Approximation de superchaîne, indexation et assemblage de génome

Bastien Cazaux

Collège Doctoral du Languedoc-Roussillon

THÈSE

Pour obtenir le grade de

Docteur

Délivré par l'**Université de Montpellier**

Préparée au sein de l'école doctorale **I2S**
Et de l'unité de recherche **LIRMM**

Spécialité: **Informatique**

Présentée par **Bastien Cazaux**

Lien entre structures d'indexation et problèmes sur les superchaînes

Soutenue le 7 décembre 2016 devant le jury composé de

M. Maxime Crochemore	Pr	U. Paris-Est Marne-la-Vallée/LIGM	Rapporteur
Mme Irena Rusu	Pr	U. de Nantes	Rapporteur
M. Guillaume Blin	Pr.	U. de Bordeaux/LABRI	Examinateur
M. Christophe Paul	DR	CNRS/LIRMM	Président
M. Eric Rivals	DR	CNRS/LIRMM	Directeur

Université Montpellier

Table des matières

Table des figures

Remerciements

Je tiens à remercier toutes les personnes qui m'ont permis de mener à bien ces années de recherche, ceux que je vais citer dans la suite et ceux que je risque d'oublier.

Pour commencer, je souhaite remercier mon directeur de thèse/encradrant/partenaire de badminton, Eric Rivals pour m'avoir aidé, soutenue et laissé l'indépendance dont j'avais besoin pendant ces trois ans. Il a su tout au long de ces trois années me redonner de l'énergie et structurer ma démarche quand cela s'est avéré nécessaire.

Je souhaite remercier aussi tous mes co-bureaux qui se sont succédés pour venir me tenir compagnie et me supporter. Merci à Stéphanie pour m'avoir acceuilli chaleureusement dans leur bureau. Merci à Mathieu pour sa gentillesse, Sophie pour sa douceur et son calme, Amal pour sa bienveillance, Leena et Aravind pour m'avoir forcé à parler anglais, Vincent M pour son flegme et enfin en grand merci à Damien pour mon breton préféré avec qui je partage énormenent de points communs.

Je souhaite ensuite remercier le reste de l'équipe MAB, qui restera pour moi une deuxième famille. Merci aux permanents (i.e. Laurent, Stéphane, Annie, Anne-Murielle, Vincent B, Vincent L, Alban, Francois, Olivier, Fabio, Krister, Sylvain M) pour leurs gentillesses, leurs sagesses et leurs flegmes. Merci aux non-permanents (désolé de faire la distinction) pour l'esprit de cohésion. Merci aux thésards (Manu, Elodie, Emma, Sylvain P., Christophe, May, Chloé, Clément) pour avoir fait de l'équipe MAB plus qu'une simple équipe. Merci aussi aux post-doctorants (Maxime, Raphael, Rodrigo, Mathias, Daniel, Jimmy) qui m'ont permis de me projeter vers un avenir parfois lointin.

Ensuite, je voudrais remercier toutes les autres personnes avec qui j'ai travaillé que cela soit du LIRMM ou d'autres laboratoires (et plus particuliérement Lyon, Rennes et Rouen).

Je voudrais aussi remercier les membres de mon jury de thése qui m'ont forcé à donner le meilleur de moi.

Enfin je voudrais remercier ma famille, mes amis et surtout ma petite Laure qui m'a soutenu et motivé lors de ces trois ans.

Introduction

Contexte

Le problème de la plus petite superchaîne (≪ *Shortest Linear Superstring* ≫ en anglais), noté SLS est un des problèmes les plus connus d'algorithmique du texte. Sa popularité vient de trois grandes raisons. La première est la simplicité de sa définition : pour un ensemble de mots en instance, on cherche une superchaîne de cet ensemble de mots, c'est-à-dire un nouveau mot qui contient tous les mots de l'instance et qui soit de longueur minimale.

La deuxième raison est que ce problème a de nombreuses applications. En effet, ce problème est intéressant en compression où pour un ensemble de mots, on cherche à trouver la plus petite façon de représenter cet ensemble de mots. Une des solutions possibles est de construire la plus courte superchaîne et garder en mémoire les positions de départ des mots de début dans cette superchaîne [83, 85, 84, 37]. Une autre application connue est en ordonnancement où on cherche à minimiser le temps pour effectuer un nombre de tâches différentes et dans ce cas là, les tâches sont modélisées par des mots et on cherche la façon la plus rapide pour enchaîner les tâches [98].

La troisième et dernière grande raison est que ce problème est difficile tant dans le sens de la complexité que dans celui de la compréhension. En effet, ce problème pose de manière assez naturelle de nombreuses questions pour lesquelles on peut utiliser un grand nombre d'outils théoriques. La plus connue est celle que l'on appelle la conjecture gloutonne qui nous dit que l'algorithme glouton, qui est un des algorithmes sur SLS les plus simples, a un ratio d'approximation optimal de 2, c'est-à-dire qu'il n'aura jamais de solutions de longueur plus grande que deux fois la longueur d'une solution optimale. Cette conjecture résume bien la difficulté du problème SLS. En effet, cette conjecture qui a été énoncée par Blum et al. [11] en 1991 n'a toujours pas été prouvée presque trente ans plus tard et plus important, le meilleur ratio d'approximation connu pour le problème SLS est de $2 + \frac{11}{30}$ [67], c'est-à-dire bien supérieur à celui conjecturé sur l'algorithme glouton.

Dans la littérature, on trouve ce que nous appelons des *variantes naturelles* de SLS. Citons trois exemples. Tout d'abord, une restriction de SLS pour des mots de longueur donnée [31], une variante nommée Reverse-SLS où pour chaque mot, ce mot ou son renversé doit apparaître dans la superchaîne [42], enfin une variante où l'on cherche non plus une superchaîne, mais une plus petite couverture circulaire de mots (SCCS) [69].

Historiquement, on trouve les premières traces du problème SLS en 1967 [81] et depuis, on retrouve régulièrement des papiers sur ce problème avec comme unique sujet l'amélioration du meilleur ratio d'approximation connu. La raison de cette course au meilleur ratio d'approximation est que même si ce problème est lié à l'algorithmique du texte, c'est dans l'algorithmique du graphe que l'on a le plus de papier sur le sujet. En effet le problème SLS est une application directe d'un autre problème sur les graphes qui

correspond à trouver un chemin hamiltonien, c'est-à-dire qui passe une et une seule fois par tous les sommets du graphe.

Si la question de l'approximation du problème SLS est très étudiée [11, 88, 18, 3, 4, 5, 14, 86, 44, 68, 63], peu d'articles s'intéressent à exhiber des algorithmes exacts [8, 45, 48, 36]. Plusieurs auteurs se sont consacrés à des variantes naturelles plus simples que SLS pour lesquelles ils ont pu trouver un algorithme exact [16, 31, 32, 9, 33, 39, 89], mais là aussi les études d'algorithmes d'approximation sur ces variantes abondent [9, 12, 15, 16, 18, 22, 30, 40, 41, 42, 43, 49, 53, 56, 65, 67, 68, 77, 87, 88, 90].

Une autre chose étonnante est l'étude de SCCS, une variante de SLS qui ne cherche plus la plus petite superchaîne mais la plus petite couverture circulaire de mots, c'est-à-dire un ensemble de mots circulaires. Ce problème est très important pour SLS car comme il est moins difficile que SLS, il est souvent utilisé comme première étape dans les algorithmes d'approximation de SLS. En dépit de ce lien, l'étude de SCCS se résume à un article de 1982 [69] qui donne un algorithme exact cubique. On peut alors se demander s'il existe un algorithme exact pour SCCS qui est quadratique ou même linéaire.

On peut se demander d'où vient la sur-représentation d'articles traitant d'algorithmes d'approximation par rapport à ceux consacrés aux algorithmes exacts. Une réponse se trouve sans doute dans l'application la plus répandue du problème SLS qui est l'assemblage de génomes. En effet, pour le problème de l'assemblage de génomes on dispose en entrée d'un ensemble de parties d'un génome et on essaye de retrouver la séquence du génome ciblé. Comme un génome est modélisé comme un texte sur un alphabet bien spécifique (sur $\{A, T, C, G\}$ dans le cas de l'ADN), l'assemblage de génomes est souvent cité comme une application de SLS. Effectivement, Lesk [52] montre que ce problème appliqué sur des fragments du génome conserve les structures biologiquement importantes du génome. De plus, Li [53] montre que la plus petite superchaîne de l'ensemble des fragments du génome est une bonne représentation du génome.

Vu la taille des données traitées pour assembler un génome, on ne peut raisonnablement pas utiliser un algorithme exact. En effet, un algorithme quadratique serait déjà clairement trop gourmand pour des données de séquençage à haut débit. On se tourne alors vers des algorithmes d'approximation pour SLS en temps linéaire de façon à trouver une solution en temps raisonnable. Malgré le lien entre l'algorithmique du texte et le problème de l'assemblage de génomes, il existe un énorme gap entre les algorithmes sur SLS ou ses variantes naturelles et les algorithmes utilisés pour l'assemblage de génome. Effectivement, les logiciels qui servent à assembler un génome utilisent des heuristiques qui ont été testées et améliorées à l'aide de jeux de données, et qui s'éloignent des algorithmes d'approximation connus. Une catégorie de logiciels d'assemblage effectue un parcours d'un graphe de De Bruijn [74], c'est-à-dire un graphe où on a découpé les mots de départ en k-mers. L'autre catégorie de logiciels préfère se servir d'un graphe des chevauchements, c'est-à-dire un graphe qui recèle tous les chevauchements maximaux possibles permettant de fusionner un mot avec un autre. Chaque catégorie a ses limites : le graphe de De Bruijn ne regarde qu'une partie de l'information fournie par l'ensemble de mots en entrée, tandis que le graphe des chevauchements garde lui toute l'information, mais exige un espace mémoire trop important en pratique. On peut alors se demander s'il existe un graphe intermédiaire entre le graphe de De Bruijn et celui des chevauchements, qui permettrait de conserver toute l'information sans être trop gourmand en espace. En outre, la relation entre les parcours des graphes de De Bruijn et les algorithmes d'approximation pour SLS ou ses variantes naturelles reste à explorer.

Pour combler ce gap entre calcul de superchaîne et assemblage de génomes, peut-on

prendre en compte des contraintes liées à l'assemblage dans la définition et l'étude des problèmes de superchaînes ? Par exemple, des logiciels d'assemblage comme SPAdes [7] ou IDBA [73], qui utilisent des graphes de De Bruijn de plusieurs ordres, s'interdisent de prendre en compte des chevauchements trop courts. Pour modéliser ce cas, on pourra introduire une variante de SLS qui prend en entrée un ensemble de mots et un ensemble de chevauchements interdits. Nous appellerons ces nouveaux problèmes des *variantes d'assemblage*, pour les distinguer des variantes naturelles.

En pratique, pour l'assemblage du génome, on conserve en mémoire une structure de données d'indexation pour représenter les mots de départ. En effet, une telle structure est souvent utilisée en amont de l'assemblage, par exemple pour corriger les séquences/mots d'ADN. Peut-on alors réutiliser cette structure pour construire directement le graphe de De Bruijn ou le graphe des chevauchements ? Du point de vue théorique, peut-on exploiter ces structures d'indexation pour améliorer la complexité d'un algorithme approximant SLS ou une de ses variantes ?

Dans cette thèse, on regarde le lien entre la partie pratique de l'assemblage de génome et la partie théorique des algorithmes d'algorithmique du texte sous trois angles différents.

Le premier angle est une étude générale des ratios d'approximation des algorithmes gloutons pour les variantes naturelles de SLS. Effectivement, en utilisant la théorie des matroïdes et plus exactement celle des systèmes héréditaires, on a pu généraliser l'algorithme glouton classique pour SLS à des variantes naturelles, connues ou nouvelles, comme le problème de la plus petite superchaîne circulaire ou le problème de la plus petite couverture circulaire de mots. La théorie des systèmes héréditaires, et plus exactement les travaux de Mestre [59], nous ont permis d'obtenir, en adaptant les preuves, des bornes pour les ratios d'approximation des algorithmes gloutons pour ces variantes naturelles.

Dans le deuxième angle, on examine l'apport d'une structure d'indexation pour comprendre et améliorer les différents algorithmes gloutons. Grâce aux travaux de Ukkonen [91] et de Gusfield [34], on savait que l'on pouvait utiliser une structure d'indexation pour construire l'algorithme glouton pour le problème de la plus petite superchaîne en temps linéaire en la norme de l'instance. On a généralisé cette construction et montré que pour certaines variantes naturelles de SLS, on pouvait construire un graphe, le *graphe glouton*, en temps linéaire en la norme de l'instance, que l'on ancre dans une structure d'indexation et qui résume l'ensemble des solutions gloutonnes pour ces problèmes.

Enfin, le troisième angle examiné concerne la construction du graphe de De Bruijn pour l'assemblage à partir d'une structure d'indexation. Nous montrons qu'à partir de structures telles que l'arbre des suffixes, on peut construire efficacement ce graphe dans sa version classique ou même dans sa version compactée. En outre, on peut améliorer cette construction en partant d'un arbre des suffixes tronqué, que l'on a redéfini à cette occasion. En effet, avec cette structure tronquée, la construction du graphe de De Bruijn devient linéaire en mémoire en la taille de ce graphe.

En ancrant le graphe de De Bruijn dans une structure d'indexation, on s'est alors rendu compte qu'il était un cas particulier de graphe glouton. Le graphe glouton sur l'ensemble des k-mers des mots en entrée inclut l'ensemble des chevauchements du graphe de De Bruijn : il en découle que tous les parcours du graphe de De Bruijn sont inclus dans ceux du graphe glouton. De plus, on peut construire ce graphe glouton de manière linéaire en temps et en espace en la norme de l'instance. Enfin, grâce à l'algorithme glouton pour la variante de SCCS adaptée aux parcours du graphe de De Bruijn, on a obtenu un ratio d'approximation pour cette formulation de l'assemblage.

En résumé, ces trois angles d'approche particuliers nous ont permis de donner un graphe englobant celui de De Bruijn, de montrer qu'il peut être construit en temps linéaire en la taille de l'instance, et que la solution obtenue pour cette variante d'assemblage est garantie par un ratio d'approximation.

Plan de la thèse

La thèse est divisée en trois grandes parties qui correspondent en partie aux trois angles d'approches présentés ci-dessus.

La Partie I donne les bases nécessaires à l'étude des superchaînes. Le Chapitre 1 présente les notations de base, tandis que le Chapitre 2 introduit le problème de la plus courte superchaîne linéaire et ses principales variantes naturelles. Pour chacun de ces problèmes, on donne une définition formelle ainsi que son état de l'art. En définitive, parmi les problèmes abordés dans ce chapitre, seul le problème de la plus petite superchaîne linéaire a été largement étudié. Notre étude détaillée vise à donner des définitions, des propriétés et des preuves formelles de résultats souvent connus, mais malheureusement fréquemment non prouvés et parfois utilisés à tort. Cette étude nous permettra d'appliquer certaines propriétés du problème SLS aux autres variantes naturelles de SLS.

