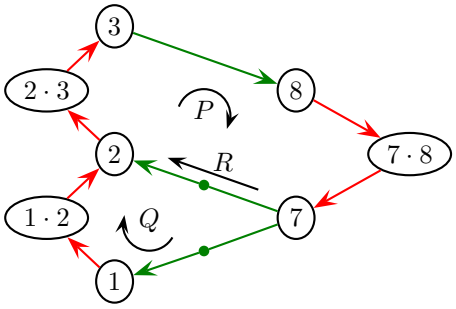


Figure 6.6: Cycle $\mathcal{C} = 2 \xrightarrow{P}{}^*7 \xrightarrow{Q}{}^*2$ is a conflict-cycle (it contains $7 \to 1 \in X$). The length-1 path R forms a shortcut for $\mathcal{C}$ (with $\mathcal{C}' = 2 \xrightarrow{P}{}^*7 \xrightarrow{R}{}^*2$, $\mathcal{C}'$ is a conflict-cycle, and $W(Q) = \{\langle 1, 2\rangle\}$). So $\mathcal{C}'$ is the only minimal conflict-cycle.

6.4.1 Definitions

Shortcut and Minimal Conflict-Cycle

A path $v \xrightarrow{R}{}^*_D u$ in (Σ, D) is said to be a *shortcut* of a conflict-cycle $\mathcal{C}$ (see Figure 6.6), if:

- $u, v \in \Sigma(\mathcal{C})$ (we write P and Q for the paths such that $\mathcal{C} = u \xrightarrow{P}{}^+ v \xrightarrow{Q}{}^+ u$),
- cycle $\mathcal{C}' = u \xrightarrow{P}{}^+ v \xrightarrow{R}{}^*_D u$ is a conflict-cycle,
- $W(Q) \neq \emptyset$ (using the shortcut removes at least one adjacency).

We say that a conflict-cycle is *minimal* if it has no shortcut. With the following property, we ensure that removing minimal conflict-cycles is enough to remove all conflict-cycles.

Property 6.7. *If an adjacency-order graph contains a conflict-cycle, it also contains a minimal conflict-cycle.*

Proof. Take a non-minimal conflict-cycle $\mathcal{C}$. If $\mathcal{C}$ is simple, we use the shortcut to create a conflict-cycle $\mathcal{C}'$ with $|W(\mathcal{C}')| < |W(\mathcal{C})|$. Otherwise, if $\mathcal{C}$ is not simple, there exists a vertex u such that $\mathcal{C} = u \xrightarrow{P}{}^+ u \xrightarrow{Q}{}^+ u$, and one of P and Q, say P, uses at least one arc of X. Thus P is a conflict-cycle, with $|W(P)| \leq |W(\mathcal{C})|$, and $|\ell(\mathcal{C}')| < |\ell(\mathcal{C})|$. In both cases, we create a conflict-cycle $\mathcal{C}'$ with either $|W(\mathcal{C}')| < |W(\mathcal{C})|$, or $|W(\mathcal{C}')| = |W(\mathcal{C})|$ and $|\ell(\mathcal{C}')| < |\ell(\mathcal{C})|$. If $\mathcal{C}'$ is not a minimal conflict-cycle, we can replace $\mathcal{C}$ by $\mathcal{C}'$ and iterate this process: it necessarily ends ($\ell(\mathcal{C})$ and $|W(\mathcal{C})|$ must remain positive integers), and reaches a minimal conflict-cycle. □

Joints and Indices

In a cycle $\mathcal{C}$, we call *joint* a vertex $e \in \Sigma$ whose incident arcs in $\mathcal{C}$ belong to $D(\mathcal{C})$ and to $F(\mathcal{C})$. Alternatively, there exist vertices e^D and e^F such that either

i. both arcs $e^D \to_D e$, $e \to_F e^F$ appear in $\mathcal{C}$, or

ii. both arcs $e^F \to_F e$, $e \to_D e^D$ appear in $\mathcal{C}$.

Consequently, two types of joints are identified: a joint is of type *(i.)* (resp. *(ii.)*) if it marks the end of a subpath with arcs from D (resp. from F) and the beginning of a subpath with arcs from F (resp. from D). In both cases, $e^F = \langle e, e{+}1\rangle$ or

$e^F = \langle e-1, e \rangle$. We say that e is a *low joint* if $e^F = \langle e, e+1 \rangle$ (intuitively, e is at the bottom of a subpath with arcs from F, independently of its type). Given a vertex $w \in W(\mathcal{C})$, we say that e is the *low joint associated to* w in $\mathcal{C}$, if the cycle $\mathcal{C}$ uses one of the paths $w \to_F^* e$ or $e \to_F^* w$ (e is either the first low joint after w in $\mathcal{C}$, or the last one before w).

Recall that Π is created from gene maps, indexed from 1 to m. From these gene maps, we can deduce two properties: (1) if there is an arc between u and v in Π, then u and v appear in consecutive blocks of the same gene map, and (2) if u and v appear in the same gene map, but in different blocks, then there exists a path $u \to^+ v$ or $v \to^+ u$ in Π.

For each $u \in \Sigma$, we denote $I(u) \subset [\![1\,;\, m]\!]$ the indices of the gene maps in which u appears (u appears in at least one gene map, so $I(u) \neq \emptyset$). For each arc $u \to_D v$ of D, we denote $\eta(u \to_D v)$ the index of a gene map in which u and v appear in consecutive blocks. Then $\eta(u \to_D v) \in I(u) \cap I(v)$. Given a cycle $\mathcal{C}$, we extend the notation η to each of its joints e: $\eta(e) = \eta(e^D \to e)$ if e is of type *(i)*; and $\eta(e) = \eta(e \to e^D)$ otherwise.

We also extend I to paths: given a path $P = u_0 \to u_1 \to \ldots \to u_\ell$, we write

$$I(P) = \bigcup_{\substack{0 \leq i \leq \ell \\ u_i \in \Sigma}} I(u_i)$$

Property 6.8. *Let $e \to f$ be an arc of D, and let $u \in \Sigma$ such that $\eta(e \to f) \in I(u)$. Then one of the paths $e \to^* u$ or $u \to^* f$ appears in the graph (Σ, D).*

Proof. The three markers u, e and f appear in the same gene map with index $\eta(e \to f)$. If u appears in a block strictly following e, then there exists a path $e \to^* u$ in Π, and thus in the graph (Σ, D). Else, since the block of e strictly precedes the block of f, u also appears strictly before f in this gene map, so there exists a path $u \to^* f$ in Π (and thus in the graph (Σ, D)).

□

6.4.2 Algorithm

Algorithm 6.2 is an (m^2+4m-4)-approximation for AOG-Subset-FVS. Used as a subroutine in Algorithm 6.1, it gives us an approximation of ratio (m^2+4m-4) for the MBL problem (see Corollary 6.16). The idea is that few conflict cycles can traverse the same vertex without using the same joints (here, "few" represents a function of the number of maps). It thus deletes all low joints of successive minimal conflict-cycles, until no such cycle remains. The correctness is a trivial consequence of Property 6.7.

The approximation ratio is thoroughly analyzed in the next section; this analysis can be briefly summarized as follows. The first step is to bound the number of low joints that can appear in some minimal conflict-cycle by $O(m)$, see Lemma 6.11 for a more precise bound. Then, we show that for any adjacency $w \in W$, at most $2m$ minimal conflict-cycles using w can appear during step 2 of Algorithm 6.2: by Lemma 6.13, there cannot be more than two low joints, associated to w in different cycles, appearing in the same gene map. Hence we can bound the number of vertices deleted by Algorithm 6.2 for a given vertex $w \in W$ by $O(m^2)$. This is the object of Theorem 6.15, which gives the precise value of the approximation ratio of Algorithm 6.2: $m^2 + 4m - 4$.

Algorithm 6.2 $(m^2 + 4m - 4)$-approximation for AOG-SUBSET-FVS.

Input: An adjacency-order graph $G_\Pi = (V, E), X[G_\Pi], w[G_\Pi]$

1: $W'' \leftarrow \emptyset$
2: **while** there exists a minimal conflict-cycle $\mathcal{C}$ in
3: $\quad L \leftarrow$ the set of low joints of $\mathcal{C}$
4: $\quad W'' \leftarrow W'' \cup \{e^F \ : \ e \in L\}$
5: **return** W''

6.4.3 Approximation Ratio Analysis

Bounding the number of joints in a minimal conflict-cycle

The lemmas below study the possible values of η for different low joints of one minimal conflict-cycle. They thus entail an upper bound on the number of such joints.