La Partie II est consacrée à l'étude de l'algorithme glouton pour certaines variantes naturelles de SLS. Plus précisément, on étudie les ratios d'approximation optimaux de ces algorithmes gloutons pour la variante compression de chacun de ces problèmes. Dans le Chapitre 3, on définit les variantes compression des problèmes classiques de superchaînes et on étudie le lien entre ces variantes de SLS et leurs versions compression. Ce lien nous permet de trouver facilement des bornes pour les ratios d'approximation optimaux des restrictions de ces problèmes lorsque les mots sont de longueur fixée. Enfin, on reformule le lien connu entre les problèmes classiques des superchaînes et les problèmes sur les graphes (notamment le « Maximum Travelling Salesman Problem »). Dans le Chapitre 4, on présente les algorithmes gloutons pour les problèmes des superchaînes ainsi que pour les systèmes héréditaires, et on montre comment définir les premiers en fonction des seconds. Cette nouvelle façon de voir ces algorithmes gloutons facilite l'établissement de bornes sur leurs ratios d'approximation optimaux. Pour finir cette partie, dans le Chapitre 5, on applique cette nouvelle méthode pour obtenir des bornes concernant les algorithmes gloutons des variantes d'assemblage de SLS. On regarde notamment la variante du complémentaire-renversé, la variante avec multiplicités, et celle où l'on s'interdit un ensemble de chevauchements.

Dans la Partie III nous exhibons de nouveaux algorithmes pour construire les solutions gloutonnes de certaines variantes naturelles de SCCS en temps linéaire en la norme de l'instance. À cette fin, on utilise des structures de données d'indexation, déjà connues, comme par exemple l'arbre d'Aho-Corasick ou l'arbre des suffixes, et d'autres nouvelles, comme l'Aho-Corasick généralisé ou le graphe glouton. En outre, ces structures nous permettent de construire le graphe de De Bruijn et de montrer que l'on peut généraliser ce graphe pour mémoriser plus de chevauchements. Le Chapitre 6 commence par définir l'arbre d'Aho-Corasick et l'arbre des suffixes, ainsi qu'une généralisation de ce dernier, qui permet de définir un nouvel arbre des suffixes tronqué. On utilise l'arbre des suffixes et l'arbre des suffixes tronqué pour construire le graphe de De Bruijn en temps linéaire en la norme de l'instance, dans le Chapitre 9. Dans le Chapitre 7, on montre ensuite que toutes ces solutions de l'algorithme glouton pour le problème de la plus petite couverture circulaire de mots (SCCS) sont comprises dans un graphe que l'on nomme le graphe

glouton. À l'aide de ce graphe et de l'optimalité de l'algorithme glouton pour SCCS, on montre comment construire une solution optimale en temps linéaire. Dans le Chapitre 8, on applique ces résultats aux algorithmes gloutons de certaines variantes d'assemblage vues au Chapitre 5. Enfin, après avoir montré la construction du graphe de De Bruijn à l'aide des structures d'indexation, dans le Chapitre 9, on montre la relation entre graphe de De Bruijn et graphe glouton.

Listes des publications originales

Cette thèse est basée sur les publications originales suivantes. On a rajouté pour chaque article, à quelles parties du mémoire de thèse, il faisait référence.

(I) Bastien Cazaux, Rodrigo Cánovas, and Eric Rivals. Shortest DNA cyclic cover in compressed space. In *Data Compression Conference DCC*, pages 536–545. IEEE Computer Society Press, 2016.

> On retrouve les résultats de cet article dans la sous-section 5.1.3 et dans la section 8.1.

(II) Bastien Cazaux, Thierry Lecroq, and Eric Rivals. Linking indexing data structures to De Bruijn graphs : Construction and update. *Journal of Computer and System Sciences*, 2016.

> On retrouve certains résultats de cet article dans les chapitres 6 et 9.

(III) Antoine Limasset, Bastien Cazaux, Eric Rivals, and Pierre Peterlongo. Read mapping on De Bruijn graphs. *BMC Bioinformatics*, volume 17, pages 1–12, 2016.

> Le résultat de cet article n'est pas directement présenté dans la thèse.

(IV) Bastien Cazaux, Gustavo Sacomoto, and Eric Rivals. Superstring Graph : a new approach for genome assembly. In *Proc. 9th International Conference on Algorithmic Aspects in Information and Management*, page in press. Springer Verlag, 2016.

> On retrouve certains résultats de cet article dans la section 9.4.

(V) Bastien Cazaux and Eric Rivals. A linear time algorithm for Shortest Cyclic Cover of Strings. *Journal of Discrete Algorithms*, volume 37, pages 56–67, 2016.

> On retrouve certains résultats de cet article dans le chapitre 7

(VI) Bastien Cazaux and Eric Rivals. DNA Shortest Superstrings and DNA Shortest Cyclic Covers. Research report, LIRMM, 2016. submitted.

On retrouve certains résultats de cet article dans
la section 5.1 et dans la section 9.3.

(VII) Bastien Cazaux, Thierry Lecroq, and Eric Rivals. Construction of a De Bruijn Graph for Assembly from a Truncated Suffix Tree. In A.H. Dediu, editor, *Proc. of the 9th Int. Conf. on Language and Automata Theory and Applications (LATA)*, volume 8977, pages 109–120. Springer International Publishing Switzerland, 2015.

On retrouve certains résultats de cet article dans
les chapitres 6 et 9.

(VIII) Bastien Cazaux and Eric Rivals. The power of greedy algorithms for approximating Max-ATSP, Cyclic Cover, and superstrings. *Discrete Applied Mathematics*, volume 212, pages 48–60, 2016.

On retrouve certains résultats de cet article dans
le chapitre 4.

(IX) Bastien Cazaux and Eric Rivals. 3-Shortest Superstring is 2-approximable by a greedy algorithm. Research Report RR-14009, LIRMM, June 2014. accepted.

On retrouve certains résultats de cet article dans
la section 3.3.

(X) Bastien Cazaux, Thierry Lecroq, and Eric Rivals. From Indexing Data Structures to De Bruijn Graphs. In *Proc. of the 25th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 8486, pages 89–99. Springer International Publishing Switzerland, 2014.

On retrouve certains résultats de cet article dans
la section 9.2.

(XI) Bastien Cazaux and Eric Rivals. Approximation of greedy algorithms for Max-ATSP, Maximal Compression, Maximal Cycle Cover, and Shortest Cyclic Cover of Strings. In *Proc. of Prague Stringology Conference (PSC)*, pages 148–161, Czech Technical Univ Prague, 2014.

On retrouve certains résultats de cet article dans
le chapitre 4.

(XII) Bastien Cazaux and Eric Rivals. Reverse engineering of compact suffix trees and links : A novel algorithm. *Journal of Discrete Algorithms*, volume 28, pages 9–22, 2014.

On retrouve certains résultats de cet article dans
la sous-section 7.3.

Première partie

Notations et problèmes des superchaînes

Chapitre 1

Notations usuelles

Dans ce premier chapitre, on va définir les notations que l'on va utiliser par la suite. On va commencer par reprendre rapidement les définitions d'ensemble et de uplet pour pouvoir définir ensuite les chaînes et les chaînes circulaires. Ensuite, on va parler partiellement des permutations car elles vont avoir plusieurs utilités (pour construire des superchaînes, pour définir le complémentaire renversé (ou « Reverse-Complement » en anglais), ...). Enfin on va aborder brièvement les problèmes en général et leur complexité. Ce chapitre va nous permettre de donner les notations de base pour les chapitres suivants.

Sommaire

1.1 Ensemble et uplet

Un *ensemble* est une collection d'objets distincts. Un objet de cet ensemble est appelé un *élément* de l'ensemble. Soient E un ensemble et x un objet, on note $x \in E$, le fait que x soit un élément de E. On dit qu'un ensemble est *fini* si le nombre de ses éléments est borné par un entier, il est *infini* dans l'autre cas. Soit E un ensemble fini, le *cardinal* de E, noté $|E|$, est le nombre d'éléments de E. L'*ensemble vide*, noté $\emptyset$, est l'ensemble avec aucun élément.

Exemple 1.1 L'ensemble des entiers positifs $\mathbb{N}$ est un ensemble infini, alors que l'intervalle d'entier entre a et b avec $a \leq b$, noté $\{a, \ldots, b\}$, est un ensemble fini de cardinal $b - a + 1$.

Un *sous-ensemble* d'un ensemble E est un ensemble E' tel que tous les éléments de E' sont des éléments de E. Par convention, l'ensemble vide est un sous-ensemble de chaque ensemble. On écrit par $E' \subseteq E$, le fait que E' soit un sous-ensemble de E. Une *partition* d'un ensemble E est un ensemble $\{E_1, \ldots, E_k\}$ de sous-ensembles de E tel que $\bigcup_{i=1}^{k} E_i = E$ et pour tout $i \neq j$, $E_i \cap E_j = \emptyset$.

Exemple 1.2 L'ensemble des entiers positifs $\mathbb{N}$ est un sous-ensemble de l'ensemble des entiers relatifs $\mathbb{Z}$. L'ensemble $\{\{0\}, \{-1, 1\}, \ldots, \{-i, i\}, \ldots\}$ est une partition de $\mathbb{Z}$.

Une *relation d'ordre* $\leq$ sur un ensemble E est une relation qui relie deux éléments de l'ensemble et qui vérifie les trois conditions suivantes. Pour x, y et z éléments de E :

1. $x \leq x$,
2. si $x \leq y$ et $y \leq x$ alors $x = y$,
3. si $x \leq y$ et $y \leq z$ alors $x \leq z$. Par souci de concision, on note $x \leq y \leq z$ les conditions $x \leq y$ et $y \leq z$.

Le couple $(E, \leq)$ est appelé un *ensemble ordonné*. On pose $<$ la relation d'ordre stricte relative à un ordre $\leq$ telle que pour deux éléments a et b de E, $a < b$ si et seulement si $a \leq b$ et $a \neq b$. On dit que la relation d'ordre $\leq$ sur E est *totale* si pour tout x, y de E, $x \leq y$ ou $y \leq x$. On dit alors que $(E, \leq)$ est un *ensemble totalement ordonné*.

Exemple 1.3 Le couple $(\mathbb{N}, \leq)$ est un ensemble totalement ordonné où $\leq$ est la relation d'ordre "plus petite ou égale" définie sur $\mathbb{N}$.

Soient $(E, \leq)$ un ensemble totalement ordonné et A un sous-ensemble de E. On note $max(A)$ (respectivement $min(A)$) l'élément a de A tel que $\forall x \in A, x \leq a$ (respectivement $\forall x \in A, a \leq x$).

Soient E un ensemble et n un entier. Un *n-uplet* ou *n-uplet linéaire* de E est une application de $\{1, \ldots, n\}$ vers E. Pour i entre 1 et n, le *caractère* d'un n-uplet u à la position i est l'image de u par i, *i.e.* $u(i)$. Pour i et j deux entiers tels que $1 \leq i \leq j \leq n$, l'*intervalle* du n-uplet u entre i et j, noté $u[i, j]$, est le $(j - i + 1)$-uplet de E tel que :

$$\begin{array}{rccl} u[i,j] : & \{1, \ldots, j-i+1\} & \to & E \\ & k & \mapsto & u(i+k-1) \end{array}$$

Soient u un n-uplet et i un entier strictement plus petit que n. Le *caractère suivant* d'un caractère à la position i est le caractère à la position $i + 1$. On a de plus que le caractère à la position n n'a pas de caractère suivant, c'est le *dernier élément* du n-uplet.

Soient u_1 un n_1-uplet sur E et u_2 un n_2-uplet sur E. La *concaténation* de u_1 et de u_2, notée $u_1.u_2$, est le $(n_1 + n_2)$-uplet sur E tel que :

$$\begin{array}{rccll} u_1.u_2 : & \{1, \ldots, n_1+n_2\} & \to & E & \\ & k & \mapsto & u_1(k) & \text{Si } 1 \leq k \leq n_1 \\ & & & u_2(k) & \text{Si } n_1+1 \leq k \leq n_1+n_2 \end{array}$$

1.2 Chaîne/Mot

Un *alphabet* est un ensemble totalement ordonné fini ou infini. Une *chaîne* ou *chaîne linéaire* ou *mot* sur un alphabet Σ de *longueur* n est un n-uplet sur Σ. La longueur d'un

mot w, que l'on note $|w|$, est alors le nombre de caractères du mot. Soit Σ un alphabet, on note Σ^k l'ensemble des mots de longueur k sur Σ et $\Sigma^* := \bigcup_{k=0}^{\infty} \Sigma^k$ l'ensemble des mots sur Σ où on pose $\Sigma^0 := \{\varepsilon\}$ où ε est le mot vide. L'ensemble des mots Σ^* est aussi appelé *langage*. Pour un ensemble de mots P, la *norme* de P, notée $||P||$, est la somme des longueurs des mots de P, *i.e.*

$$||P|| = \sum_{w \in P} |w|.$$

Soient w un mot sur Σ de longueur n et i et j, tel que $0 \leq i \leq j \leq n-1$. La *sous-chaîne* entre i et j de w, notée $w[i, j]$, est l'intervalle du n-uplet w entre i et j.

Exemple 1.4 Soient $\Sigma = \{a, b, c\}$ un alphabet et w_1 le mot sur Σ de longueur 4 tel que

$$\begin{array}{rccc} w_1 : & \{1, \ldots, 4\} & \rightarrow & \{a, b, c\} \\ & 1 & \mapsto & a \\ & 2 & \mapsto & b \\ & 3 & \mapsto & b \\ & 4 & \mapsto & a \end{array}$$

On note alors $w_1 = w_1(1)w_1(2)w_1(3)w_1(4) = abba$. Le mot $w_2 = w_1(2)w_1(3) = bb$ est la sous-chaîne de w_1 entre les positions 2 et 3 ($w_2 = w_1[2, 3]$).

Soit i un entier plus petit ou égal que la longueur de w. Le *préfixe* de w finissant à la position i est la sous-chaîne de w entre les positions 1 et i, *i.e.* $w[1, i]$. Le *suffixe* de w commençant à la position i est la sous-chaîne de w entre les positions i et $|w|$, *i.e.* $w[i, |w|]$. On dit qu'un préfixe (ou un suffixe) est *propre* s'il est différent du mot initial et du mot vide ε. Comme un mot de longueur n est un n-uplet, on peut utiliser la concaténation sur les mots : soient w_1 et w_2 deux mots, la concaténation de w_1 et de w_2 est le mot $w_1.w_2$ de longueur $|w_1| + |w_2|$.

Exemple 1.5 Soient $\Sigma = \{a, b\}$, $w_1 := abba$ et $w_2 := babb$. On a alors que $w_1.w_2 = abba.babb = abbababb$. De plus, abb est un préfixe de w_1 et un suffixe de w_2, mais ba est un préfixe de w_2 mais pas un suffixe de w_1.

Pour un mot w, on pose $Fact(w)$ comme l'ensemble des sous-chaînes de w, $Fact_k(w)$ comme l'ensemble des sous-chaînes de longueur k de w, $Suffixe(w)$ comme l'ensemble des suffixes de w et $Prefixe(w)$ comme l'ensemble des préfixes de w.

1.3 Comparaison entre deux mots

Pour deux mots x et y, un *chevauchement* de x sur y est un suffixe propre de x qui est aussi un préfixe propre de y. Un chevauchement de x sur y est dit *maximal* s'il correspond au plus long chevauchement de x sur y.

Remarque 1.6 Il est intéressant de noter que le chevauchement est asymétrique, c'est-à-dire que si w est un chevauchement de x sur y, cela ne veut pas dire que w est un chevauchement de y sur x (voir Exemple 1.7). Dans la littérature, on parle souvent de chevauchement entre deux mots, ce qui suppose la symétrie de la définition et qui peut emmener à l'erreur.

Exemple 1.7 Soient $aabb$ et $bbaa$. L'ensemble des chevauchements de $aabb$ sur $bbaa$ est $\{bb, b\}$ et l'ensemble des chevauchements de $bbaa$ sur $aabb$ est $\{aa, a\}$.

On note $ov(x, y)$ le chevauchement maximal de x sur y. Le *préfixe* de x sur y, noté $pr(x, y)$ est alors le préfixe de x restant lorsqu'on a enlevé de x le suffixe de $ov(x, y)$, *i.e.* $x = pr(x, y).ov(x, y)$. De même, le *suffixe* de x sur y, noté $su(x, y)$ est alors le suffixe de y qui ne correspond pas au chevauchement maximal de x sur y, *i.e.* $y = ov(x, y).su(x, y)$. Enfin, la *fusion* de x sur y est le mot $x \oplus y := pr(x, y).ov(x, y).su(x, y)$.

Soient Σ un alphabet et $\leq$ l'ordre total sur Σ. L'*ordre lexicographique* $\preceq$ associé à $\leq$ est un ordre total sur l'ensemble des n-uplets Σ^n, tels que pour u et v deux n-uplets :

$$u \preceq v \quad \text{si et seulement si} \quad u(0) < v(0) \text{ ou } (u(0) = v(0) \text{ et } u(1, n) \preceq v(1, n))$$

On peut étendre cet ordre lexicographique à l'ensemble des uplets Σ^* (on ne fixe plus la longueur). Soient u un k_1-uplet et v un k_2-uplet :

$$u \preceq v \quad \text{si et seulement si} \quad u(0, l) \prec v(0, l) \text{ ou } (u(0, l) = v(0, l) \text{ et } k_1 \leq k_2)$$

où l est le minimum entre k_1 et k_2.

Exemple 1.8 Soient $\Sigma = \{a, b, c\}$ un alphabet et $w_1 = \mathbf{bababb}$ et $w_2 = \mathbf{bbaa}$. On a que $ov(w_1, w_2) = \mathbf{bb}$, $pr(w_1, w_2) = \mathbf{baba}$, $su(w_1, w_2) = \mathbf{aa}$ et $fu(w_1, w_2) = \mathbf{bababbaa}$. On a de plus que $w_1 \preceq w_2$.

De plus, on peut définir un autre ordre sur l'ensemble Σ^* : l'*ordre de sous-chaîne*, noté $\underset{\text{sub}}{\subset}$. Cet ordre de sous-chaîne est défini tel que, pour u et v deux mots de Σ^* :

$$u \underset{\text{sub}}{\subset} v \quad \text{si et seulement si} \quad u \text{ est une sous-chaîne de } v.$$

L'ordre lexicographique est un ordre total sur Σ^*, alors que l'ordre de sous-chaîne ne l'est pas (voir Exemple 1.9).

Exemple 1.9 Soient $abbba$ et bb. On a que $bb \underset{\text{sub}}{\subset} abbba$. Soient abb et bba. Aucun de ces deux mots n'est sous-chaîne de l'autre.

1.4 Uplet circulaire

Dans la section 1.5, on introduit la notion de chaîne circulaire. Pour définir correctement les chaînes circulaires, on va utiliser un concept mathématique de l'algèbre qui est l'anneau $\mathbb{Z}/n\mathbb{Z}$, et plus exactement on va utiliser le groupe $(\mathbb{Z}/n\mathbb{Z}, +_n)$.

Comme le sujet ici n'est pas de redéfinir l'algèbre mais juste d'utiliser le concept, on ne va pas entrer dans les détails. Pour nous, l'ensemble $\mathbb{Z}/n\mathbb{Z}$ est l'ensemble des restes possibles de la division d'un nombre par n, c'est-à-dire que l'ensemble $\mathbb{Z}/n\mathbb{Z}$ est isomorphe à $\{0, \ldots, n-1\}$ et donc à $\{1, \ldots, n\}$ (par l'application $x \mapsto x + 1$).

L'addition $\oplus_n$ sur $\mathbb{Z}/n\mathbb{Z}$ est telle que pour deux éléments x et y de $\mathbb{Z}/n\mathbb{Z}$, $x \oplus_n y$ est le reste de la division par n de $x + y$. On pose alors $+_n$ l'addition sur $\{1, \ldots, n\}$ telle que pour $x, y \in \{1, \ldots, n\}$,

$$x +_n y = \begin{array}{ll} x + y & \text{Si } x + y \leq n \\ x + y - n & \text{Si } x + y > n \end{array}$$

Exemple 1.10 Soient 4 et 3 deux éléments de $\mathbb{Z}/6\mathbb{Z}$. On a alors que $4 \oplus_6 3 = 1$. Pour les deux éléments 4 et 3 de $\{1, \ldots, n\}$, on a que $4 +_6 3 = 1$ mais que $5 \oplus_6 1 = 0$ et $5 +_6 1 = 6$.

Soient E un ensemble et n un entier. Un *n-uplet circulaire* est une application de $(\{1, \ldots, n\}, +_n)$ vers E.

De la même façon que pour un n-uplet, on définit le concept de *caractère* et d'*intervalle* pour un n-uplet circulaire. Par contre, pour i entre 1 et n, le *caractère suivant* du caractère à la position i est le caractère à la position $(i +_n 1)$. On a donc que le caractère suivant du caractère à la position n est le caractère à la position 1.

1.5 Chaîne circulaire

Une *chaîne circulaire* ou *mot circulaire* de longueur n sur un alphabet Σ est un n-uplet circulaire sur Σ.

À partir d'un mot linéaire, on peut construire un mot circulaire de telle sorte que le caractère suivant du dernier caractère du mot linéaire est son premier caractère : pour w un mot linéaire sur Σ, on pose $\langle w \rangle$ le *mot circulaire associé* à w sur Σ tel que :

$$\begin{array}{rccc} \langle w \rangle & : & \{1, \ldots, n\} & \to & \Sigma \\ & & k & \mapsto & w(k) \end{array}$$

Comme pour tout mot circulaire c de longueur n, on peut trouver un mot linéaire w tel que $c = \langle w \rangle$, dans la suite et dans un effort de clarté, on utilisera la notation w pour un mot linéaire et $\langle w \rangle$ pour un mot circulaire.

De la même façon que l'on passe d'un mot linéaire à un mot circulaire, on peut passer d'un mot circulaire à un ensemble de mots linéaires. Soient $\langle w \rangle$ un mot circulaire de taille n et i un entier entre 1 et n. La *représentation linéaire* de $\langle w \rangle$ commençant à la position i, notée $lin_i(\langle w \rangle)$, est telle que :

$$\begin{array}{rccc} lin_i(\langle w \rangle) & : & \{1, \ldots, n\} & \to & \Sigma \\ & & k & \mapsto & \langle w \rangle(k +_n i) \end{array}$$

Soient w_1 et w_2 deux mots sur Σ. On dit que w_2 est un *décalage circulaire* de w_1 s'il existe j entre 1 et n tel que $w_2 = lin_j(\langle w_1 \rangle)$. En posant $Lin(\langle w \rangle) = \{lin_i(\langle w \rangle) \mid 0 \leq i \leq n-1\}$, on a que pour tout j entre 1 et n, $Lin(\langle w \rangle) = Lin(\langle lin_j(\langle w \rangle) \rangle)$. De manière générale, on dit que deux mots circulaires $\langle w_1 \rangle$ et $\langle w_2 \rangle$ sont *égaux* si $Lin(\langle w_1 \rangle) = Lin(\langle w_2 \rangle)$.

Remarque 1.11 On peut partitionner l'ensemble des mots linéaires Σ^* en prenant comme relation d'équivalence le fait qu'un mot est le décalage circulaire d'un autre. On a alors que les classes d'équivalence sont de la forme $Lin(\langle w \rangle)$ où $\langle w \rangle$ est un mot circulaire. Parmi tous les représentants de ces classes d'équivalence, le mot de Lyndon [55] en est le plus connu. Le mot de Lyndon est le mot le plus petit au sens de l'ordre lexicographique parmi tous ses décalés circulaires. Le mot de Lyndon de $Lin(\langle w \rangle)$ est alors $min(Lin(\langle w \rangle))$. L'utilisation des mots de Lyndon a notamment été utilisée par Mucha [63] pour améliorer la borne d'approximation d'algorithmes pour le problème de la plus petite superchaîne.

Exemple 1.12 Soit $\langle w \rangle = \langle abbbab \rangle$ un mot circulaire (voir figure 1.1). On a que $Lin(\langle abbbab \rangle) = \{abbbab, bbbaba, bbabab, babab, ababbb, babbba\}$, $bbabab$ est un décalage circulaire de $abbbab$ et que $\langle abbbab \rangle$ et $\langle bbabab \rangle$ sont égaux.

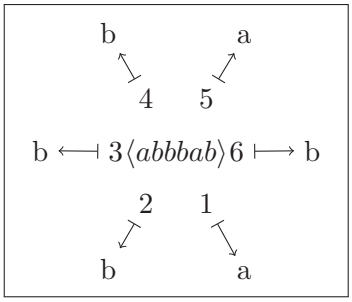


FIGURE 1.1 – Exemple de mot circulaire : $\langle abbbab \rangle$. Le caractère suivant de b à la position 6 est le a à la position 1. Pour trouver le caractère suivant, il faut regarder le prochain dans le sens des aiguilles d'une montre.

On peut généraliser l'ordre de sous-chaîne des chaînes linéaires au cas des chaînes circulaires. Pour un mot linéaire w_l et un mot circulaire $\langle w_c \rangle$, on a

$$w_l \underset{\text{sub}}{\subset} \langle w_c \rangle \quad \text{si et seulement si} \quad \begin{array}{l} \text{il existe } j \text{ entre 1 et } |\langle w_c \rangle| \\ \text{tel que } w_l \underset{\text{sub}}{\subset} (lin_j(\langle w_c \rangle))^\infty, \\ \text{où } w^\infty = w.w.\ldots \end{array}$$

Exemple 1.13 Soient $abaabaab$ un mot linéaire et $\langle baa \rangle$ un mot circulaire. On a alors que $abaabaab \underset{\text{sub}}{\subset} \langle baa \rangle$ car $abaabaab \underset{\text{sub}}{\subset} baabaabaabaa = (baa)^4 \underset{\text{sub}}{\subset} (baa)^\infty$.

1.6 Permutation

Une *permutation* d'un ensemble fini E est une bijection de E sur lui-même. Une permutation de $\{1, \ldots, n\}$ peut être alors vue comme un n-uplet de $\{1, \ldots, n\}$ qui a comme image $\{1, \ldots, n\}$ (tous les éléments de $\{1, \ldots, n\}$ sont vus). En effet, une application surjective entre deux ensembles finis de mêmes cardinaux est une bijection. Soit σ une permutation de E. On note $\sigma = (e_1, \ldots, e_{|E|})$ où pour tout i entre 1 et $|E|$, $\sigma(i) = e_i$ et $E = \{e_1, \ldots, e_{|E|}\}$.

Soit σ une permutation d'un ensemble fini E. Un *successeur* d'un élément x de E par σ est un autre élément y de E tel qu'il existe un entier k tel que $y = \sigma^k(x)$, où $\sigma^1(x) = \sigma(x)$ et $\sigma^k(x) = \sigma^{k-1}(\sigma(x))$ (on a alors que $\sigma^k = \underbrace{\sigma \circ \ldots \circ \sigma}_{k \text{ fois}}$).

Pour un élément x de E, on pose $Succ_\sigma(x)$ l'ensemble des successeurs de x par σ. On a alors que pour tout $y \in Succ_\sigma(x)$, $Succ_\sigma(x) = Succ_\sigma(y)$.

La *partition induite par une permutation* σ de E, notée $Part_\sigma$, est une partition $\{E_1, \ldots, E_p\}$ telle que pour tout i entre 1 et p et tout x de E_i, on ait $Succ_\sigma(x) = E_i$.

On dit qu'une permutation σ de E est *circulaire* si le cardinal de $Part_\sigma$ est égal à 1. Soit σ_c une permutation circulaire de E, on note $\sigma_c = (e_1 \to \ldots \to e_k)$ où $Part_{\sigma_c} = \{\{e_1, \ldots, e_{|E|}\}\}$, pour tout i entre 1 et $|E|$, $\sigma_c(e_i) = e_{i+_{|E|}1}$ et $E = \{e_1, \ldots, e_{|E|}\}$.

Exemple 1.14 Soit $\sigma_1 = (3, 1, 2, 4)$ une permutation de $\{1, \dots, 4\}$. On a alors

$$\begin{array}{rccc} \sigma_1 : & \{1, \dots, 4\} & \to & \{1, \dots, 4\} \\ & 1 & \mapsto & 3 \\ & 2 & \mapsto & 1 \\ & 3 & \mapsto & 2 \\ & 4 & \mapsto & 4 \end{array}$$

On a $Part_{\sigma_1} = \{\{1, 2, 3\}, \{4\}\}$. Pour $\sigma_2 = (2, 4, 1, 3)$, la permutation σ_2 est circulaire et on peut l'écrire $\sigma_2 = (1 \to 2 \to 4 \to 3)$.

Soit σ une permutation de E. On peut alors décomposer σ en permutations circulaires : $\sigma = \sigma_1 \dots \sigma_p$ où

$$Part_\sigma = \{\{e_1^1, \dots, e_{n_1}^1\}, \dots, \{e_1^p, \dots, e_{n_p}^p\}\},$$

telle que $\{\{e_1^1\}, \dots, \{e_{n_1}^1\}, \dots, \{e_{n_p}^p\}\} = E$ et pour tout i entre 1 et p, σ_i est la permutation circulaire telle que

$$Part_{\sigma_i} = \{\{e_1^i, \dots, e_{n_i}^i\}\}$$

et pour tout x de $\{e_1^i, \dots, e_{n_i}^i\}$, $\sigma_i(x) = \sigma(x)$.

Cette décomposition est unique et est appelée la *décomposition en permutations circulaires* de σ.

Exemple 1.15 La décomposition en permutations circulaires de $(1, 3, 4, 0, 2, 5)$ est $(1, 3, 4, 0, 2, 5) = (0 \to 1 \to 3)(2 \to 4)(5)$.

L'*ordre* d'une permutation σ de E, noté $ordre(\sigma)$ est le plus petit entier k tel que $\sigma^k = id$ où id_E est la permutation identité de E (pour tout $x \in E$, $id(x) = x$).

Remarque 1.16 Attention l'ordre d'une permutation n'est pas forcément le cardinal de l'ensemble de la partition induite par σ qui a le plus d'éléments, i.e $ordre(\sigma) \neq max(\{|E_i| \mid E_i \in Part_\sigma\})$ (voir Exemple 1.17).

Exemple 1.17 Soit $\sigma = (2, 3, 1, 5, 4) = (1 \to 2 \to 3)(4 \to 5)$, on a alors $Part_\sigma = \{\{1, 2, 3\}, \{4, 5\}\}$. L'ordre de σ est 6 :

- $\sigma^2 = (3, 1, 2, 4, 5) = (1 \to 3 \to 2)(4)(5)$
- $\sigma^3 = (1, 2, 3, 5, 4) = (1)(2)(3)(4 \to 5)$
- $\sigma^4 = (2, 3, 1, 4, 5) = (1 \to 2 \to 3)(4)(5)$
- $\sigma^5 = (3, 1, 2, 5, 4) = (1 \to 3 \to 2)4 \to 5)$
- $\sigma^6 = (1, 2, 3, 4, 5) = (1)(2)(3)(4)(5) = id_{\{1,\dots,5\}}$.