Lemma 6.9. *Let $\mathcal{C}$ be a minimal conflict-cycle where three vertices $u, e, f \in \Sigma(\mathcal{C})$ are such that (see Figure 6.7):*

- $\mathcal{C} = u \overset{P_1}{\rightarrow}{}^{+} e \rightarrow_D f \overset{P_2}{\rightarrow}{}^{+} u$
- *Each of the paths P_1 and P_2 uses at least one vertex from W and at least one arc from D.*

Then $\eta(e \rightarrow_D f) \notin I(u)$.

Proof. By contradiction, assume that $\eta(e \rightarrow_D f) \in I(u)$. Then, by Property 6.8, there exists a path R in D connecting either e to u or u to f. In the first case, we write $P = P_1$ and $Q = e \rightarrow_D f \overset{P_2}{\rightarrow}{}^{+} u$, and in the second, $P = P_2$ and $Q = u \overset{P_1}{\rightarrow}{}^{+} e \rightarrow_D f$, so that there exists a cycle $\mathcal{C}' = u \overset{P}{\rightarrow}{}^{+} e \overset{R}{\rightarrow}{}^{*} u$ (resp., $\mathcal{C}' = f \overset{P}{\rightarrow}{}^{+} u \overset{R}{\rightarrow}{}^{*} f$). Since $\mathcal{C}$ is a minimal conflict-cycle then R cannot be a shortcut, and with $W(Q)$ not being empty, cycle $\mathcal{C}'$ cannot be a conflict-cycle. Hence, no arc in $D(\mathcal{C}')$ appears in X.

Let $a = \min(\Sigma(\mathcal{C}'))$ and $b = \max(\Sigma(\mathcal{C}'))$. For all $c \in [\![a\,;\, b-1]\!]$, we can apply Lemma 6.1 on $\mathcal{C}'$, where only proposition (ii) can be true. Hence, $\mathcal{C}'$ contains every

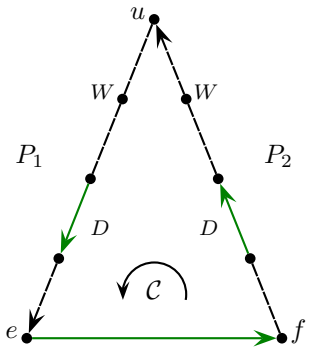


Figure 6.7: A cycle satisfying the conditions of Lemma 6.9 (black dotted lines represent paths).

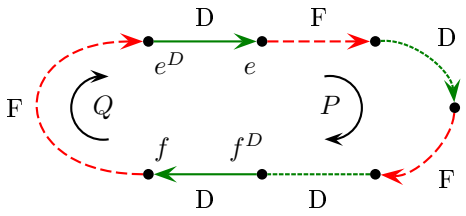


Figure 6.8: Illustration of Lemma 6.10, with $\lambda = 6$. Here $e_s = e^D$, $e_t = e$, $f_s = f^D$, $f_t = t$.

arc $c+1 \to_F \langle c, c+1\rangle$, $\langle c, c+1\rangle \to_F c$ between b and a. Moreover, none of these arcs can appear in R, so they all come from path P.

But P also contains at least one arc from D: let $a' \to_D b'$ be such an arc. By definition of a and b, we have $(a', b') \in [\![a\,;\,b]\!]$. The path P is part of a simple cycle, so it can enter and leave only once each vertex. This implies $a' \notin [\![a+1\,;\,b]\!]$ and $b' \notin [\![a\,;\,b-1]\!]$, so $a' = a$ and $b' = b$. Thus P contains every arc of the cycle $a \to_D b \to_F^* a$, which contradicts the fact that $\mathcal{C}$ is simple.

□

Lemma 6.10. *Let $\mathcal{C}$ be a minimal conflict-cycle, with $\lambda \geq 5$ joints. Let e and f be two non consecutive joints of $\mathcal{C}$. Then $\eta(e) \neq \eta(f)$.*

Proof. We write e_s, e_t, f_s, f_t for the four vertices of Σ such that $\{e_s, e_t\} = \{e, e^D\}$, $\{f_s, f_t\} = \{f, f^D\}$, and $\mathcal{C}$ uses both arcs $e_s \to e_t$ and $f_s \to f_t$ (see Figure 6.8). We write P and Q for the paths such that:

$$\mathcal{C} = e_s \to e_t \xrightarrow{P}{}^* f_s \to f_t \xrightarrow{Q}{}^* e_s$$

Since e and f are not consecutive joints, both paths P and Q use an arc from F (and a vertex from W). Moreover, since there are at least five joints, one of P and Q has a joint as an inner vertex. Wlog, let this path be P. Thus P uses an arc from D.

We denote by Q' the path $f_s \to f_t \xrightarrow{Q}{}^* e_s$, which also contains an arc from D. Hence the cycle $\mathcal{C} = f_s \xrightarrow{Q'}{}^* e_s \to e_t \xrightarrow{P}{}^* f_s$ satisfies the conditions of Lemma 6.9: $\eta(e) \notin I(f_s)$. This proves the lemma, since $\eta(f) \in I(f_s)$.

□

Lemma 6.11. *Let $\mathcal{C}$ be a minimal conflict-cycle with λ joints. Let P be a path in F between two consecutive joints in $\mathcal{C}$, and P' be the path P where both ends are removed. Then $\lambda \leq 2(m - |I(P')|) + 4$.*

Proof. If $\lambda \leq 4$, the lemma is proved, so we can assume there are at least five joints.

Wlog, we assume that e_2 and e_3 are the two consecutive joints linked by P. Let $u \in \Sigma(P')$. Vertex u appears between joints e_2 and e_3, so we can use Lemma 6.9 with u, e_i and e_i^D for all $i \notin [\![1\,;\,4]\!]$. Indeed, paths P_1 and P_2 contain respectively $u \to_F^+ e_3 \to_D^+ e_4$ and $e_1 \to_D^+ e_2 \to_F^+ u$, so they both use an arc from D, an arc from F, and a vertex from W. Hence, for all $i \in [\![5\,;\,\lambda]\!]$, we have $\forall u \in \Sigma(P')$, $\eta(e_i) \notin I(u)$, so $\eta(e_i) \notin I(P')$.

By Lemma 6.10, the cycle $\mathcal{C}$ cannot have more than two joints with the same value of η, since three joints cannot be pairwise consecutive when $\lambda \geq 5$. Thus we have $\lambda - 4 \leq 2|\,[\![1\,;\,m]\!] - I(P')| = 2(m - |I(P')|)$.

□

Bounding the number of cycles considered in Algorithm 6.2

We now aim at computing the number of cycles that Algorithm 6.2 can consider, in particular the number of cycles traversing a single vertex $w \in W$.

Lemma 6.12. *Let $w = \langle v, v{+}1\rangle \in W$, $\mathcal{C}_1$ and $\mathcal{C}_2$ be two cycles being considered during step 2 of Algorithm 6.2 ($\mathcal{C}_1$ being considered before $\mathcal{C}_2$), such that $w \in W(\mathcal{C}_1) \cap W(\mathcal{C}_2)$. Let a be the low joint associated to w in $\mathcal{C}_1$ and b be the one associated to w in $\mathcal{C}_2$. Then either*
(1) $\eta(a) \neq \eta(b)$, or
(2) $\eta(a) = \eta(b)$, a is of type ii*, and a^D and b appear in the same block of the gene map $\eta(a)$.*

Proof. Vertices a and b are low joints associated to $w = \langle v, v{+}1\rangle$, so $a \leq v$ and $b \leq v$. Vertex $a^F = \langle a, a{+}1\rangle$ is deleted when $\mathcal{C}_1$ is being considered so it cannot appear in $\mathcal{C}_2$. Thus $a < b$, and consequently b^F appears in the path $a \to_F^* w$ or $w \to_F^* a$. In $\mathcal{C}_2$, we write P for the path linking w and b in F, and Q_2 for the path linking b^D and w in $F \cup D$ (see Figure 6.9a). Depending on the type of b, we have

i $\mathcal{C}_2 = w \xrightarrow{Q_2}{}^* b^D \to_D b \xrightarrow{P}{}^* w$, or

ii $\mathcal{C}_2 = w \xrightarrow{P}{}^* b \to_D b^D \xrightarrow{Q_2}{}^* w$.

In $\mathcal{C}_1$, we write P' for the path linking b and a in F, and Q_1 for the path linking a^D and w in $F \cup D$. Thus, depending on the type of a, we have

i $\mathcal{C}_2 = w \xrightarrow{Q_1}{}^* a^D \to_D a \xrightarrow{P'}{}^* b \xrightarrow{P}{}^* w$, or

ii $\mathcal{C}_2 = w \xrightarrow{P}{}^* b \xrightarrow{P'}{}^* a \to_D a^D \xrightarrow{Q_2}{}^* w$.

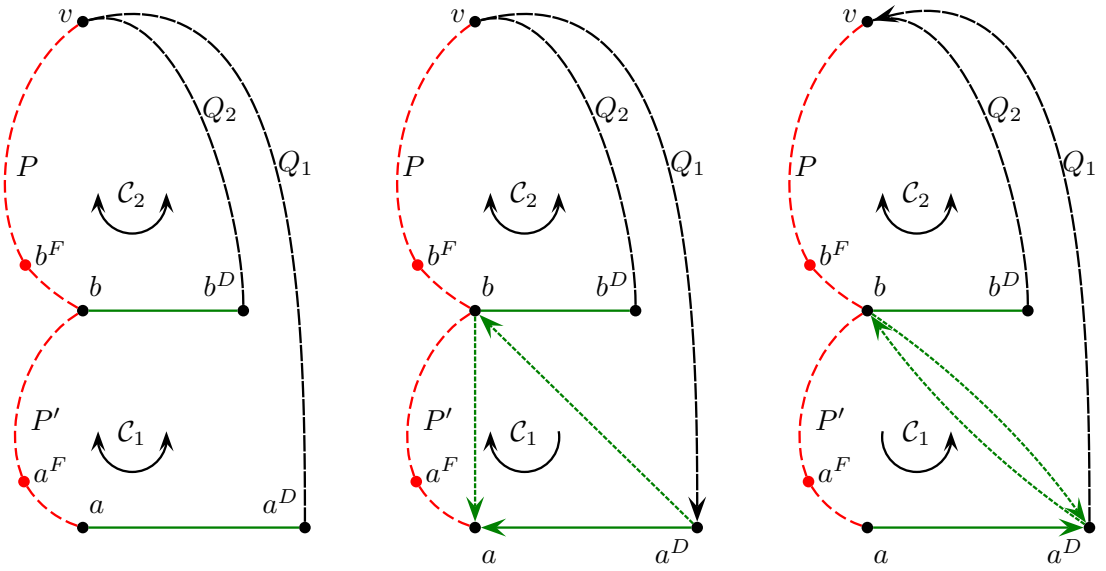
Note that P can be followed in different ways in $\mathcal{C}_1$ and $\mathcal{C}_2$. We do not distinguish this case in the notations, since paths in F can always be followed in both ways.

We suppose, by contradiction, that $\eta(a) = \eta(b)$.

First case: a is a joint of type (i). We use Property 6.8 with b and $a^D \to a$: there exists a path R in (Σ, D) from b to a or from a^D to b (see Figure 6.9b). If $R = b \to^* a$, we define $\mathcal{C}' = a \xrightarrow{P'}{}^* b \xrightarrow{R}{}^* a$, then $\mathcal{C}'$ is a conflict-cycle (this is due to Lemma 6.1: $\mathcal{C}'$ cannot contain $a^F \to a$ but visits the vertices a and $a+1$, hence it contains an arc from X). Moreover, $W(\mathcal{C}') \subsetneq W(\mathcal{C}_1)$, hence $\mathcal{C}_1$ cannot be a minimal conflict-cycle. Similarly, if $R = a^D \to^* b$, then $\mathcal{C}' = b \xrightarrow{P}{}^* w \xrightarrow{Q_1}{}^* a^D \xrightarrow{R}{}^* b$ is a conflict-cycle and $\mathcal{C}_1$ is not a minimal conflict-cycle. Thus this first case is a contradiction to the fact that $\eta(a) = \eta(b)$.

Second case: a is a joint of type (ii). We distinguish three sub-cases 2.1, 2.2 and 2.3 where, respectively, the path $b \xrightarrow{R}{}^* a^D$ exists in (Σ, D), the path $a^D \xrightarrow{R}{}^* b$ exists in (Σ, D), and a^D and b are incomparable in D (see Figure 6.9b).

2.1 $R = b \to^* a^D$. We consider $\mathcal{C}' = b \xrightarrow{R}{}^* a^D \xrightarrow{Q_1}{}^* w \xrightarrow{P}{}^* b$. If Q_1 contains an arc from X, $\mathcal{C}'$ is a conflict-cycle. Otherwise, the only marked arc in $\mathcal{C}_1$ is necessarily $a \to_X a^D$, so $a > a^D$. Hence the path R in D uses at least one marked arc (since $b > a > a^D$), so $\mathcal{C}'$ is again a conflict-cycle. In both cases, it contradicts the fact that $\mathcal{C}_1$ is a minimal conflict-cycle.



(a) Decomposition of $\mathcal{C}_1$ and $\mathcal{C}_2$ into arc and paths.

(b) First case (joint a is of type i)

(c) Second case (joint a is of type ii)

Figure 6.9: Path decomposition of $\mathcal{C}_1$ and $\mathcal{C}_2$ and case study for Lemma 6.12.

2.2 $R = a^D \rightarrow^* b$. We consider $\mathcal{C}' = b \xrightarrow{P'}{}^* a \rightarrow a^D \xrightarrow{R}{}^* b$. If $a^D < a$, then $a \rightarrow a^D$ is in X, so $\mathcal{C}'$ is a conflict-cycle. Otherwise, since $\mathcal{C}_1$ is simple, $a^D > b > a$, R contains an arc from X, and $\mathcal{C}'$ is also a conflict-cycle. Again, $\mathcal{C}_1$ cannot be a minimal conflict-cycle.

2.3 a^D and b are incomparable in (Σ, D). Since they appear in the same gene map with index $\eta(a) = \eta(b)$, they appear in the same block of this map.

□

Lemma 6.13. *Let $w = \langle v, v{+}1 \rangle \in W$, $\mathcal{C}_1$, $\mathcal{C}_2$ and $\mathcal{C}_3$ three cycles being considered during step 2 of Algorithm 6.2 (in this order), such that $w \in W(\mathcal{C}_1) \cap W(\mathcal{C}_2) \cap W(\mathcal{C}_3)$. Denote respectively by a, b and c the low joints associated to w in $\mathcal{C}_1$, $\mathcal{C}_2$ and $\mathcal{C}_3$. Then we cannot have $\eta(a) = \eta(b) = \eta(c)$.*

Proof. Assume that $\eta = \eta(a) = \eta(b) = \eta(c)$. Then we use Lemma 6.12 with $(\mathcal{C}_1, \mathcal{C}_2)$, $(\mathcal{C}_1, \mathcal{C}_3)$ and $(\mathcal{C}_2, \mathcal{C}_3)$ successively, thus (see Figure 6.10) :

- a^D and b appear in the same block of gene map η,
- a^D and c appear in the same block of gene map η,
- b^D and c appear in the same block of gene map η.

So, b and b^D both come from the same block of gene map η, which contradicts $\eta(b) = \eta$ (in the gene map $\eta(b)$, b and b^D appear in consecutive blocks).

□

Computing the Approximation Ratio

Lemma 6.14. *Let $w \in W$ and let $\mathbb{C}$ be the set of all cycles considered during step 2 of Algorithm 6.2 going through w. Then the cardinality of the set of low joints in cycles of $\mathbb{C}$ is upper bounded by $m^2 + 4m - 4$.*

Proof. We write $w = \langle v, v{+}1 \rangle \in W$, and $\mathbb{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_q\}$ for the set of the q cycles considered, in this order, by Algorithm 6.2. In each cycle $\mathcal{C}_h$, $1 \leq h \leq q$, w can

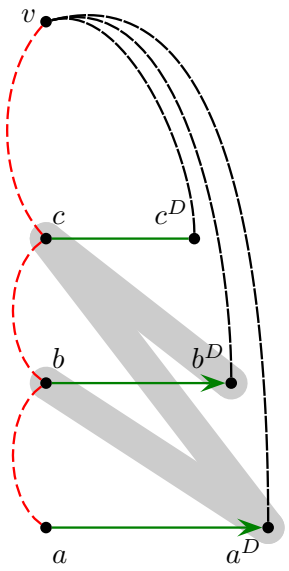


Figure 6.10: Illustration of the proof of Lemma 6.13. Vertices appearing in the same block of the gene map η are marked in gray.

be associated to a low joint v_h and to the corresponding deleted vertex $w_h = v_h^F = \langle v_h, v_h{+}1 \rangle$. We write P_h for the path $w_h \to_F^* w$ or $w \to_F^* w_h$ in $\mathcal{C}_h$, and λ_h for the number of joints of $\mathcal{C}_h$. Thus $\frac{\lambda_h}{2}$ is the number of low joints (and the number of deleted vertices) in this cycle.