1.7 Fonctions sur les mots

Dans la sous-section 5.1.3, on va parler du problème de recherche de la plus petite superchaîne appliqué au complémentaire renversé et dans la section 6.1, on va utiliser l'ensemble des mots renversés d'un ensemble de mots. Dans cette section, on va donner

une généralisation du complémentaire renversé et du renversé, ce qui va nous permettre de définir les notations correspondantes.

Soient Σ un alphabet fini et P un ensemble de mots de Σ^*. Pour une permutation φ d'ordre au plus 2 de Σ (pour tout x de Σ, $\varphi^2(x) = x$), on définit l'application $\overleftarrow{\square}^{\varphi}$ de Σ^* vers Σ^* telle que

$$\overleftarrow{u_1 \dots u_m}^{\varphi} = \varphi(u_m) \dots \varphi(u_1)$$

On obtient alors cette propriété importante :

Proposition 1.18 Soient u et v deux mots linéaires. On a alors que $ov(\overleftarrow{u}^{\varphi}, \overleftarrow{v}^{\varphi}) = \overleftarrow{ov(v,u)}^{\varphi}$.

Preuve Soient x et y deux mots de Σ^*. On a que $\overleftarrow{xy}^{\varphi} = \overleftarrow{y}^{\varphi}\overleftarrow{x}^{\varphi}$. On peut décomposer $\overleftarrow{u}^{\varphi} = \overleftarrow{pr(u,v)ov(u,v)}^{\varphi} = \overleftarrow{ov(u,v)}^{\varphi}\overleftarrow{pr(u,v)}^{\varphi}$ et $\overleftarrow{v}^{\varphi} = \overleftarrow{ov(u,v)su(u,v)}^{\varphi} = \overleftarrow{su(u,v)}^{\varphi}\overleftarrow{ov(u,v)}^{\varphi}$. Alors on a que $\overleftarrow{ov(u,v)}^{\varphi}$ est une sous-chaîne de $ov(\overleftarrow{v}^{\varphi}, \overleftarrow{u}^{\varphi})$. Comme $|\overleftarrow{ov(u,v)}^{\varphi}| = |ov(u,v)|$, $ov(\overleftarrow{v}^{\varphi}, \overleftarrow{u}^{\varphi}) \geq |ov(u,v)|$. Comme $\overleftarrow{\overleftarrow{u}^{\varphi}}^{\varphi} = u$, $|ov(u,v)| = |ov(\overleftarrow{v}^{\varphi}, \overleftarrow{u}^{\varphi})|$ et alors $\overleftarrow{ov(u,v)}^{\varphi} = ov(\overleftarrow{v}^{\varphi}, \overleftarrow{u}^{\varphi})$. ■

Cette notation généralise les deux concepts de renversé et de complémentaire renversé et permet de définir formellement ces applications.

— Pour l'application identité id_Σ sur l'alphabet Σ et un mot w, on pose $w^{-1} = \overleftarrow{w}^{id_\Sigma}$.

— Pour la permutation c d'ordre 2 de $\{A, T, C, G\}$ telle que $c(A) = T$, $c(T) = A$, $c(C) = G$ et $c(G) = C$ et un mot w, on pose $\overleftarrow{w} = \overleftarrow{w}^{c}$.

1.8 Complexité

Dans cette partie, on va faire quelques rappels sur la complexité, sans trop entrer dans les détails. Pour plus d'informations, on pourra se référer à [26].

1.8.1 Définitions sur les problèmes

Sans entrer trop dans les détails, un *problème* est une question que l'on essaye de résoudre relativement à une entrée. On dit qu'un problème est de *décision* si la réponse est *oui* ou *non*. Un *problème d'optimisation* est un problème où on cherche une solution qui minimise ou maximise une mesure.

Exemple 1.19 Le problème pour un ensemble de mots P de trouver une sous-chaîne commune à tous les mots de P est un problème en informatique théorique. Le problème qui demande de trouver la plus petite sous-chaîne commune à un ensemble de mots est un problème d'optimisation. Enfin le problème qui pour un ensemble de mots P et un entier k cherche à savoir s'il existe une sous-chaîne de longueur plus grande ou égale à k commune à tous les mots de P est un problème de décision. Pour tous ces problèmes, l'entrée est identique, c'est un ensemble de mots.

Un *algorithme* sur un problème $\mathcal{P}$ donne une réponse à la question du problème $\mathcal{P}$. Cette réponse peut correspondre à la solution attendue par le problème alors on dit que

cet algorithme est *exact*. Dans certain cas, par exemple pour un problème d'optimisation qui maximise, si un algorithme nous donne une solution qui n'est pas la plus optimale des solutions mais une solution qui n'est pas si éloigné, on peut s'en contenter. On va alors chercher à savoir à quel point cet algorithme nous donne des solutions éloignées de la solution la plus optimale. Dans le cas d'un problème d'optimisation $\mathcal{P}$, la solution attendue de ce problème est appelée *solution optimale* de $\mathcal{P}$. On note alors $OPT_{\mathcal{P}}(\mathcal{I})$ l'ensemble des solutions optimales du problème $\mathcal{P}$ pour l'entrée $\mathcal{I}$. On note de plus $SOL_{\mathcal{P}}(\mathcal{I})$ l'ensemble des solutions pas forcément optimales du problème $\mathcal{P}$ pour l'entrée $\mathcal{I}$. On a alors de manière immédiate que $OPT_{\mathcal{P}}(\mathcal{I}) \subseteq SOL_{\mathcal{P}}(\mathcal{I})$. Pour un algorithme $\mathcal{A}$ sur $\mathcal{P}$, on pose $SOL_{\mathcal{A}}(\mathcal{I})$, l'ensemble des solutions générées par l'algorithme $\mathcal{A}$. On a alors de manière immédiate que $SOL_{\mathcal{A}}(\mathcal{I}) \subseteq SOL_{\mathcal{P}}(\mathcal{I})$. On a de plus que si un algorithme $\mathcal{A}$ est exact sur $\mathcal{P}$, on a alors que $SOL_{\mathcal{A}}(\mathcal{I}) \subseteq OPT_{\mathcal{P}}(\mathcal{I})$. Enfin, on dit qu'un algorithme $\mathcal{A}$ sur un problème $\mathcal{P}$ est *déterministe* si pour toute entrée $\mathcal{I}$ de $\mathcal{P}$, $|SOL_{\mathcal{A}}(\mathcal{I})| = 1$.

Soient $\mathcal{P}$ un problème d'optimisation et $\mathcal{A}$ un algorithme sur $\mathcal{P}$. On peut définir ce qu'est un *ratio d'approximation* et ce qu'est un ratio d'approximation *optimal* (la définition va dépendre de la nature de $\mathcal{P}$: maximisation ou minimisation).

— Si $\mathcal{P}$ est un problème de minimisation, un ratio d'approximation α de $\mathcal{A}$ est un réel tel que

$$|w_{\mathcal{A}}| \leq \alpha \times |w_{opt}|$$

pour toute solution $w_{\mathcal{A}}$ de l'algorithme $\mathcal{A}$ et pour toute solution optimale w_{opt} de $\mathcal{P}$ où $|.|$ est la mesure qu'on minimise dans $\mathcal{P}$. Un ratio d'approximation α de $\mathcal{A}$ est optimal si pour tout ratio d'approximation α' de $\mathcal{A}$, on a que $\alpha \leq \alpha'$.

— Si $\mathcal{P}$ est un problème de maximisation, un ratio d'approximation β de $\mathcal{A}$ est un réel tel que

$$|w_{\mathcal{A}}| \geq \beta \times |w_{opt}|$$

pour toute solution $w_{\mathcal{A}}$ de l'algorithme $\mathcal{A}$ et pour toute solution optimale w_{opt} de $\mathcal{P}$ où $|.|$ est la mesure qu'on maximise dans $\mathcal{P}$. Le ratio d'approximation β de $\mathcal{A}$ est optimal si pour tout ratio d'approximation β' de $\mathcal{A}$, on a que $\beta \geq \beta'$.

Remarque 1.20 Dans le cas d'un problème d'optimisation de maximisation ou de minimisation, plus le ratio d'approximation optimal d'un algorithme est proche de 1 plus les solutions de l'algorithme sont considérées comme « meilleures », *i.e.* en taille proche de la taille d'une solution optimale. Si le ratio d'approximation optimal d'un algorithme est de 1, alors cet algorithme est exact.

1.8.2 Complexité des problèmes

Pour un algorithme $\mathcal{A}$ sur un problème $\mathcal{P}$, la *complexité* de $\mathcal{A}$ va correspondre à son temps de calcul en fonction de la taille de l'entrée et de la solution (la complexité peut aussi dépendre de paramètres structurels comme pour les algorithmes FPT). Par exemple, on dit qu'un algorithme est polynomial si son temps de calcul est en $O(P(n+m))$ où n est la taille de l'entrée, m est la taille de la solution et $P(X)$ est un polynôme en X.

Il existe alors des classes de problèmes qui correspondent à leurs difficultés (voir Figure 1.2).

— Le plus simple est l'ensemble **P** qui correspond à l'ensemble des problèmes ayant un algorithme polynomial qui résout le problème.

— L'ensemble **NP** est l'ensemble des problèmes où on peut vérifier en temps polynomial qu'une solution résout bien le problème.

— L'ensemble **NP-Difficile** est l'ensemble des problèmes tels que si on trouve un algorithme polynomial pour ce problème, on peut produire un algorithme polynomial pour l'ensemble des problèmes de **NP**.

— L'ensemble **NP-complet** est l'ensemble des problèmes de décision de **NP** qui sont aussi dans **NP-Difficile**.

— L'ensemble **APX** est l'ensemble des problèmes d'optimisation qui sont dans **NP** et tels que pour chaque problème d'optimisation il existe un algorithme polynomial avec un ratio d'approximation constant pour ce problème.

— L'ensemble **APX-Difficile** (voir aussi **Max-SNP-Difficile** [70]) est l'ensemble des problèmes d'optimisation de **APX** qui n'admettent pas un ratio aussi proche que l'on veut de 1.

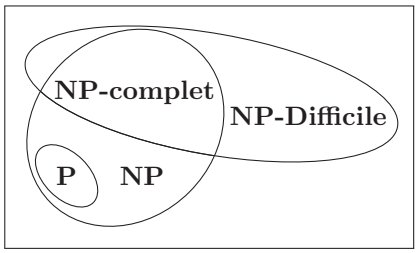


FIGURE 1.2 – Diagramme d'Euler des classes de complexité **P**, **NP**, **NP-complet** et **NP-Difficile**.

1.8.3 Variantes d'un problème d'optimisation

À un problème d'optimisation, on peut lui associer différentes variantes. Soit $\mathcal{P}$ un problème d'optimisation. On peut alors définir un problème $\mathcal{P}_d$ de décision qui correspond à ce problème. Avant cela, définissons le problème d'optimisation sur la longueur de la sortie relatif à $\mathcal{P}$ que l'on va noter $\mathcal{P}_l$.

Le problème $\mathcal{P}_l$ va être pour une entrée de $\mathcal{P}$ de trouver la taille des solutions optimales de $\mathcal{P}$. Il est clair que le problème $\mathcal{P}_l$ est un problème d'optimisation de même nature que $\mathcal{P}$, *i.e.* si $\mathcal{P}$ est un problème de maximisation (respectivement de minimisation), $\mathcal{P}_l$ est un problème de maximisation (respectivement de minimisation). De plus, si on sait résoudre $\mathcal{P}$, on sait alors résoudre $\mathcal{P}_l$.

On va alors définir $\mathcal{P}_d$ en fonction de la nature de $\mathcal{P}$ et de $\mathcal{P}_l$.

— Si $\mathcal{P}$ et $\mathcal{P}_l$ sont des problèmes de maximisation, le problème $\mathcal{P}_d$ est alors de savoir pour une entrée de $\mathcal{P}$ et un entier k, s'il existe une solution de $\mathcal{P}$ de taille plus grande ou égale à k.

— Si $\mathcal{P}$ et $\mathcal{P}_l$ sont des problèmes de minimisation, le problème $\mathcal{P}_d$ est alors de savoir pour une entrée de $\mathcal{P}$ et un entier k, s'il existe une solution de $\mathcal{P}$ de taille plus petite ou égale à k.

On a alors que les problèmes $\mathcal{P}_l$ et $\mathcal{P}_d$ sont de même complexité. En effet, une liste d'un ensemble borné de solutions de $\mathcal{P}_d$ nous donne la solution de $\mathcal{P}_l$ et inversement. On a alors la proposition suivante :

Proposition 1.21 Soit $\mathcal{P}$ un problème d'optimisation tel que $\mathcal{P}_d$ est **NP-complet**. On a alors que $\mathcal{P}$ est **NP-Difficile**.

Enfin, on va donner une dernière variante d'un problème d'optimisation $\mathcal{P}$, la version compression que l'on va noter $\mathcal{P}_c$.

— Si $\mathcal{P}$ est un problème de maximisation, $\mathcal{P}_c$ est le problème qui va chercher pour une entrée $\mathcal{I}$ de $\mathcal{P}$ à trouver une solution $\mathcal{O}$ de $\mathcal{P}$ qui minimise la différence entre la taille de $\mathcal{I}$ et la taille de $\mathcal{O}$.

— Si $\mathcal{P}$ est un problème de minimisation, $\mathcal{P}_c$ est le problème qui va chercher pour une entrée $\mathcal{I}$ de $\mathcal{P}$ à trouver une solution $\mathcal{O}$ de $\mathcal{P}$ qui maximise la différence entre la taille de $\mathcal{I}$ et la taille de $\mathcal{O}$.

On a alors que les problèmes $\mathcal{P}$ et $\mathcal{P}_c$ ont la même complexité. En effet, l'ensemble des solutions optimales de $\mathcal{P}$ est exactement l'ensemble de solutions optimales de $\mathcal{P}_c$. Cette variante va être surtout appliquée sur des problèmes de superchaînes dans cette thèse (voir chapitre 3) mais peut être appliquée sur tout problème d'optimisation.

Remarque 1.22 Même s'il est aussi difficile de résoudre $\mathcal{P}$ que $\mathcal{P}_c$, en terme d'approximation, les deux problèmes ne sont pas forcément comparables de manière immédiate (voir section 3.2).

Chapitre 2

Problèmes de superchaînes

Dans ce chapitre, on va s'intéresser à un problème phare de l'algorithmique du texte, le problème de la plus petite superchaîne. Ce problème a un long historique : on trouve déjà des traces de ce problème dans un article de Shapiro d'octobre 1967 [81] et ce problème est encore très étudié : Paluch donne en 2014 [67] un algorithme d'approximation pour ce problème avec un meilleur ratio d'approximation optimal par rapport à ceux existant [11, 88, 18, 3, 4, 5, 14, 86, 44, 68, 63]. Le problème de la plus petite superchaîne a plusieurs applications, notamment en compression [83, 85, 84, 37], en assemblage [82, 71, 52, 53, 75, 94, 21] et en ordonnancement [98]. On va de plus s'intéresser et présenter ici des variantes naturelles du problème de la plus petite superchaîne : des problèmes qui ont été peu étudiée comme par exemple le problème de la plus petite couverture circulaire de mots et des problèmes pas du tout étudiée comme par exemple le problème de la plus petite superchaîne circulaire. L'apport de ce chapitre n'est pas dans les propositions données, car elles sont souvent connues et admises mais dans leurs formalismes et leurs preuves. En effet, par exemple la restriction à des solutions factor-free ou à des solutions issues d'une permutation n'est pas forcément expliquée dans la littérature et peut entraîner à des confusions et à des erreurs.