Since w_h is deleted while $\mathcal{C}_h$ is being considered, for all $h' \in [\![h+1\,;\,q]\!]$, $w_h \notin W(\mathcal{C}_{h'})$, and $v_h < v_{h'} \leq v$. Indeed, for any $u \in [\![v_{h'}\,;\,v]\!]$, the vertex $\langle u, u{+}1 \rangle$ belongs to $W(\mathcal{C}_{h'})$, and $v_h, v_{h'} \leq v$. Finally, the path P_h uses each $v_{h'}$, $h' \in [\![h+1\,;\,q]\!]$.

Consider now the list $(\eta(v_{h+1}), \eta(v_{h+2}), \ldots, \eta(v_q))$. Using Lemma 6.13, the same value cannot appear more than twice in this list, so it contains at least $\lceil \frac{q-h}{2} \rceil$ different values. The first consequence is $q \leq 2m$ (with $h = 0$, all values are in a set of size m). And since for all $h' \in [\![h+1\,;\,q]\!]$, $\eta(v_{h'}) \in I(P_h)$, we have $|I(P_h)| \geq \lceil \frac{q-h}{2} \rceil$.

Using Lemma 6.11 for all h, we obtain $\lambda_h \leq 2(m - |I(P_h)|) + 4$. We can thus bound the number of low joints in $\mathcal{C}_h$:

$$\frac{\lambda_h}{2} \leq \frac{2(m - |I(P_h)|) + 4}{2} = m - |I(P_h)| + 2 \leq m - \left\lceil \frac{q-h}{2} \right\rceil + 2$$

By Lemma 6.10, we also have that, for $m \geq 2$, $\lambda_h \leq 2m$. Let L_w be the number of low joints in the cycles of $\mathbb{C}$:

$$\begin{aligned} L_w &= \sum_{h=1}^{q} \frac{\lambda_h}{2} \\ &\leq \sum_{h=1}^{q} \min\left(m, m - \left\lceil \frac{q-h}{2} \right\rceil + 2\right) \\ &\leq \sum_{d=0}^{q-1} \min\left(m, m - \left\lceil \frac{d}{2} \right\rceil + 2\right) \\ &\leq \left(\sum_{d=0}^{2m-1} m - \left\lceil \frac{d}{2} \right\rceil + 2\right) - 4 \end{aligned}$$

$$\begin{aligned} &\leq \left(\sum_{i=0}^{m-1}(m-i+2)+(m-i+1)\right)-4 \\ &\leq 2m^2-m(m-1)+3m-4 \\ &\leq m^2+4m-4 \end{aligned}$$

thus $L_w \leq m^2+4m-4$, which proves Lemma 6.14.

□

Theorem 6.15. *Algorithm 6.2 is an* (m^2+4m-4)*-approximation of* AOG-Subset-FVS*, where* m *is the number of gene maps used to create the input graph.*

Proof. Correctness of Algorithm 6.2 follows from Property 6.7: Algorithm 6.2 outputs a set W'' such that $G_\Pi[V-W'']$ does not contain any minimal conflict-cycle, hence it does not contain any conflict-cycle (i.e., W'' contains at least one vertex from each conflict-cycle). Let $W^o=\{w_1^o,\ldots,w_k^o\}$ be an optimal solution of size k. For each w_i^o, $i\in[\![1\,;\,k]\!]$, Algorithm 6.2 deletes at most m^2+4m-4 adjacencies of W (by Lemma 6.14). Since every cycle considered by the algorithm goes through some w_i^o, the total size of the output solution is at most $k(m^2+4m-4)$.

□

Corollary 6.16. *Using Algorithm 6.2 as an approximation for* AOG-Subset-FVS *in Algorithm 6.1 yields an* (m^2+4m-4)*-approximation for the* MBL *problem.*

Conclusion

In this chapter, we proposed a new graph structure to help solve MBL by representing the conflicts between the given partial order Π and the reference genome $\mathcal{I}_n$. Furthermore, we characterized the cycles containing these conflicts, we showed how the MBL problem reduces to solving the AOG-Subset-FVS problem in our structure, and we proposed three approximation and an FPT algorithm. This allows us to approach a given, practical instance of MBL from different viewpoints, by choosing the appropriate algorithm depending on the data at hand (i.e., whether the instance is created from few gene maps) and on the parameter evaluation (k and $|X|$).

It would be interesting to know whether there exists a constant-ratio approximation algorithm for MBL (which would classify MBL as APX-complete). Another challenge is to improve the running time of the FPT algorithm for MBL, either by using the specific structure of our adjacency-order graphs in [49], or by designing FPT algorithms based on new ideas, possibly with other parameters such as, for example, the number m of gene maps that were used to construct the partial order Π.

Conclusion and Perspectives

Following the plan of the manuscript, we now summarize the results presented in this work and draw the possible outlook we could follow for further research.

Sorting under the Permutation Model Is not Always Simple

As we have seen in Chapters 1 and 2, comparing genomic sequences even in the simplest model (where all genes are unique) is not always a trivial task. Although it is possible to compute many dissimilarity measures such as breakpoint and common interval distances, maximum and summed adjacency disruptions, etc., and even rearrangement distances with signed reversals or signed DCJs, we have proven that the problems Sorting by Transpositions and Sorting by Prefix Reversals are NP-hard. Yet, some clues make us assume that these problems should admit more precise approximation algorithms than the existing ones (with ratio 1.375 for SBT and 2 for SBPR). Indeed, the NP-hardness proofs we have provided do not yield a lower bound on the approximation ratio, and the "NP-hard sequences" they entail require a particularly large number of elements to simulate simple boolean formulas (a formula with n literals is transformed into permutations of size approximately $30n$ for SBT and $40n$ for SBPR).

An interesting question, regarding the Sorting by Transpositions problem, is whether this problem is still NP-hard when the size of the exchanged blocks is bounded. Notably the problem of sorting a permutation by *3-bounded* transpositions (transpositions involving at most three elements) has already received a particular attention [104, 92].

Another intriguing open problem is to determine the complexity class of the sorting by prefix reversals problem under the signed model (our approach based on the breakpoint distance cannot be used as such to prove that this problem is NP-hard). Indeed, having a sign constraint reduces the number of efficient operations, and should help find a good sorting scenario more easily, as is the case for the problems of sorting by reversals and by DCJ. However, the best polynomial algorithm is, to date, a 2-approximation (as in the unsigned case). Can this problem be solved with a better approximation, or even with an exact polynomial-time algorithm?

Finally, do there exist FPT algorithms with relevant parameters for these NP-hard problems? Both breakpoint and rearrangement distances are trivial parameters, but they are too high in practical cases to be of interest here.

Comparing Sequences with Duplicates Is Even More Challenging

Comparison problems become far more difficult when input data consists of strings instead of permutations. If some local string edition distances can still be computed (e.g., Hamming and Levenshtein distances), the problems considering large-scale evolutionary events (e.g., in a rearrangement distance) become immediately intractable. A possible approach consists in constructing a duplication scenario that allows to match different copies across genomes, in order to be able to model subsequent genomes with permutations.

In the exemplar model (see Chapter 3), we consider that only one copy of each gene directly originates from the common ancestor (and that other copies are recent duplications), and try to identify this "original" copy. However, whatever the optimization function we consider, computing an Exemplar Distance is APX-hard. The exemplar model entails a field of open problems, since no constant-ratio approximation or FPT algorithm is known for any of the many possible exemplar distances.

The matching model, where each copy of a gene in a sequence has to be matched to a unique copy in the other sequence, yields equally intractable problems. Yet, we have presented in Chapter 4 an FPT algorithm that computes the breakpoint distance under this model (better known as Minimum Common String Partition, MCSP). This algorithm is firstly of theoretical interest since it has only one parameter, the breakpoint distance (equivalently, the number of blocks in the optimal partition, k). On the other hand, with a complexity of $O(k^{21k^2}\,\mathrm{poly}(n))$, it is hardly relevant for practical implementation. Can a more efficient algorithm be obtained with interesting running times, both from theoretical and practical points of view? Also do there exist constant-ratio approximation algorithms for MCSP? Indeed, even though the problem does not deal with gene deletion and thus requires to have balanced strings in input, it is a central problem when one need to handle genomes with duplications.

Correcting and Completing Genomic Data

A major challenge for comparative genomics is to take into account the error-prone processes that generate the input of the problems considered. One representative example is genome assembling from high-speed sequencing (NGS) data: such data usually contains a high percentage of errors, balanced by an important redundancy of the reads. But errors, ambiguities and holes exist in virtually all genomic data. Ideally, all comparative genomics problems should take such a fact into account. However, due to the increase in the combinatorial complexity this represents, data sanitizing is a realm dominated by heuristics and is usually deferred as a pretreatment task.

Two such tasks are presented in Chapters 5 and 6. In Chapter 5, the Maximal Strip Recovery problem, and its variants, aim at extracting markers out of genomic sequences of close species that do not seem to belong to synteny blocks. We present a panel of algorithms and hardness results, which give a clear view of the computational complexity of the problem. In particular, authorizing duplicated genes yields – yet again – strongly intractable problems. On the other hand, with a model based on permutations, it is possible to create efficient FPT or approximation algorithms (see e.g. the $O^*(2.36^k\,\mathrm{poly}(n))$ FPT algorithm or the 2.5-approximation). They can get even more efficient when we bound the number of consecutive errors,

with an $O^*(2^k)$ FPT algorithm and a 1.8-approximation for the 1-gap variant.

In Chapter 6, we finally focus on the problem of infering a total gene order (a *linearization*) when only an incomplete precedence relation is known. Problems dealing with partially ordered genomes usually aim at removing conflicts within the input order, so that there *exists* a linearization, see e.g. *scaffold reconstruction*, and the exact algorithm Opera [74]. In the Minimum Breakpoint Linearization problem, we try to infer the *optimal* linearization from a conflict-free partial order. Optimality here is measured as the breakpoint distance to a reference genome. We propose a graph model for this problem, yielding a direct reduction to a known variant of the Feedback Vertex Set problem and thus to several approximations and to an FPT algorithm.

Further Perspectives

The most challenging question remains: can traditional comparative genomic methods apply to noisy or incomplete data? As a random example, can we define and compute a DCJ distance between partially ordered genomes? One possible outlook is via the Minimum Common String Partition problem. Indeed, we have presented an algorithm which, it seems, should allow for data with a bounded number of local errors, or even with uncertainties in the number of repetitions in the periodic strings. A large number of methods remain to be explored – or invented – in order to provide algorithms, which are both precise and efficient from a theoretical point of view, and which take into account all the facets of biological problems.

Extending the view out of solely comparative genomics, a variety of other aspects of bioinformatics need to be investigated. An example is graph-based problems: data such as metabolic networks or gene interactions are naturally modeled as graphs; and one aims at extracting remarkable subgraphs, discerning patterns, etc. Graph-based problems are a fertile ground for parameterized algorithms, since most methods like kernelization, color-coding, etc. naturally apply to this data representation. Hence, as a natural extension of this thesis, we propose to investigate such problems under the light of fixed-parameter tractability theory.

Bibliography

[1] S. B. Akers and B. Krishnamurthy. A group-theoretic model for symmetric interconnection networks. *IEEE Transactions on Computers*, 38(4):555–566, 1989. Cited pp. F.20 and 48.

[2] J. Akiyama and V. Chvátal. A short proof of the linear arboricity for cubic graphs. *The Bulletin of Liberal Arts & Sciences, Nippon Medical School*, 2:1–3, 1982. Cited p. 131.

[3] P. Alimonti and V. Kann. Hardness of approximating problems on cubic graphs. In G. C. Bongiovanni, D. P. Bovet, and G. D. Battista, editors, *CIAC*, volume 1203 of *LNCS*, pages 288–298. Springer, 1997. Cited pp. 124 and 134.

[4] P. Alimonti and V. Kann. Some APX-completeness results for cubic graphs. *Theoretical Computer Science*, 237(1-2):123–134, 2000. Cited p. 82.

[5] A. Amir, Y. Aumann, G. Benson, A. Levy, O. Lipsky, E. Porat, S. Skiena, and U. Vishne. Pattern matching with address errors: Rearrangement distances. *Journal of Computer and System Sciences*, 75(6):359–370, 2009. Cited pp. F.12 and 12.

[6] A. Amir, Y. Aumann, P. Indyk, A. Levy, and E. Porat. Efficient computations of ℓ_1 and ℓ_∞ rearrangement distances. In N. Ziviani and R. A. Baeza-Yates, editors, *SPIRE*, volume 4726 of *LNCS*, pages 39–49. Springer, 2007. Cited pp. F.12 and 12.

[7] S. Angibaud, G. Fertin, I. Rusu, A. Thévenin, and S. Vialette. On the Approximability of Comparing Genomes with Duplicates. *Journal of Graph Algorithms and Applications*, 13(1):19–53, 2009. Cited pp. F.9, F.24, 6, and 77.

[8] S. Angibaud, G. Fertin, I. Rusu, and S. Vialette. A General Framework for Computing Rearrangement Distances between Genomes with Duplicates. *Journal of Computational Biology*, 14(4):379–393, 2007. Cited pp. F.8 and 6.

[9] K. Appel and W. Haken. Every planar map is four colorable. *Bulletin of the American Mathematics Society*, pages 82–711, 1976. Cited p. 125.

[10] D. A. Bader, B. M. Moret, and M. Yan. A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. *Journal of Computational Biology*, 8(5):483–491, 2001. Cited pp. F.8 and 6.

[11] V. Bafna and P. Pevzner. Genome rearrangements and sorting by reversals. In *FOCS*, pages 148–157. IEEE, 1993. Cited pp. F.20, 48, and 72.

[12] V. Bafna and P. A. Pevzner. Sorting permutations by transpositions. In Clarkson [58], pages 614–623. Cited pp. F.12, 12, and 14.

[13] V. Bafna and P. A. Pevzner. Genome rearrangements and sorting by reversals. *SIAM Journal on Computing*, 25(2):272–289, 1996. Cited pp. F.7 and 5.

[14] V. Bafna and P. A. Pevzner. Sorting by transpositions. *SIAM Journal of Discrete Mathematics*, 11(2):224–240, 1998. Cited pp. F.8, F.12, 6, 12, and 21.

[15] R. Bar-Yehuda, M. Halldórsson, J. Naor, H. Shachnai, and I. Shapira. Scheduling split intervals. *SIAM Journal on Computing*, 36(1):1–15, 2006. Cited p. 124.

[16] M. Benoît-Gagné and S. Hamel. A new and faster method of sorting by transpositions. In B. Ma and K. Zhang, editors, *CPM*, volume 4580 of *LNCS*, pages 131–141. Springer, 2007. Cited pp. F.12 and 12.

[17] A. Bergeron, J. Mixtacki, and J. Stoye. A unifying view of genome rearrangements. In P. Bucher and B. M. E. Moret, editors, *WABI*, volume 4175 of *LNCS*, pages 163–173. Springer, 2006. Cited pp. F.8 and 6.

[18] A. Bergeron and J. Stoye. On the similarity of sets of permutations and its applications to genome comparison. In T. Warnow and B. Zhu, editors, *COCOON*, volume 2697 of *LNCS*, pages 68–79. Springer, 2003. Cited pp. F.7 and 5.

[19] P. Berman, S. Hannenhalli, and M. Karpinski. 1.375-approximation algorithm for sorting by reversals. In R. Möhring and R. Raman, editors, *ESA*, volume 2461 of *LNCS*, pages 200–210. Springer, 2002. Cited pp. F.20 and 48.

[20] P. Berman and M. Karpinski. On some tighter inapproximability results (extended abstract). In J. Wiedermann, P. van Emde Boas, and M. Nielsen, editors, *ICALP*, volume 1644 of *LNCS*, pages 200–209. Springer, 1999. Cited pp. F.20, F.25, 48, 77, 79, and 80.

[21] T. C. Biedl, G. Kant, and M. Kaufmann. On triangulating planar graphs under the four-connectivity constraint. *Algorithmica*, 19(4):427–446, 1997. Cited pp. 125 and 127.

[22] N. L. Biggs, E. K. Lloyd, and R. J. Wilson. *Graph theory, 1736-1936*. Oxford, Clarendon Press, 1976. Cited p. 125.

[23] G. Blin, E. Blais, D. Hermelin, P. Guillon, M. Blanchette, and N. El-Mabrouk. Gene maps linearization using genomic rearrangement distances. *Journal of Computational Biology*, 14(4):394–407, 2007. Cited pp. F.38, F.39, 8, 170, and 171.

[24] G. Blin, C. Chauve, G. Fertin, R. Rizzi, and S. Vialette. Comparing Genomes with Duplications: a Computational Complexity Point of View. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4(4):523–534, 2007. Cited pp. F.9, F.24, F.25, 6, and 77.

[25] G. Blin, G. Fertin, and C. Chauve. The breakpoint distance for signed sequences. In *1st Conference on Algorithms and Computational Methods for biochemical and Evolutionary Networks (CompBioNets'04)*, volume 3 of *Texts in Algorithms*, pages 3–16, Recife, Brazil, 2004. King's College London publications. Cited pp. F.8 and 6.