Sommaire

2.1 Superchaîne linéaire

Dans cette première section, on va définir le problème de la plus petite superchaîne et on va replacer ce problème dans l'état de l'art actuel. On va de plus reprendre et expliquer les travaux de Ukkonen [91] et de Vassilevska [93] qui combinés permettent de redéfinir le problème classique en limitant l'ensemble des solutions envisageables. Ce problème est un bon problème pour définir l'assemblage de génomes [29, 72, 47].

2.1.1 Définition du problème SLS

Soit P un ensemble de mots. Une *superchaîne* de P est un mot w tel que tout mot s de P est une sous-chaîne de w, *i.e.* $s \underset{\text{sub}}{\subset} w$. Une superchaîne de P peut être aussi appelée une *superchaîne linéaire* de P. On fait cette distinction car on verra par la suite des superchaînes circulaires.

On peut alors définir le problème classique de la plus petite superchaîne linéaire. Pour garder le nom que ce problème a dans la littérature, on va l'appeler Shortest Linear Superstring (SLS). Il peut aussi être trouvé sous le nom de Shortest Superstring Problem (SSP) ou Shortest Common Superstring (SCS), mais on gardera le nom et la définition suivante pour ce problème.

Problèmme 2.1 ***Shortest Linear Superstring*** *(SLS) (voir Figure 2.1)* Soit P un ensemble de mots, on cherche une superchaîne linéaire de P qui soit de longueur minimale.

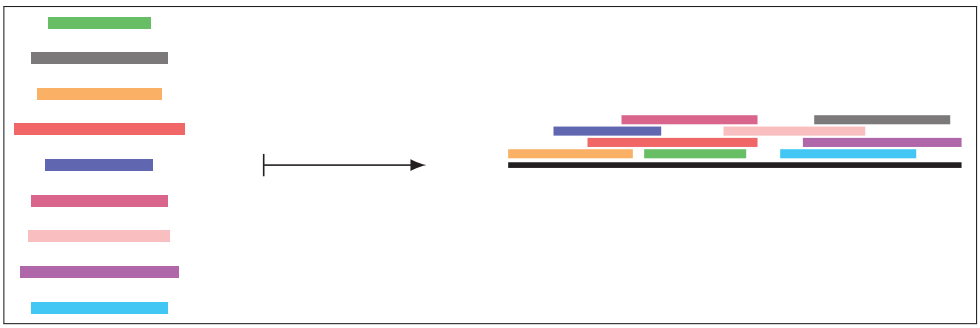


FIGURE 2.1 – Schéma du problème SLS. Les rectangles de couleurs représentent les mots de l'instance et le rectangle noir correspond à une superchaîne de l'instance.

Exemple 2.2 Soit $P = \{ab, bc, db\}$ un ensemble de mots. *bcabdb*, *abcdb* et *abdbc* sont des exemples de superchaînes de P.

b	c	a	b	d	b
b	c			d	b
		a	b		

a	b	c	d	b
a	b		d	b
	b	c		

a	b	d	b	c
		d	b	
a	b		b	c

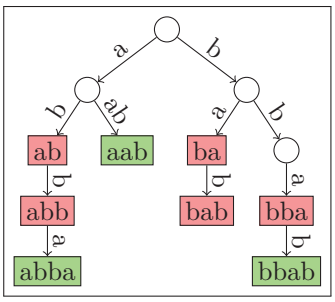
De plus, *abcdb* et *abdbc* sont toutes les deux optimales pour le problème SLS, c'est-à-dire qu'on ne peut pas trouver de superchaînes de P strictement plus petites.

Comme on peut le voir dans Exemple 2.2, il peut exister plusieurs solutions optimales au problème SLS. Il en sera de même pour les autres problèmes liés aux superchaînes. On parlera alors d'une des plus petites superchaînes d'une instance et non de la plus petite superchaîne.

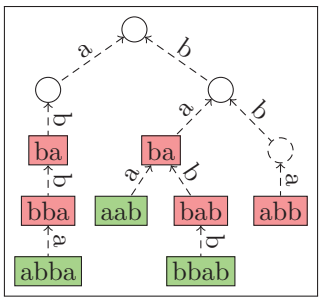
On dit qu'un ensemble de mots P est *factor-free* si aucun mot de P n'est sous-chaîne d'un autre mot de P. Soit P un ensemble de mots. On peut alors rendre cet ensemble factor-free en supprimant les mots les plus petits : on pose $Factor\text{-}free(P)$, le sous-ensemble P' de P tel que P' soit factor-free et s'il existe x et y deux mots de P tels que $x \underset{\text{sub}}{\subset} y$ alors x n'appartient pas à P'.

À partir d'un ensemble P, construire l'ensemble $Factor\text{-}free(P)$ peut être fait en un temps linéaire en $||P||$. On peut retrouver ce résultat dans Ukkonen [91]. L'idée de l'algorithme est d'utiliser l'arbre des suffixes et l'arbre des liens suffixes de P (voir Chapitre 6.2). On a alors que $Factor\text{-}free(P)$ est exactement l'intersection entre l'ensemble des feuilles de l'arbre des suffixes de P et l'ensemble des feuilles dans l'arbre des liens suffixes de P (voir Figure 2.2).

Exemple 2.3 Soit $P := \{abba, bba, ba, ab, aab, bbab, abb, bab\}$. On a alors $Factor\text{-}free(P)$
$= \{abba, aab, bbab\}$. En effet on a : $bba \underset{\text{sub}}{\subset} abba$, $ba \underset{\text{sub}}{\subset} abba$, $ab \underset{\text{sub}}{\subset} aab$, $bab \underset{\text{sub}}{\subset} bbab$ et $abb \underset{\text{sub}}{\subset} abba$.



(a) Arbre des suffixes de P



(b) Arbre des liens suffixes de P

FIGURE 2.2 – Arbre des suffixes et arbre des liens suffixes pour l'ensemble de mots $P = \{abba, bba, ba, ab, aab, bbab, abb, bab\}$. Les nœuds en vert sont les nœuds de $Factor\text{-}free(P)$ et les nœuds en rouge sont les nœuds de $P \setminus Factor\text{-}free(P)$. En effet, les seuls mots de P qui sont des feuilles dans l'arbre des suffixes et dans l'arbre des liens suffixes de P sont *abba*, *aab* et *bbab*.

On a de plus une équivalence entre les solutions optimales pour SLS de P et de $Factor\text{-}free(P)$:

Proposition 2.4 Soit w un mot. On a alors que w est une superchaîne de P si et seulement si w est une superchaîne de $Factor\text{-}free(P)$.

Preuve Soient x et y deux mots de P tels que $x \underset{\text{sub}}{\subset} y$, si y est sous-chaîne d'un mot w, *i.e.* $y \underset{\text{sub}}{\subset} w$. Comme $\underset{\text{sub}}{\subset}$ est une relation d'ordre, on a que $x \underset{\text{sub}}{\subset} w$. ■

En appliquant la proposition 2.4, on a alors qu'une des plus petites superchaînes de P et aussi une des plus petites superchaînes de $Factor\text{-}free(P)$ et inversement, *i.e.* $OPT_{\mathsf{SLS}}(P) = OPT_{\mathsf{SLS}}(Factor\text{-}free(P))$. Comme on peut construire linéairement en $||P||$ l'ensemble $Factor\text{-}free(P)$, dans la suite de cette section, on va prendre directement un ensemble de mots $P = \{s_1, \ldots, s_n\}$ qui est factor-free.

En général, par la suite quand on parlera du problème SLS, on prendra comme entrée une ensemble P qui sera factor-free. En effet si on trouve un algorithme au moins linéaire pour SLS pour un ensemble factor-free, on pourra faire de même pour un ensemble qui n'est pas factor-free. Ce qu'on peut expliciter formellement de la manière suivante :

Proposition 2.5 Soit $\mathcal{A}$ un algorithme pour SLS avec un ratio d'approximation de α qui est en temps $O(f(P))$ avec $f(P) \geq ||P||$. Soit un ensemble de mots P' qui n'est pas factor-free, en construisant $Factor\text{-}free(P')$ et en appliquant $\mathcal{A}$ sur $Factor\text{-}free(P')$, on obtient un algorithme $\mathcal{A}'$ avec un ratio de α qui est en temps $O(f(P'))$.

Remarque 2.6 Il existe des algorithmes parallèles qui sont polylogarithmiques sur un nombre polynomial de machines en parallèle [18]. Pour ces algorithmes, on ne peut pas appliquer la proposition 2.5 car $Log^n(||P||) < ||P||$.

2.1.2 Superchaîne générée par une permutation

Soient x et y deux mots linéaires. On pose $Occ(x, y) = \{i \in \{1, \ldots, |y| - |x| + 1\} \mid y[i, i + |x| - 1] = x\}$, *i.e.* l'ensemble des positions des occurrences de x dans y. Il est facile de voir que si $|y| < |x|$, alors $\{1, \ldots, |y| - |x| + 1\}$ est forcément vide et donc $Occ(x, y) = \emptyset$. Soient w une superchaîne de P et s_i et s_j deux mots différents de P. Comme P est un ensemble factor-free, on a que $Occ(s_i, w) \cap Occ(s_j, w) = \emptyset$. En effet, dans une superchaîne w, chaque mot de P apparaît à une position différente, sinon le plus petit serait un préfixe du plus grand, ce qui est impossible car P est factor-free.

Remarque 2.7 En regardant d'un peu plus près, pour une position k_i de s_i dans w, s_j ne peut pas apparaître dans w à la position k_i ni entre $k_i + 1$ et $k_i + |pr(s_i, s_j)| - 1$. On peut résumer cela par :

$$Occ(s_j, w) \bigcap \Big(\bigcup_{k_i \in Occ(s_i, w)} \{k_i, \ldots, k_i + |pr(s_i, s_j)| - 1\} \Big) = \emptyset$$

On a de plus que $pr(s_i, s_j) \neq \varepsilon$ et donc $|pr(s_i, s_j)| \geq 1$.

Exemple 2.8 Soient $w = ababaababbaa$, $s_1 = bab$ et $s_2 = baa$. On a alors $Occ(s_1, w) = \{2, 7\}$ et $Occ(s_2, w) = \{4, 10\}$. D'après la remarque 2.7, on a bien que

$$\begin{aligned} Occ(s_1, w) \bigcap \Big(\bigcup_{k_1 \in Occ(s_1, w)} \{k_1, \ldots, k_1 + |pr(s_1, s_2)| - 1\} \Big) &= \{4, 10\} \bigcap \{2, 3, 7, 8\} \\ &= \emptyset \end{aligned}$$

On va maintenant utiliser une permutation pour créer une superchaîne d'un ensemble de mots. Soient P un ensemble de mots factor-free de cardinal n ($n = |P|$) et σ une permutation de $\{1, \ldots, n\}$. Comme P est factor-free, tous les mots de P sont différents les uns des autres, et on peut donc en utilisant l'ordre lexicographique classer les éléments de $P = \{s_1, \ldots, s_n\}$ de telle sorte que pour tout i entre 1 et $n-1$, $s_i \preceq s_{i+1}$.

Remarque 2.9 Le fait d'ajouter un ordre, permet de donner une numérotation unique des mots de P.

On a alors la définition suivante :

Définition 2.10 La *superchaîne générée* par la permutation σ (voir Figure 2.3), notée $P_l(\sigma)$, est telle que :

$$P_l(\sigma) = pr(s_{\sigma(1)}, s_{\sigma(2)}).\ldots.pr(s_{\sigma(n-1)}, s_{\sigma(n)}).s_{\sigma(n)}.$$

On a alors que $P_l(\sigma)$ correspond à la superchaîne où apparaît en premier lieu une occurrence de $s_{\sigma(1)}$ puis une occurrence de $s_{\sigma(2)}$ et ainsi de suite jusqu'à $s_{\sigma(n)}$. Pour construire cette superchaîne, on commence par prendre le plus petit préfixe de $s_{\sigma(1)}$ que l'on ne retrouve pas dans $s_{\sigma(2)}$, ce qui est exactement $pr(s_{\sigma(1)}, s_{\sigma(2)})$. On réitère en concaténant à la fin les $pr(s_{\sigma(i)}, s_{\sigma(i+1)})$ jusqu'à arriver à $s_{\sigma(n)}$. Enfin, on concatène à la fin $s_{\sigma(n)}$ pour que $s_{\sigma(n)}$ soit une sous-chaîne de $P_l(\sigma)$.

Remarque 2.11 La définition que l'on donne de $P_l(\sigma)$ n'est pas unique, en effet on peut définir aussi $P_l(\sigma)$ comme :

$$P_l(\sigma) = s_{\sigma(1)}.su(s_{\sigma(1)}, s_{\sigma(2)}).\ldots.su(s_{\sigma(n-1)}, s_{\sigma(n)}).$$

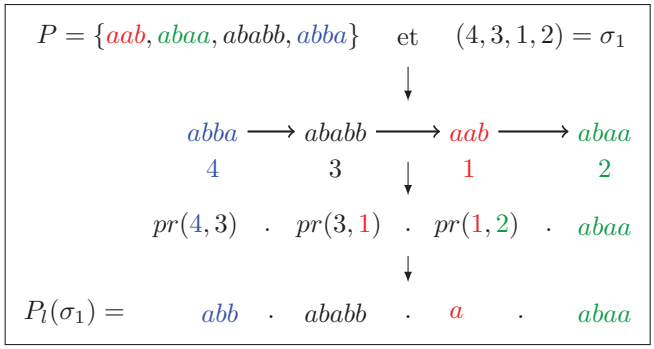


FIGURE 2.3 – Exemple de superchaîne générée par la permutation $\sigma_1 = (4, 3, 1, 2)$ sur l'ensemble de mots $P = \{aab, abaa, ababb, abba\}$. On a alors $P_l(\sigma_1) = abbababbaabaa$.

On note alors $PSLS(P)$ l'ensemble des superchaînes générées par des permutations de $\{1, \ldots, n\}$, *i.e.* $PSLS(P) = \{P_l(\sigma) \mid \sigma$ une permutation sur $\{1, \ldots, n\}\}$. On a alors une propriété fondamentale que chaque solution optimale de SLS est une superchaîne générée par une permutation, ce qui, traduit, donne la proposition suivante et la Figure 2.4 :

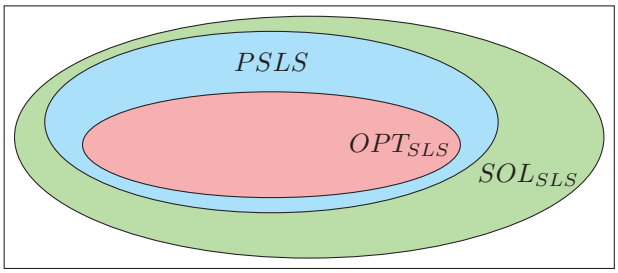


FIGURE 2.4 – Diagramme d'Euler du lien entre $OPT_{SLS}(P)$, $PSLS(P)$ et $SOL_{SLS}(P)$ et cela pour tout ensemble de mots P.