[26] G. Blin, G. Fertin, F. Sikora, and S. Vialette. The Exemplar Breakpoint Distance for non-trivial genomes cannot be approximated. In S. Das and R. Uehara, editors, *WALCOM*, volume 5431 of *LNCS*, pages 357–368. Springer, 2009. Cited pp. F.24 and 76.

[27] D. Bongartz. *Algorithmic Aspects of Some Combinatorial Problems in Bioinformatics*. PhD thesis, RWTH Aachen University, Germany, 2006. Cited pp. F.12 and 12.

[28] P. Bonizzoni, G. D. Vedova, R. Dondi, G. Fertin, R. Rizzi, and S. Vialette. Exemplar longest common subsequence. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4(4):535–543, 2007. Cited pp. F.24 and 77.

[29] L. Bulteau, G. Fertin, M. Jiang, and I. Rusu. Tractability and approximability of maximal strip recovery. In R. Giancarlo and G. Manzini, editors, *CPM*, volume 6661 of *LNCS*, pages 336–349. Springer, 2011. Cited pp. F.33, 117, and 120.

[30] L. Bulteau, G. Fertin, M. Jiang, and I. Rusu. Tractability and approximability of maximal strip recovery. *Theoretical Computer Science*, 440–441:14–28, 2012. Cited pp. F.33 and 117.

[31] L. Bulteau, G. Fertin, and I. Rusu. Maximal strip recovery problem with gaps: Hardness and approximation algorithms. In Y. Dong, D.-Z. Du, and O. H. Ibarra, editors, *ISAAC*, volume 5878 of *LNCS*, pages 710–719. Springer, 2009. Cited pp. F.33, 117, and 119.

[32] L. Bulteau, G. Fertin, and I. Rusu. Revisiting the minimum breakpoint linearization problem. In J. Kratochvíl, A. Li, J. Fiala, and P. Kolman, editors, *TAMC*, volume 6108 of *LNCS*, pages 163–174. Springer, 2010. Cited pp. F.37 and 169.

[33] L. Bulteau, G. Fertin, and I. Rusu. Sorting by transpositions is difficult. In L. Aceto, M. Henzinger, and J. Sgall, editors, *ICALP (Part 1)*, volume 6755 of *LNCS*, pages 654–665. Springer, 2011. Cited pp. F.11 and 11.

[34] L. Bulteau, G. Fertin, and I. Rusu. Pancake flipping is hard. In B. Rovan, V. Sassone, and P. Widmayer, editors, *MFCS*, volume 7464 of *LNCS*, pages 247–258. Springer, 2012. Cited pp. F.19 and 47.

[35] L. Bulteau, G. Fertin, and I. Rusu. Sorting by transpositions is difficult. *SIAM Journal of Discrete Mathematics*, 26(3):1148–1180, 2012. Cited pp. F.11 and 11.

[36] L. Bulteau, G. Fertin, and I. Rusu. Maximal strip recovery problem with gaps: Hardness and approximation algorithms. *Journal of Discrete Algorithms*, 19:1–22, 2013. Cited pp. F.33 and 117.

[37] L. Bulteau, G. Fertin, and I. Rusu. Pancake flipping is hard. *Journal of Computer and System Sciences*, 2013. Submitted. Cited pp. F.19 and 47.

[38] L. Bulteau, G. Fertin, and I. Rusu. Revisiting the minimum breakpoint linearization problem. *Theoretical Computer Science*, In Press, 2013. doi:10.1016/j.tcs.2012.12.026. Cited pp. F.37, F.39, 169, 170, and 171.

[39] L. Bulteau and M. Jiang. Inapproximability of (1, 2)-exemplar distance. In L. G. Bleris, I. I. Mandoiu, R. Schwartz, and J. Wang, editors, *ISBRA*, volume 7292 of *LNCS*, pages 13–23. Springer, 2012. Cited pp. F.23 and 75.

[40] L. Bulteau and M. Jiang. Inapproximability of (1,2)-exemplar distance. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, PrePrints, 2013. doi:10.1109/TCBB.2012.144. Cited pp. F.23, F.24, 75, and 77.

[41] L. Bulteau and C. Komusiewicz. Minimum common string partition parameterized by partition size is fixed-parameter tractable. In *Proceedings 21st European Symposium on Algorithms*, 2013. Submitted. Cited p. F.29.

[42] A. Caprara. Sorting by reversals is difficult. In *RECOMB*, pages 75–83, 1997. Cited pp. F.8 and 6.

[43] X. Chen. On sorting permutations by double-cut-and-joins. In M. T. Thai and S. Sahni, editors, *COCOON*, volume 6196 of *LNCS*, pages 439–448. Springer, 2010. Cited pp. F.8, F.26, 6, 78, and 79.

[44] X. Chen and Y. Cui. An approximation algorithm for the minimum breakpoint linearization problem. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 6(3):401–409, 2009. Cited pp. F.38, F.39, 170, and 171.

[45] X. Chen, J. Zheng, Z. Fu, P. Nan, Y. Zhong, S. Lonardi, and T. Jiang. Assignment of orthologous genes via genome rearrangement. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2(4):302–315, 2005. Cited pp. F.30, F.31, 86, and 113.

[46] Z. Chen, R. H. Fowler, B. Fu, and B. Zhu. On the inapproximability of the exemplar conserved interval distance problem of genomes. *Journal of Combinatorial Optimization*, 15(2):201–221, 2008. Cited pp. F.24 and 77.

[47] Z. Chen, B. Fu, M. Jiang, and B. Zhu. On recovering syntenic blocks from comparative maps. *Journal of Combinatorial Optimization*, 18(3):307–318, 2009. Cited pp. F.34, F.36, 118, 119, 120, 121, and 139.

[48] Z. Chen, B. Fu, and B. Zhu. The approximability of the exemplar breakpoint distance problem. In S.-W. Cheng and C. K. Poon, editors, *AAIM*, volume 4041 of *LNCS*, pages 291–302. Springer, 2006. Erratum in FAW-AAIM 2012:368. Cited pp. F.24 and 77.

[49] R. H. Chitnis, M. Cygan, M. T. Hajiaghayi, and D. Marx. Directed subset feedback vertex set is fixed-parameter tractable. In A. Czumaj, K. Mehlhorn, A. M. Pitts, and R. Wattenhofer, editors, *ICALP (Part 1)*, volume 7391 of *LNCS*, pages 230–241. Springer, 2012. Cited pp. F.40, 170, 177, and 186.

[50] B. Chitturi, W. Fahle, Z. Meng, L. Morales, C. Shields, I. Sudborough, and W. Voit. An $(18/11)n$ upper bound for sorting by prefix reversals. *Theoretical Computer Science*, 410(36):3372–3390, 2009. Cited pp. F.20 and 48.

[51] B. Chitturi and I. H. Sudborough. Bounding prefix transposition distance for strings and permutations. In *HICSS*, page 468. IEEE, 2008. Cited pp. F.12 and 12.

[52] M. Chlebík and J. Chlebíková. Complexity of approximating bounded variants of optimization problems. *Theoretical Computer Science*, 354(3):320 – 338, 2006. Cited pp. 124 and 139.

[53] V. Choi, C. Zheng, Q. Zhu, and D. Sankoff. Algorithms for the extraction of synteny blocks from comparative maps. In R. Giancarlo and S. Hannenhalli, editors, *WABI*, volume 4645 of *LNCS*, pages 277–288. Springer, 2007. Cited p. 120.

[54] D. A. Christie. Sorting permutations by block-interchanges. *Information Processing Letters*, 60(4):165–169, 1996. Cited pp. F.12 and 12.

[55] D. A. Christie. *Genome Rearrangement Problems*. PhD thesis, University of Glasgow, Scotland, 1998. Cited pp. F.12, 12, and 45.

[56] D. A. Christie and R. W. Irving. Sorting strings by reversals and by transpositions. *SIAM Journal of Discrete Mathematics*, 14(2):193–206, 2001. Cited pp. F.12 and 12.

[57] J. Cibulka. On average and highest number of flips in pancake sorting. *Theoretical Computer Science*, 412(8-10):822–834, 2011. Cited pp. F.20 and 48.

[58] K. L. Clarkson, editor. *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, 22-24 January 1995. San Francisco, California.* ACM/SIAM, 1995. Cited pp. 191 and 196.

[59] D. Cohen and M. Blum. On the problem of sorting burnt pancakes. *Discrete Applied Mathematics*, 61(2):105–120, 1995. Cited pp. F.20 and 48.

[60] G. Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. In *SODA*, pages 667–676, 2002. Cited pp. F.12 and 12.

[61] P. Damaschke. Minimum common string partition parameterized. In K. A. Crandall and J. Lagergren, editors, *WABI*, volume 5251 of *LNCS*, pages 87–98. Springer, 2008. Cited pp. F.9, F.30, 7, 86, and 90.

[62] Z. Dias and J. Meidanis. Sorting by prefix transpositions. In A. H. F. Laender and A. L. Oliveira, editors, *SPIRE*, volume 2476 of *LNCS*, pages 65–76. Springer, 2002. Cited pp. F.12 and 12.

[63] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999. Cited p. 5.

[64] H. Dweighter [pseudonym of J. E. Goodman]. *American Mathematics Monthly*, 82(1), 1975. Cited pp. F.8, F.19, 6, and 48.

[65] I. Elias and T. Hartman. A 1.375-approximation algorithm for sorting by transpositions. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(4):369–379, 2006. Cited pp. F.12 and 12.

[66] H. Eriksson, K. Eriksson, J. Karlander, L. J. Svensson, and J. Wästlund. Sorting a bridge hand. *Discrete Mathematics*, 241(1-3):289–300, 2001. Cited pp. F.12 and 12.

[67] G. Even, J. Naor, B. Schieber, and M. Sudan. Approximating minimum feedback sets and multi-cuts in directed graphs. In E. Balas and J. Clausen, editors, *IPCO*, volume 920 of *LNCS*, pages 14–28. Springer, 1995. Cited pp. F.40, 8, 170, 171, 176, and 177.

[68] J. Feng and D. Zhu. Faster algorithms for sorting by transpositions and sorting by block interchanges. *ACM Transactions on Algorithms*, 3(3), 2007. Cited pp. F.12 and 12.

[69] G. Fertin, A. Labarre, I. Rusu, E. Tannier, and S. Vialette. *Combinatorics of Genome Rearrangements*. Computational Molecular Biology. MIT Press, 2009. Cited pp. F.12, F.31, 4, 12, and 113.

[70] J. Fischer and S. Ginzinger. A 2-approximation algorithm for sorting by prefix reversals. In G. S. Brodal and S. Leonardi, editors, *ESA*, volume 3669 of *LNCS*, pages 415–425. Springer, 2005. Cited pp. F.20 and 48.

[71] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006. Cited p. 5.

[72] B. Fu, H. Jiang, B. Yang, and B. Zhu. Exponential and polynomial time algorithms for the minimum common string partition problem. In W. Wang,

X. Zhu, and D.-Z. Du, editors, *COCOA*, volume 6831 of *LNCS*, pages 299–310. Springer, 2011. Cited pp. F.30 and 86.

[73] Z. Fu and T. Jiang. Computing the breakpoint distance between partially ordered genomes. *Journal of Bioinformatics and Computational Biology*, 5(5):1087–1101, 2007. Cited pp. F.38, F.39, 8, 170, and 171.

[74] S. Gao, W.-K. Sung, and N. Nagarajan. Opera: Reconstructing optimal genomic scaffolds with high-throughput paired-end sequences. *Journal of Computational Biology*, 18(11):1681–1691, 2011. Cited p. 189.

[75] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979. Cited pp. F.7 and 5.

[76] W. Gates and C. Papadimitriou. Bounds for sorting by prefix reversal. *Discrete Mathematics*, 27(1):47–57, 1979. Cited pp. F.20 and 48.

[77] A. Goldstein, P. Kolman, and J. Zheng. Minimum common string partition problem: Hardness and approximations. *The Electronic Journal of Combinatorics*, 12, 2005. Cited pp. F.30 and 86.

[78] M. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York, 1980. Cited p. 124.

[79] Q.-P. Gu, S. Peng, and Q. M. Chen. Sorting permutations and its applications in genome analysis. *Lectures on Mathematics in the Life Science*, 26:191–201, 1999. Cited pp. F.12 and 12.

[80] S. A. Guyer, L. S. Heath, and J. P. Vergara. Subsequence and run heuristics for sorting by transpositions. Technical report, Virginia State University, 1997. Cited pp. F.12 and 12.

[81] M. M. Halldórsson. Approximating discrete collections via local improvements. In Clarkson [58], pages 160–169. Cited pp. 125 and 143.

[82] R. W. Hamming. Error detecting and error correcting codes. *Bell System technical journal*, 29(2):147–160, 1950. Cited pp. F.7 and 5.

[83] S. Hannenhalli and P. Pevzner. Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. In *STOC*, pages 178–189. ACM, 1995. Cited p. F.20.

[84] S. Hannenhalli and P. A. Pevzner. Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. *Journal of the ACM*, 46(1):1–27, 1999. Cited pp. F.25, F.28, 48, 77, and 78.

[85] T. Hartman and R. Shamir. A simpler and faster 1.5-approximation algorithm for sorting by transpositions. *Information and Computation*, 204(2):275–290, 2006. Cited pp. F.12 and 12.

[86] S. Heber and J. Stoye. Finding all common intervals of k permutations. In A. Amir and G. M. Landau, editors, *CPM*, volume 2089 of *LNCS*, pages 207–218. Springer, 2001. Cited pp. F.7 and 5.

[87] M. Heydari and I. Sudborough. On the diameter of the pancake network. *Journal of Algorithms*, 25(1):67–94, 1997. Cited pp. F.20 and 48.

[88] M. H. Heydari and I. H. Sudborough. On sorting by prefix reversals and the diameter of pancake networks. In F. Meyer auf der Heide, B. Monien, and A. L. Rosenberg, editors, *Heinz Nixdorf Symposium*, volume 678 of *LNCS*, pages 218–227. Springer, 1992. Cited pp. F.20 and 48.

[89] H. Jiang, Z. Li, G. Lin, L. Wang, and B. Zhu. Exact and approximation algorithms for the complementary maximal strip recovery problem. *Journal of Combinatorial Optimization*, 23(4):493–506, 2012. Cited pp. 120, 156, and 157.

[90] H. Jiang and B. Zhu. A linear kernel for the complementary maximal strip recovery problem. In J. Kärkkäinen and J. Stoye, editors, *CPM*, volume 7354 of *Lecture Notes in Computer Science*, pages 349–359. Springer, 2012. Cited pp. F.36, 120, 121, and 157.

[91] H. Jiang, B. Zhu, D. Zhu, and H. Zhu. Minimum common string partition revisited. *Journal of Combinatorial Optimization*, 23:519–527, 2012. Cited pp. F.9, F.30, 7, and 86.

[92] H. Jiang, D. Zhu, and B. Zhu. A $(1+\epsilon)$-approximation algorithm for sorting by short block-moves. *Theoretical Computer Science*, 439:1–8, 2012. Cited pp. 12 and 187.

[93] M. Jiang. Inapproximability of maximal strip recovery. In Y. Dong, D.-Z. Du, and O. H. Ibarra, editors, *ISAAC*, volume 5878 of *LNCS*, pages 616–625. Springer, 2009. Cited p. 126.

[94] M. Jiang. On the parameterized complexity of some optimization problems related to multiple-interval graphs. *Theoretical Computer Science*, 411(49):4253–4262, 2010. Cited pp. F.36, 120, and 121.

[95] M. Jiang. Inapproximability of maximal strip recovery. *Theoretical Computer Science*, 412(29):3759–3774, 2011. Cited pp. F.34, F.36, 119, 120, 121, 131, 139, 140, and 154.

[96] M. Jiang. The zero exemplar distance problem. *Journal of Computational Biology*, 18(9):1077–1086, 2011. Cited pp. F.24 and 76.

[97] J. Kececioglu and D. Sankoff. Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement. *Algorithmica*, 13(1-2):180–210, 1995. Cited pp. F.7 and 5.

[98] A. Labarre. New bounds and tractable instances for the transposition distance. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(4):380–394, 2006. Cited pp. F.12 and 12.

[99] A. Labarre. Edit distances and factorisations of even permutations. In D. Halperin and K. Mehlhorn, editors, *ESA*, volume 5193 of *LNCS*, pages 635–646. Springer, 2008. Cited pp. F.12 and 12.

[100] A. Labarre and J. Cibulka. Polynomial-time sortable stacks of burnt pancakes. *Theoretical Computer Science*, 412(8-10):695–702, 2011. Cited pp. F.20, 48, and 72.

[101] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. In *Soviet physics doklady*, volume 10, page 707, 1966. Cited pp. F.7 and 5.

[102] G. Lin, R. Goebel, Z. Li, and L. Wang. An improved approximation algorithm for the complementary maximal strip recovery problem. *Journal of Computer and System Sciences*, 78(3):720 – 730, 2012. Cited pp. F.36, 120, and 121.

[103] Y. C. Lin, C. L. Lu, H.-Y. Chang, and C. Y. Tang. An efficient algorithm for sorting by block-interchanges and its application to the evolution of vibrio species. *Journal of Computational Biology*, pages 102–112, 2005. Cited pp. F.8 and 6.