Proposition 2.12 Soit P un ensemble factor-free. $OPT_{\mathsf{SLS}}(P) \subseteq PSLS(P)$.

Donc, pour chaque solution optimale w_{opt} de SLS de P, il existe une permutation σ de $\{1, \ldots, n\}$ telle que $P_l(\sigma) = w_{opt}$. Mais attention, comme dans l'exemple 2.13, cette permutation n'est pas forcément unique, en effet il peut exister deux permutations qui génèrent la même solution optimale. De plus, toute superchaîne générée par une permutation n'est pas forcément optimale, *i.e.* $PSLS(P) \neq OPT_{\mathsf{SLS}}(P)$ (voir Exemple 2.13).

Exemple 2.13 Soient $P = \{aba, bad, baf, cab, eab\}$, $\sigma_1 = (4, 1, 2, 5, 3)$, $\sigma_2 = (4, 2, 5, 1, 3)$ et $\sigma_3 = (1, 2, 3, 4, 5)$. On a alors $P_l(\sigma_1) = c.a.bad.ea.baf = cabadeabaf \in OPT_{SLS}(P)$, $P_l(\sigma_2) = ca.bad.e.a.baf = cabadeabaf \in OPT_{SLS}(P)$ et $P_l(\sigma_3) = a.bad.baf.cab.eab = abadbafcabeab \notin OPT_{SLS}(P)$.

On peut même en dire un peu plus que la proposition 2.12. Soit w une superchaîne de P. On va décomposer w en mot de P. Soient $i = (i_1, \ldots, i_n)$ un n-uplet tel que pour tout $j \in \{1, \ldots, n\}$, $i_j \in Occ(s_j, w)$ et σ_i la permutation de $\{1, \ldots, n\}$ telle que pour tout j, k de $\{1, \ldots, n\}$, $\sigma_i(j) < \sigma_i(k)$ si et seulement si $i_j < i_k$. On a alors la proposition suivante :

Proposition 2.14 $|P_l(\sigma_i)| \leq |w|$.

La proposition 2.14 nous dit alors que $P_l(\sigma)$ est la plus petite superchaine correspondant à l'ordre des mots σ. En effet, pour chaque superchaîne w de P, il existe au moins un élément w_i de $PSLS(P)$ tel que les mots apparaissent dans le même ordre dans w que dans w_i, et de plus on a que w_i est plus petit en taille que w. Comme pour chaque n-ulpet, on a une unique permutation σ_i, le nombre d'éléments w_i de $PSLS(P)$ tels que les mots apparaissent dans le même ordre dans w que dans w_i est $\Pi_{j \in \{1,\ldots,n\}} |Occ(s_j, w)|$.

Grâce à la proposition 2.12, on peut alors redéfinir le problème SLS :

Problèmme 2.15 ***Shortest Linear Superstring (SLS) version 2*** Soit P un ensemble de mots factor-free. On cherche une superchaîne linéaire de P de longueur minimale et générée par une permutation de $\{1, \ldots, |P|\}$.

En effet, cette nouvelle définition du problème donne un problème équivalent au problème classique parce que $OPT_{\mathsf{SLS}}(P) \cap PSLS(P) = OPT_{\mathsf{SLS}}(P)$.

Remarque 2.16 Hormis Vassilevska [93] qui explicite le problème 2.15, beaucoup d'articles explicitent le problème 2.1 mais utilisent en réalité le problème 2.15. Or pour utiliser la version du problème 2.15, on a besoin d'avoir en entrée un ensemble de mots factor-free, ce qui n'est pas forcément expliqué. De plus, de par la définition du problème 2.15, on a que le nombre de superchaînes à envisager est fini ce qui n'est pas vrai dans la définition classique du problème SLS (problème 2.1).

Grâce à cette nouvelle définition du problème SLS, on a quelques propriétés simples :

Proposition 2.17 Soit P un ensemble de mots factor-free de cardinal n.

1. $1 \leq |OPT_{SLS}(P)| \leq n!$
2. Pour $w_{opt} \in OPT_{SLS}(P)$, $|P| + max(\{|w| \mid w \in P\}) - 1 \leq |w_{opt}| \leq ||P||$.

Le 2. de la proposition 2.17 vient du fait que dans w_{opt} chaque s_i apporte au moins une lettre et que w_{opt} contient en outre au moins le mot le plus long de P. Comme on peut le voir sur l'exemple 2.18, les bornes données par la proposition 2.17 sont atteintes pour certains ensembles de mots (dans certains cas les bornes peuvent même se rejoindre).

Exemple 2.18 Soit $\Sigma = \{a_1, \ldots, a_\ell\}$ un alphabet.

— Soit $P_1 = \{a_1a_2, a_2a_3, \ldots, a_{\ell-1}a_\ell\}$. Alors $OPT_{SLS}(P_1) = \{a_1a_2 \ldots a_\ell\}$ et $|a_1a_2 \ldots a_\ell| = \ell = |P_1| + max(\{|w| \mid w \in P_1\}) - 1$.

— Soit $P_2 = \{a_1, \ldots, a_\ell\}$. Alors $|OPT_{SLS}(P_2)| = \ell!$ et $|P_2| + max(\{|w| \mid w \in P_2\}) - 1 = ||P_2|| = \ell$.

2.1.3 Etat de l'art

Dans cette sous-section, on va parler un peu plus en détail du problème SLS dans la littérature. On pourra se référer à [27] pour plus de détails. On va commencer par parler de la complexité du problème et ensuite des algorithmes d'approximation pour SLS.

Le problème SLS est associé au problème de décision SLS_d :

Problèmme 2.19 SLS_d Soient P un ensemble de mots et k un entier. Existe-t-il une superchaîne de P qui a une taille inférieure ou égale à k ?

Le problème SLS_d a été montré comme **NP-complet** par Maier et al. [57]. Cette première preuve a été faite à l'aide d'une réduction polynomiale à partir du problème de couverture par sommets. Maier et al. montrent que le problème SLS_d est **NP-complet** même si les mots sont de longueurs 8 ou plus. Il y eut ensuite d'autres preuves de la NP-complétude du problème SLS_d pour des restrictions d'ensemble de mots plus petits ou sur un alphabet restreint [25, 89, 60, 61, 10]. On a de plus que le problème SLS_d est difficile à approximer (**APX-Difficile**) [65] et [11] grâce à [6].

Le problème SLS_d est équivalent à trouver la taille de la plus petite superchaîne de P, en effet si on connaît sa taille, pour tous les k plus grands ou égaux à cette taille, SLS_d donnera une réponse positive et dans les autres cas, une réponse négative. De l'autre coté, en trouvant un k tel que SLS_d donne une réponse positive pour k et une réponse négative pour $k-1$, k est alors la taille de la plus petite superchaîne de P. Comme en trouvant une des plus petites superchaînes pour SLS, on peut calculer sa taille linéairement en la

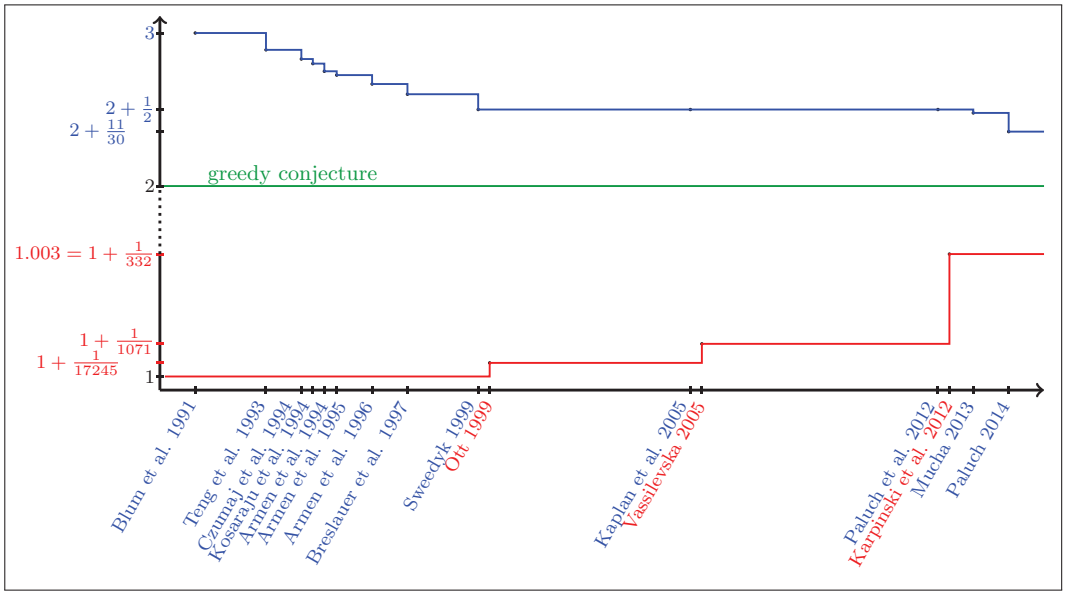


FIGURE 2.5 – Frise chronologique des bornes d'approximations constantes (en bleu), d'inapproximations constantes (en rouge) et de la conjecture du greedy (en vert) de SLS.

taille de l'instance le k qui résout SLS_d, on connaît donc aussi la taille de la plus petite superchaîne de P. On a donc que SLS est aussi difficile que SLS_d.

Le problème SLS est très étudié et depuis les années 1990, régulièrement une nouvelle borne d'approximation pour SLS est trouvée (voir Figure 2.5). Actuellement la meilleure a été donnée par Paluch et est de $2 + \frac{11}{30}$ [67]. L'algorithme utilisé par Paluch peut être décrit de la façon suivante :

1. Soit P un ensemble de mots.
2. On calcule une couverture circulaire de mots C de P.
3. À partir de C, on calcule C' une couverture circulaire de mots canonique (chaque composante a au moins deux mots comme sous-chaîne) de P.
4. On prend P' un ensemble de mots linéaires obtenus par la linéarisation des mots circulaires de C'.
5. On utilise un autre algorithme $\mathcal{A}$ sur SLS_c qui a un ratio d'approximation optimal constant sur l'ensemble de mots linéaires P'.

Paluch donne dans [67] un algorithme $\mathcal{A}$ avec un ratio d'approximation optimal de $\frac{3}{4}$ sur SLS_c, ce qui entraîne un ratio d'approximation optimal sur SLS pour l'algorithme général de $2+\frac{11}{30}$. La majorité des algorithmes avec des ratios d'approximation constants étudiés sur SLS sont des applications de cet algorithme général [11, 88, 18, 3, 4, 5, 14, 86, 44, 68, 63].

De même, on a des bornes inférieures pour les ratios d'approximations possibles [65, 93, 46].

Parmi les algorithmes connus sur SLS, on va s'intéresser à un autre algorithme : l'algorithme glouton. Même si actuellement le meilleur ratio d'approximation connu est de $3 + \frac{1}{2}$ [44], le ratio d'approximation est conjecturé à 2 [84, 87, 11]. On parlera plus en détail de l'algorithme glouton dans la section 4.1.

2.2 Superchaîne circulaire

Après s'être intéressé au cas du problème de la plus petite superchaîne, on va examiner le problème de la plus petite superchaîne circulaire. On ne trouve pas de référence à ce problème dans la littérature même s'il a aussi des applications pratiques en ordonnancement, en compression et en assemblage. En effet, on sait que chez certains eucaryotes, l'ADN mitochondrial et l'ADN chloroplastique sont circulaires. On retrouve chez certains virus et bactéries aussi des génomes circulaires. On va faire ici un point sur ce que l'on peut dire sur ce problème en utilisant et en extrapolant les résultats sur le problème de la plus petite superchaîne.

2.2.1 Définition du problème SCS

Une *superchaîne circulaire* de P est un mot circulaire $\langle w \rangle$ tel que tout mot s de P est une sous-chaîne de $\langle w \rangle$, *i.e.* $s \underset{\text{sub}}{\subset} \langle w \rangle$. On peut alors définir le problème de la plus courte superchaîne circulaire.

Problèmme 2.20 ***Shortest Cyclic Superstring*** *(*SCS*) (voir Figure 2.6)* Soit P un ensemble de mots. On cherche une superchaîne circulaire de P de longueur minimale.

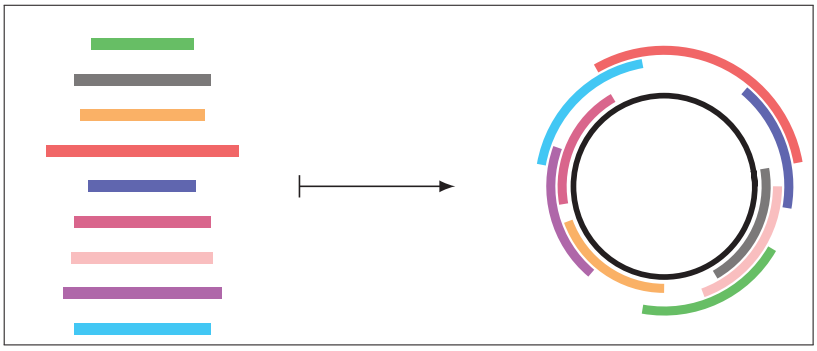


FIGURE 2.6 – Schéma du problème SCS. Les rectangles de couleur représentent les mots de l'instance et le rectangle noir correspond à une superchaîne circulaire de l'instance.

Pour commencer, les propositions 2.4 et 2.5 peuvent être généralisées à SCS : on prendra alors pour la suite de cette section un ensemble $P = \{s_1, \ldots, s_n\}$ de mots qui est factor-free.

De plus, on a vu dans la section 2.1 la définition d'une superchaîne linéaire générée par une permutation. Nous verrons ici une autre façon de créer un superchaîne à l'aide d'une permutation : on appellera ce type de superchaîne, une superchaîne issue d'une permutation.

Remarque 2.21 On fait ici la différence entre une superchaîne générée par une permutation et une superchaîne issue d'une permutation. Dans le cas de la superchaîne générée par une permutation, la permutation nous dit dans quel ordre on va voir les mots dans la superchaîne. Dans le cas de la superchaîne issue d'une permutation, la permutation nous dira pour un mot, quel est le mot que l'on verra juste après dans la superchaîne.

Soit $P = \{s_1, \dots, s_n\}$ un ensemble de mots tels que pour tout i entre 1 et $n-1$, $s_i \preceq s_{i+1}$ et σ_c une permutation circulaire de $\{1, \dots, n\}$. On a alors la définition suivante :

Définition 2.22 ***voir Figure 2.7*** La *superchaîne circulaire de P issue* de la permutation circulaire σ_c, notée $Circular(P, \sigma_c)$, est telle que :

$$Circular(P, \sigma_c) = \langle pr(s_1, s_{\sigma_c(1)}).pr(s_{\sigma_c^1(1)}, s_{\sigma_c^2(1)}).\dots.pr(s_{\sigma_c^{n-1}(1)}, s_1)\rangle$$

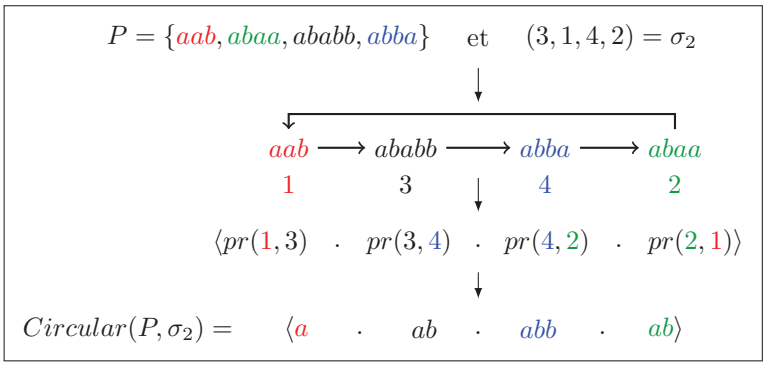


FIGURE 2.7 – Exemple de superchaîne circulaire issue de la permutation circulaire $\sigma_2 = (3, 1, 4, 2) = (1 \to 3 \to 4 \to 2)$ sur $P = \{aab, abaa, ababb, abba\}$. On a alors $Circular(P, \sigma_2) = \langle aababbab\rangle$.

Remarque 2.23 On peut alors faire plusieurs remarques sur la définition du mot circulaire $Circular(P, \sigma_c)$.

— Pour $n = 1$, on a alors que $Circular(P, \sigma_c) = \langle pr(s_1, s_1)\rangle$.

— On a fait ici le choix de débuter par s_1 mais ce choix est arbitraire, on aurait exactement la même définition en prenant i entre 1 et n et en posant

$$Circular(P, \sigma_c) = \langle pr(s_i, s_{\sigma_c(i)}).pr(s_{\sigma_c^1(i)}, s_{\sigma_c^2(i)}).\dots.pr(s_{\sigma_c^{n-1}(i)}, s_i)\rangle.$$

— Au lieu d'utiliser l'ordre lexicographique sur Σ^* et une permutation circulaire de $\{1, \dots, n\}$, on aurait pu prendre une permutation circulaire ρ_c de P et définir la superchaîne circulaire suivante :

$$Circular'(P, \rho_c) = \langle pr(s_1, \rho_c(s_1)).pr(\rho_c(s_1), \rho_c^2(s_1)).\dots.pr(\rho_c^{n-1}(s_1), s_1)\rangle.$$

Il est facile de voir que les deux définitions sont similaires et que les ensembles de superchaînes issues de ces deux définitions sont égales (voir Exemple 2.24).

Exemple 2.24 Soit $P = \{s_1, s_2, s_3, s_4\}$.

— Soit σ_c la permutation circulaire de $\{1, \dots, 4\}$ telle que :

$$\begin{array}{rccc} \sigma_c : & \{1,\dots,4\} & \to & \{1,\dots,4\} \\ & 1 & \mapsto & 2 \\ & 2 & \mapsto & 4 \\ & 3 & \mapsto & 1 \\ & 4 & \mapsto & 3 \end{array}$$

On a alors que :

$$\begin{aligned} Circular(P, \sigma_c) &= \langle pr(s_i, s_{\sigma_c(i)}).pr(s_{\sigma_c^1(i)}, s_{\sigma_c^2(i)}).\dots.pr(s_{\sigma_c^3(i)}, s_i)\rangle \\ &= \langle pr(s_1, s_2).pr(s_2, s_4).pr(s_4, s_3).pr(s_3, s_1)\rangle \end{aligned}$$

— Soit ρ_c la permutation circulaire de P telle que :

$$\begin{array}{rccc} \rho_c : & P & \to & P \\ & s_1 & \mapsto & s_2 \\ & s_2 & \mapsto & s_4 \\ & s_3 & \mapsto & s_1 \\ & s_4 & \mapsto & s_3 \end{array}$$

On a alors que :

$$\begin{aligned} Circular'(P, \rho_c) &= \langle pr(s_1, \rho_c(s_1)).pr(\rho_c(s_1), \rho_c^2(s_1)).\dots.pr(\rho_c^3(s_1), s_1)\rangle \\ &= \langle pr(s_1, s_2).pr(s_2, s_4).pr(s_4, s_3).pr(s_3, s_1)\rangle \end{aligned}$$

Comme pour le problème SLS, on pose $PSCS(P)$ l'ensemble des superchaînes circulaires issues des permutations circulaires de $\{1, \dots, n\}$, *i.e.* $PSCS(P) = \{Circular(P, \sigma_c) \mid \sigma_c$ une permutation circulaire de $\{1, \dots, n\}\}$. On peut alors donner le pendant de la proposition 2.12 et le pendant de la figure 2.8.

Proposition 2.25 Soit P un ensemble factor-free. On a $OPT_{\mathsf{SCS}}(P) \subseteq PSCS(P)$.

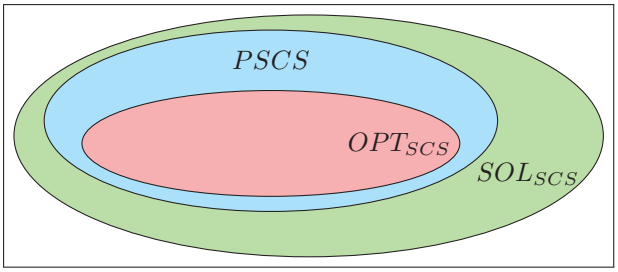


FIGURE 2.8 – Diagramme d'Euler reliant les ensembles $OPT_{\mathsf{SCS}}(P)$, $PSCS(P)$ et $SOL_{\mathsf{SCS}}(P)$ pour tout ensemble de mots P.

On peut alors, comme on l'a fait avec le problème SLS, redéfinir le problème SCS :

Problèmme 2.26 ***Shortest Cyclic Superstring (SCS) version 2*** Soit P un ensemble de mots. On cherche une superchaîne circulaire de P issue d'une permutation de $\{1, \dots, |P|\}$ qui soit de longueur minimale.

2.2.2 Liens entre les solutions optimales de *SLS* et *SCS*

Soit σ_c une permutation circulaire de $\{1, \dots, n\}$. On peut alors définir la superchaîne linéaire de P associée à un décalé circulaire de $Circular(P, \sigma_c)$. Pour i entre 1 et n :

$$Linear(P, \sigma_c, i) = pr(s_{\sigma_c(i)}, s_{\sigma_c^2(i)}).pr(s_{\sigma_c^2(i)}, s_{\sigma_c^3(i)}).\dots.pr(s_{\sigma_c^{n-1}(i)}, s_i).s_i$$

Il est facile de voir que $Linear(P, \sigma_c, i)$ est bien une superchaîne linéaire. En outre, pour P qui n'est pas un singleton (*i.e.* $|P| \geq 2$) :

$$PSLS(P) = \{Linear(P, \sigma_c, i) \mid \sigma_c \text{ une permutation circulaire de } \{1, \dots, n\} \text{ et } i \in \{1, \dots, n\}\}.$$

En effet, en posant σ_c^* la permutation de $\{1, \dots, n\}$ telle que pour tout $j \in \{1, \dots, n-1\}$, $\sigma_c^*(j) := \sigma_c^{j+1}(i)$ et $\sigma_c^*(n) = i$, on a $Linear(P, \sigma_c, i) = P_l(\sigma_c^*)$.

Comme pour tout i entre 1 et n, on a $|pr(s_i, s_{\sigma_c(i)})| \leq |s_i|$, on a donc l'inégalité $|Circular(P, \sigma_c)| \leq |Linear(P, \sigma_c, i)|$ et donc pour $\langle w_{opt,c} \rangle \in OPT_{\mathsf{SCS}}(P)$ et $w_{opt,l} \in OPT_{\mathsf{SLS}}(P)$, on a

$$|\langle w_{opt,c} \rangle| \leq |w_{opt,l}|.$$

Malheureusement, on ne peut pas trouver un réel α tel que pour tout ensemble de mots P et pour tout $\langle w_{opt,c} \rangle \in OPT_{\mathsf{SCS}}(P)$ et $w_{opt,l} \in OPT_{\mathsf{SLS}}(P)$, $|w_{opt,l}| \leq \alpha \times |\langle w_{opt,c} \rangle|$ (voir Exemple 2.27).

Exemple 2.27 Soient n un entier et $P = \{(ab)^n, (ba)^n\}$. On a alors que $OPT_{\mathsf{SLS}}(P) = \{(ab)^n a, (ba)^n b\}$ et $OPT_{\mathsf{SCS}}(P) = \{\langle aba \rangle, \langle bab \rangle\}$.

On ne peut pas encadrer directement une solution de $OPT_{\mathsf{SLS}}(P)$ avec une solution de $OPT_{\mathsf{SCS}}(P)$, mais on peut transformer une solution de $OPT_{\mathsf{SCS}}(P)$ pour la comparer à $OPT_{\mathsf{SLS}}(P)$. Soit σ_c une permutation circulaire telle que $Circular(P, \sigma_c) \in OPT_{\mathsf{SCS}}(P)$. On a alors que pour tout $i \in \{1, \dots, n\}$ et pour tout $w_{opt,l} \in OPT_{\mathsf{SLS}}(P)$, $|w_{opt,l}| \leq |Linear(P, \sigma_c, i)|$. On a alors la proposition suivante :

Proposition 2.28 Soient σ_c une permutation circulaire de $\{1, \dots, |P|\}$ telle que le mot circulaire $Circular(P, \sigma_c)$ est dans $OPT_{\mathsf{SCS}}(P)$ et σ_l une permutation circulaire telle qu'il existe $j \in \{1, \dots, n\}$ tel que $Linear(P, \sigma_l, j) \in OPT_{\mathsf{SLS}}(P)$. Pour tout $i \in \{1, \dots, n\}$:

$$|Circular(P, \sigma_c)| \leq |Circular(P, \sigma_l)| \leq |Linear(P, \sigma_l, j)| \leq |Linear(P, \sigma_c, i)|$$

Exemple 2.29 Soit $P = \{abcdfa, abcfab, fababc\}$. On a alors que $OPT_{\mathsf{SLS}}(P) = \{abcfababcdfa\}$ et $OPT_{\mathsf{SCS}}(P) = \{\langle abcfababcdf \rangle, \langle abcdfababcf \rangle\}$. Comme $abcfababcdfa = Linear(P, (1, 2, 3), 2)$, $\langle abcfababcdf \rangle = Circular(P, (1, 2, 3))$ et $\langle abcdfababcf \rangle = Circular(P, (1, 3, 2))$, on a :

$$Circular(P, (1, 3, 2)) \leq Circular(P, (1, 2, 3)) \leq Linear(P, (1, 2, 3), 2) \leq Linear(P, (1, 3, 2), 2)$$

car

$$\begin{array}{ccccccc} |\langle abcdfababcf \rangle| & \leq & |\langle abcfababcdf \rangle| & \leq & |abcfababcdfa| & \leq & |abcdfababcfab| \\ 11 & \leq & 11 & \leq & 12 & \leq & 13 \end{array}$$

`Q : Peut-on trouver une instance de mots` P `telle que les inégalités de la proposition 2.28 soient toutes strictes ?`

On a alors le corollaire suivant de la proposition 2.28 qui nous donne une borne sur la transformation linéaire d'une solution optimale de $OPT_{\mathsf{SCS}}(P)$.

Corollaire 2.30 Soient σ_c une permutation circulaire telle que $Circular(P, \sigma_c)$ est dans $OPT_{\mathsf{SCS}}(P)$ et σ_l une permutation circulaire telle qu'il existe $j \in \{1, \ldots, n\}$ tel que $Linear(P, \sigma_l, j) \in OPT_{\mathsf{SLS}}(P)$. Pour tout $i \in \{1, \ldots, n\}$:

$$|Linear(P, \sigma_c, i)| \leq 2 \times |Linear(P, \sigma_l, j)|$$

Preuve En effet, comme $|Circular(P, \sigma_c)| \leq |Linear(P, \sigma_l, j)|$ et on a $|su(s_j, s_{\sigma_l(j)})| \leq min(|s_j|, |s_{\sigma_l(j)}|) \leq |Linear(P, \sigma_l, j)|$

$$\begin{array}{rcl} |Linear(P, \sigma_l, j)| & = & |Circular(P, \sigma_c)| + |su(s_j, s_{\sigma_l(j)})| \\ & \leq & |Linear(P, \sigma_l, j)| + |Linear(P, \sigma_l, j)| \\ & = & 2 \times |Linear(P, \sigma_l, j)|. \end{array}$$

2.2.3 Complexité de SCS

Le problème SCS est associé au problème de décision SCS_d :

Problèmme 2.31 SCS_d Soient P un ensemble de mots et k un entier. Existe-t-il une superchaîne circulaire de P de longueur inférieure ou égale à k ?

En reprenant la preuve de NP-complétude de SLS_d de [25], on a alors la proposition suivante :

Proposition 2.32 Le problème SCS_d est **NP-complet** même si les mots de l'instance sont de longueur 3 et si chaque lettre n'apparaît au maximum qu'une seule fois dans chaque mot.

Preuve Soit HC_d le problème de décision de l'existence d'un cycle hamiltonien dans un graphe orienté (HC_d) (voir Sous-section 2.5.1). Comme HC_d est connu pour être **NP-complet** [26], on va faire une réduction polynomiale de HC_d vers SCS_d.

Soit $G = (V, E)$ un graphe orienté où $V = \{1, \ldots, n\}$ et $|E| = m$. On a un graphe G qui est une instance du problème HC_d, il nous reste à construire une instance de SCS_d, c'est-à-dire un ensemble de mots.

On pose $\Sigma = V \cup \{\overline{v} \mid v \in V\} \cup \{\#\}$ où pour $v \in V$, $\overline{v}$ représente un nouveau caractère qui n'est pas dans V et qui est propre à v et où $\#$ n'est pas un élément de $V \cup \{\overline{v} \mid v \in V\}$. Nous allons construire un ensemble de mots sur l'alphabet Σ. Pour chaque $v \in V$, on pose :

- $N^+(v) := \{w_1, \ldots, w_{d^+(v)}\}$ l'ensemble des voisins sortant de v, où $d^+(v)$ est le degré sortant de v. On suppose que pour tout $v \in V$, $d^+(v) \geq 1$, sinon il n'existerait pas de cycle hamiltonien sur G.
- $A_v := \begin{cases} \{\overline{v}\ w_1\ \overline{v}\ w_1\} & \text{Si } d^+(v) = 1 \\ \{\overline{v}\ w_i\ \overline{v}\ w_{i+_{d^+(v)}1} \mid w_i \in N^+(v)\} & \text{Si } d^+(v) \geq 2 \end{cases}$
- $C_v := \{v\#\overline{v}\}$

On pose $S := \bigcup_{v \in V}(A_v \cup C_v)$.

On va montrer que G a un cycle hamiltonien si et seulement si S admet une superchaîne circulaire de longueur au plus $2m + 3n$.

($\Rightarrow$) Supposons que G a un cycle hamiltonien. Soit (v, w_i) un arc de G. On a alors que $w_i \in N^+(v)$.

On pose alors

$$STD(\overline{v}, w_i) := \begin{cases} \overline{v}\ w_1\ \overline{v}\ w_1 & \text{Si } d^+(v) = 1 \\ \bigoplus_{j=0}^{d^+(v)-1} \overline{v}\ w_{i+_{d^+(v)}1}\ \overline{v}\ w_{i+_{d^+(v)}(j+1)} & \text{Si } d^+(v) \geq 2 \end{cases}$$

où $\bigoplus$ est l'opération de fusion d'un ensemble de mots.