[104] M. Mahajan, R. Rama, and S. Vijayakumar. On sorting by 3-bounded transpositions. *Discrete Mathematics*, 306(14):1569–1585, 2006. Cited pp. 12 and 187.

[105] G. J. Minty. On maximal independent sets of vertices in claw-free graphs. *Journal of Combinatorial Theory, Series B*, 28(3):284–304, 1980. Cited p. 125.

[106] J. Misra and D. Gries. A constructive proof of Vizing's theorem. *Information Processing Letters*, 41(3):131–133, 1992. Cited p. 134.

[107] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Number 31 in Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, 2006. Cited pp. F.7 and 5.

[108] C. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43(3):425–440, 1991. Cited pp. F.7, 5, and 8.

[109] G. Petsko. Rising in the east. *Genome Biology*, 11(1):102, 2010. Cited pp. F.5 and 3.

[110] P. A. Pevzner and M. S. Waterman. Open combinatorial problems in computational molecular biology. In *ISTCS*, pages 158–173, 1995. Cited pp. F.11 and 12.

[111] X.-Q. Qi. *Combinatorial Algorithms of Genome Rearrangements in Bioinformatics*. PhD thesis, University of Shandong, China, 2006. Cited pp. F.12 and 12.

[112] K. Qiu, H. Meijer, and S. Akl. Parallel routing and sorting on the pancake network. In F. Dehne, F. Fiala, and W. W. Koczkodaj, editors, *ICCI*, volume 497 of *LNCS*, pages 360–371. Springer Berlin Heidelberg, 1991. Cited pp. F.20 and 48.

[113] A. J. Radcliffe, A. D. Scott, and A. L. Wilmer. Reversals and transpositions over finite alphabets. *SIAM Journal of Discrete Mathematics*, 19:224–244, 2005. Cited pp. F.12 and 12.

[114] N. Robertson, D. P. Sanders, P. D. Seymour, and R. Thomas. Efficiently four-coloring planar graphs. In G. L. Miller, editor, *STOC*, pages 571–575. ACM, 1996. Cited p. 125.

[115] D. Sankoff. Genome rearrangement with gene families. *Bioinformatics*, 15(11):909–917, 1999. Cited pp. F.8, F.23, 6, and 76.

[116] D. Sankoff and L. Haque. Power boosts for cluster tests. In A. McLysaght and D. H. Huson, editors, *Comparative Genomics*, volume 3678 of *LNCS*, pages 121–130. Springer, 2005. Cited pp. F.24 and 77.

[117] D. Sankoff, C. Zheng, A. Muñoz, Z. Yang, Z. Adam, R. Warren, V. Choi, and Q. Zhu. Issues in the reconstruction of gene order evolution. *Journal of Computer Science and Technology*, 25(1):10–25, 2009. Cited p. 7.

[118] D. Shapira and J. A. Storer. Edit distance with move operations. In A. Apostolico and M. Takeda, editors, *CPM*, volume 2373 of *LNCS*, pages 85–98. Springer, 2002. Cited pp. F.12 and 12.

[119] D. Shapira and J. A. Storer. Edit distance with move operations. *Journal of Discrete Algorithms*, 5(2):380–392, 2007. Cited pp. F.30 and 86.

[120] K. M. Swenson, M. Marron, J. V. Earnest-DeYoung, and B. M. E. Moret. Approximating the true evolutionary distance between two genomes. *ACM*

Journal of Experimental Algorithmics, 12, 2008. Cited pp. F.30, F.31, 86, and 113.

[121] T. Uno and M. Yagiura. Fast algorithms to enumerate all common intervals of two permutations. *Algorithmica*, 26:2000, 2000. Cited pp. F.7 and 5.

[122] L. Wang and B. Zhu. On the tractability of maximal strip recovery. In J. Chen and S. B. Cooper, editors, *TAMC*, volume 5532 of *LNCS*, pages 400–409. Springer, 2009. Cited pp. F.34 and 119.

[123] L. Wang and B. Zhu. On the tractability of maximal strip recovery. *Journal of Computational Biology*, 17(7):907–914, 2010. Erratum in *Journal of Computational Biology*, 18:129, 2011. Cited pp. F.36, 120, and 121.

[124] S. Yancopoulos, O. Attie, and R. Friedberg. Efficient sorting of genomic permutations by translocation, inversion and block interchange. *Bioinformatics*, 21(16):3340–3346, 2005. Cited pp. F.8, F.25, F.26, F.28, 6, 77, and 78.

[125] I. Yap, D. Schneider, J. Kleinberg, D. Matthews, S. Cartinhourb, and S. McCouch. A graph-theoretic approach to comparing and integrating genetic, physical and sequence-based maps. *Genetics*, 165(4):2235–2247, 2003. Cited pp. F.38 and 171.

[126] C. Zheng, A. Lenert, and D. Sankoff. Reversal distance for partially ordered genomes. In *ISMB (Supplement of Bioinformatics)*, pages 502–508, 2005. Cited pp. F.38 and 170.

[127] C. Zheng and D. Sankoff. Genome rearrangements with partially ordered chromosomes. In L. Wang, editor, *COCOON*, volume 3595 of *LNCS*, pages 52–62. Springer, 2005. Cited pp. F.38, F.39, 8, 170, and 171.

[128] C. Zheng, Q. Zhu, and D. Sankoff. Removing noise and ambiguities from comparative maps in rearrangement analysis. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4(4):515–522, 2007. Cited pp. F.34, 7, 118, 119, 120, and 161.

[129] B. Zhu. Efficient exact and approximate algorithms for the complement of maximal strip recovery. In B. Chen, editor, *AAIM*, volume 6124 of *LNCS*, pages 325–333. Springer, 2010. Cited p. 120.

l'université nantes angers le mans

PÔLE DE RECHERCHE ET D'ENSEIGNEMENT SUPÉRIEUR

UNIVERSITÉ DE NANTES

Thèse de Doctorat

Laurent Bulteau

Algorithmic Aspects of Genome Rearrangements

Ordre et désordre dans l'algorithmique du génome

Résumé

Dans cette thèse, nous explorons la complexité algorithmique de plusieurs problèmes issus de la génomique comparative, et nous apportons des solutions à certains de ces problèmes sous la forme d'algorithmes d'approximation ou paramétrés.

Le dénominateur commun aux problèmes soulevés est la mise en commun d'informations génomiques provenant de plusieurs espèces dans le but de tirer des conclusions pertinentes pour l'étude de ces espèces. Les problèmes de tri par transpositions et de tri par inversions préfixes permettent de retrouver l'histoire évolutive des deux espèces. Les problèmes de distance exemplaire et de plus petite partition commune ont pour but de comparer deux génomes dans les cas algorithmiquement difficiles où chaque gène apparait avec plusieurs copies indistinguables dans le génome. Enfin, les problèmes d'extraction de bandes et de linéarisation visent à préciser ou corriger l'information génomique afin qu'elle soit plus pertinente pour des traitements ultérieurs.

Les résultats principaux que nous présentons sont la NP-difficulté des problèmes de tri (par transpositions et par inversions préfixes) dont la complexité est restée longtemps une question ouverte; une étude complète de la complexité du calcul des distances exemplaires; un algorithme paramétré pour le calcul de plus petite partition commune (avec un unique paramètre étant la taille de la partition); une étude à la fois large et approfondie des problèmes d'extraction de bandes et enfin une nouvelle structure de données permettant de résoudre plus efficacement le problème de linéarisation.

Mots clés

Génomique comparative, Théorie de la complexité, Algorithmes paramétrés, Approximabilité.

Abstract

In this thesis, we explore the combinatorial complexity of several problems stemming from comparative genomics, and we provide solutions for some of these problems in the form of parameterized or approximation algorithms.

In the problems we consider, we bring together the genomic information of several species and aim at drawing relevant conclusions for the study of these species. Sorting a genome either by transpositions or by prefix reversals leads to the reconstruction of the evolution history regarding the genomes of two species. Exemplar distances and common partition aim at comparing genomes in the algorithmically complex case where duplicated genes are present. Finally, the strip recovery and linearization problems aim at correcting or completing genomic information so that it can be used for further analysis.

The main results we expose are the NP-hardness of the sorting problems (both by transpositions and by prefix reversals), of which the computational complexity has been a long-standing open question; an exhaustive study of the computational complexity of exemplar distances; a parameterized algorithm for the minimum common string partition problem; a deep and wide study of strip recovery problems; and finally a new graph structure allowing for a more efficient solving of the linearization problem.

Key Words

Comparative genomics, Computational complexity, Parameterized algorithms, Approximability.