Il est facile de voir que $\{STD(\overline{v}, w_i) \mid w_i \in N^+(v)\}$ est l'ensemble des plus petites superchaînes de A_v. Pour $(u_1, \ldots, u_n)$ un cycle hamiltonien de G, on pose :

$$l((u_1, \ldots, u_n)) := \bigoplus_{i=0}^{n} (u_i\ \#\ \overline{u_i} \oplus STD(\overline{u_i}, u_{i+_n 1}))$$

On a alors $|STD(\overline{v}, w_i)| = \begin{cases} 4 = 2 \times d^+(v) + 2 & \text{Si } d^+(v) = 1 \\ (\sum_{j=0}^{d^+(v)-1} 2) + 2 = 2 \times d^+(v) + 2 & \text{Si } d^+(v) \geq 2 \end{cases}$

et donc

$$|l((u_1, \ldots, u_n))| = \sum_{i=1}^{n} (1 + 2 \times d^+(u_i) + 2) = 3n + 2m$$

($\Leftarrow$) Réciproquement, on commence par borner la taille de la superchaîne circulaire de S.

$$||S|| = \sum_{v=1}^{n} (||A_v|| + ||C_v||) = \sum_{v=1}^{n} (4 \times d^+(v) + 3) = 4m + 3n.$$

Soient v et v' deux éléments de V, $x \in A_v$ et $y \in A_{v'}$ tels que $x \neq y$. Dans le cas où $v = v'$, on a alors que si $v = v'$, $|ov(x, y)| \leq 2$ et sinon $|ov(x, y)| \leq 1$.

Dans le chapitre 3, on va parler un peu plus en détail de la compression, on ne va juste prendre ici que la compression d'une superchaîne w d'un ensemble de mots S est $||S|| - |w|$. De plus on va appliquer le résultat de la section 3.1 qui nous dit que dans le cas d'une superchaîne circulaire issue d'une permutation circulaire, la compression de la superchaîne circulaire est alors égale à la somme de ses chevauchements. On a alors que la compression d'une superchaîne circulaire de S est au plus $\sum_{i=1} n(2+2\times(|A_i|-1)) = 2m$ et donc que la taille d'une superchaîne circulaire de S est d'au moins $||S||-2m = 3n+2m$.

Soient y une superchaîne circulaire de taille $3n + 2m$ et x une sous-chaîne de y entre deux #. Le premier caractère de x est surligné, par exemple $\overline{v}$. Le dernier caractère de x n'est pas surligné, par exemple w_i. On peut remarquer alors que $x = STD(\overline{v}, w_i)$ et on obtient que (v, w_i) est un arc de G. Récursivement on trouve un cycle hamiltonien de G.

En ajoutant $\{\widehat{v} \mid v \in V\}$ à Σ et en remplaçant la définition de A_v par

$$A'_v := \begin{cases} \{\overline{v}\ w_1\ \overline{v}\ w_1\} & \text{Si } d^+(v) = 1 \\ \bigcup_{w_i \in N^+(v)} \{\overline{v} w_i \widehat{v}, w_i \widehat{v} \widehat{w_i}, \widehat{v} \widehat{w_i} \overline{v}, \widehat{w_i} \overline{v} w_{i+_{d^+(v)}1} \mid w_i \in N^+(v)\} & \text{Si } d^+(v) \geq 2 \end{cases}$$

on obtient le même résultat pour un ensemble de mots de longueur 3 et où chaque lettre n'apparaît au maximum qu'une seule fois dans chaque mot. ■

`Q : Le problème` *SCS* `est-il difficile à approximer ?`

2.3 Couverture circulaire de mots

On va examiner plus en détail dans cette section le problème de la plus petite couverture circulaire. On a vu dans la section 2.1 que la plupart des algorithmes pour le problème de la plus petite superchaîne commence par construire une couverture circulaire [11, 88, 18, 3, 4, 5, 14, 86, 44, 68, 63]. Malgré cette utilisation pour aider à construire la plus petite superchaîne, le problème de la plus petite couverture circulaire est très peu étudié. En effet, le seul livre référence pour ce problème est le livre de Papadimitriou et al. [69] où on a une preuve que l'algorithme hongrois résout exactement SCCS en temps $O(|P|^3)$. On va ici définir formellement ce problème et appliquer ce que l'on a fait dans les sections 2.1 et 2.2 à ce problème. On verra dans la sous-section 7.3.3 que grâce à un plongement de l'algorithme glouton pour le problème de la plus petite couverture circulaire de mots dans une structure d'indexation, nous donnerons un algorithme en temps $O(||P||)$ pour résoudre ce problème.

2.3.1 Définition du problème SCCS

Une *couverture circulaire de mots* (≪ *Cyclic Cover of Strings* ≫ en anglais) de P est un ensemble de mots circulaires $C = \{\langle c_1\rangle, \ldots, \langle c_m\rangle\}$ tel que pour tout mot s de P, s soit sous-chaîne d'un mot circulaire $\langle c_i\rangle$ de C, *i.e.* $\exists i \in \{1, \ldots, m\}$ tel que $s \underset{\text{sub}}{\subset} \langle c_i\rangle$.

On peut alors définir le problème de minimisation associé.

Problèmme 2.33 ***Shortest Cyclic Cover of Strings*** *(*SCCS*) (voir Figure 2.9)*
Soit P un ensemble de mots. On cherche la couverture circulaire de mots de P de norme minimale.

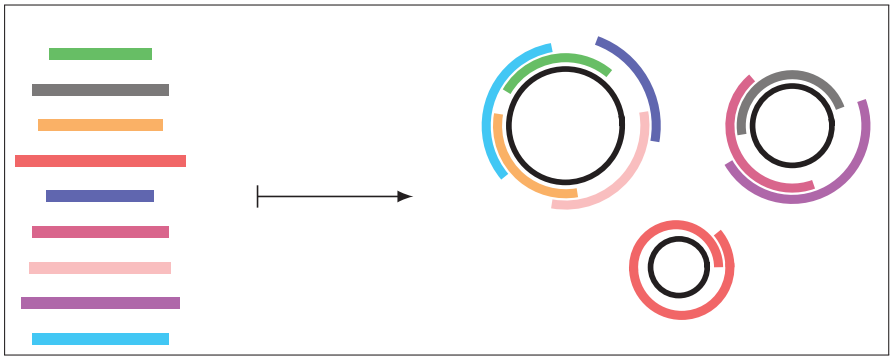


FIGURE 2.9 – Schéma du problème SCCS. Les rectangles de couleurs représentent les mots de l'instance et le rectangle noir correspond à une couverture circulaire de mots de l'instance.

Remarque 2.34 Le problème SCCS est souvent confondu avec le problème de l'affectation minimale (≪ *minimum assignment* ≫ en anglais) qui correspond pour un ensemble de mots linéaires et un ensemble de mots circulaires, de déterminer la position de chaque mot linéaire dans un mot circulaire. Cette confusion peut être par exemple

trouvé dans Blum et al. [11] où on retrouve une mention du fait que l'algorithme glouton est optimal pour le problème de l'affectation minimale alors que dans le contexte le problème visé par l'algorithme glouton est SCCS.

Même si ici, on ne s'y intéresse pas trop, il existe une variante du problème SCCS, qui est appelée version canonique de SCCS et qui interdit les auto-chevauchements des mots de départ comme solution dans la couverture circulaire de mots. Teng et al. [88] ont montré qu'on pouvait trouver un algorithme en $O(|P|^3 + |P| \times ||P||)$ pour résoudre la version canonique de SCCS. Gusfield [34] a présenté une amélioration de cet algorithme en $O(|P|^3 + ||P||)$. L'idée de l'algorithme reste identique, on commence par construire une solution de SCCS en $O(|P|^3)$ grâce à l'algorithme hongrois et ensuite en $O(|P| \times ||P||)$ pour Teng et al. [88] et en $O(||P||)$ pour Gusfield [34], on casse les mots circulaires obtenus et on les fusionne.

La proposition 2.4 (page 33) peut être aussi généralisée à SCCS, on prendra alors pour la suite de cette section un ensemble $P = \{s_1, \ldots, s_n\}$ de mots qui est factor-free.

On va de plus généraliser le concept de superchaîne circulaire issue d'une permutation circulaire en définissant la couverture circulaire de mots issue d'une permutation σ. Soient $P = \{s_1, \ldots, s_n\}$ un ensemble de mots tel que pour tout i entre 1 et $n-1$, $s_i \preceq s_{i+1}$ et σ une permutation de $\{1, \ldots, n\}$. On a alors la définition suivante :

Définition 2.35 ***Couverture circulaire de mots issue d'une permutation (voir Figure 2.10)*** La *couverture circulaire de mots de P issue* de la permutation σ, notée $CC(P, \sigma)$, est telle que :

$$CC(P, \sigma) = \{Circular(P_1, \sigma_1), \ldots, Circular(P_m, \sigma_m)\}$$

où la décomposition de σ en permutations circulaires est $\sigma_1 \ldots \sigma_m$, les partitions $Part_\sigma = \{I_1, \ldots, I_m\}$ de $\{1, \ldots, n\}$ et $\{P_1, \ldots, P_m\}$ de P sont telles que pour tout $i \in \{1, \ldots, m\}$, σ_i est une permutation sur I_i et $P_i = \{s_j \mid j \in I_j\}$.

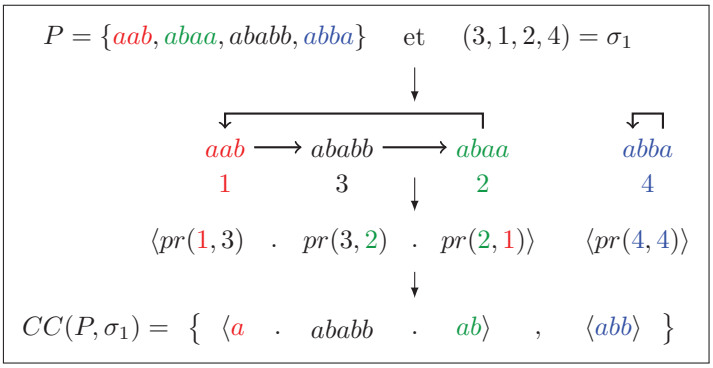


FIGURE 2.10 – Exemple de couverture circulaire de mots issue de la permutation $\sigma_1 = (3, 1, 2, 4) = (1 \to 3 \to 2)(4)$ sur $P = \{aab, abaa, ababb, abba\}$. On a alors $CC(P, \sigma_1) = \{\langle aababbab\rangle, \langle abb\rangle\}$.

Remarque 2.36 On peut généraliser la remarque 2.23 (page 40) au cas de la couverture circulaire de mots. Au lieu d'utiliser l'ordre lexicographique sur Σ^* et une permutation de $\{1,\dots,n\}$, on aurait pu prendre une permutation ρ de P et définir la superchaîne circulaire suivante :

$$CC'(P,\rho) = \{Circular'(P_1,\rho_1),\dots,Circular'(P_m,\rho_m)\}.$$

où la décomposition de ρ en permutations circulaires est $\rho_1 \dots \rho_m$ et la partition $Part_\rho = \{P_1,\dots,P_m\}$ de P est telle que pour tout $i \in \{1,\dots,m\}$, ρ_i est une permutation sur P_i.

Il est aussi facile de voir que les deux définitions sont équivalentes et que les ensembles de couvertures circulaires de mots issues de ces deux définitions sont égales.

Comme pour les problèmes SLS et SCS, on note $PSCCS(P)$ l'ensemble des couvertures circulaires de mots issues des permutations de $\{1,\dots,n\}$, *i.e.* $PSCCS(P) = \{CC(P,\sigma) \mid \sigma \text{ une permutation de } \{1,\dots,n\}\}$. On a alors une proposition équivalente aux propositions 2.12 et 2.25 (voir Figure 2.11).

Proposition 2.37 Soit P un ensemble factor-free. On a alors $OPT_{\mathsf{SCCS}}(P) \subseteq PSCCS(P)$.

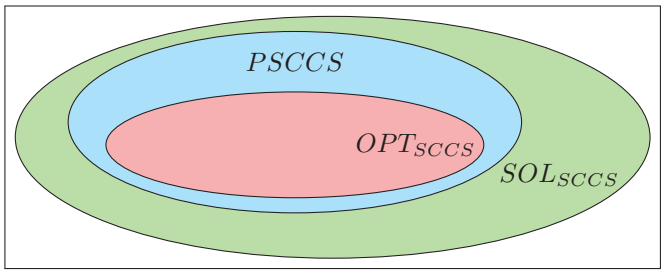


FIGURE 2.11 – Diagramme d'Euler reliant les ensembles $OPT_{\mathsf{SCCS}}(P)$, $PSCCS(P)$ et $SOL_{\mathsf{SCCS}}(P)$ pour tout ensemble de mots P.

On peut alors, comme on l'a fait avec les problèmes SLS et SCS, redéfinir le problème SCCS.

Problèmme 2.38 ***Shortest Cyclic Cover of Strings (SCCS) version 2*** Soit P un ensemble de mots. On cherche une couverture circulaire de mots de P issue d'une permutation de $\{1,\dots,|P|\}$ et de norme minimale.

2.3.2 Lien entre les solutions optimales de SCCS et SCS et SLS

Soit σ_c une permutation circulaire de $\{1,\dots,n\}$. On a alors que $CC(P,\sigma_c)$ est le singleton $\{Circular(P,\sigma_c)\}$. On en déduit que les ensembles constitués des singletons des solutions optimales de SCS sont des éléments de $PSCCS(P)$. On a donc pour $C_{opt} \in OPT_{\mathsf{SCCS}}(P)$ et $\langle w_{opt}\rangle \in OPT_{\mathsf{SCS}}(P)$ que :

$$||C_{opt}|| \leq |\langle w_{opt}\rangle|.$$

Et malheureusement, comme pour SLS et SCS, on ne peut pas trouver de réel α tel que pour tout ensemble de mots P et pour tout $C_{opt} \in OPT_{\mathsf{SCCS}}(P)$ et $\langle w_{opt} \rangle \in OPT_{\mathsf{SCS}}(P)$, $|\langle w_{opt} \rangle| \leq \alpha \times ||C_{opt}||$ (voir Exemple 2.39). On a donc que le ratio entre la norme de C_{opt} et la longueur de w_{opt} peut être aussi grand que souhaité.

Exemple 2.39 Soient n un entier et $P = \{(ab)^n, (ba)^n, (cd)^n, (dc)^n\}$. On a alors que $\{\langle ab \rangle, \langle cd \rangle\} \in OPT_{\mathsf{SCCS}}(P)$ et $\{\langle (ab)^n a (cd)^n c \rangle\} \in OPT_{\mathsf{SCS}}(P)$.

Comme pour le lien entre SLS et SCS, on va prendre une solution optimale de SCCS et on va la modifier pour obtenir une solution de SLS. Pour cela on va utiliser une partie de l'algorithme `Concat-Cycles` de Blum et al. [11] et le fait que cet algorithme est une 4-approximation de SLS. On va reprendre rapidement la définition et la preuve de l'approximation pour exprimer le lien entre SCCS et SLS.

Soit σ une permutation de $\{1, \ldots, n\}$. Soient $\sigma_1 \ldots \sigma_m$ la décomposition en permutations circulaires de σ et les partitions $Part_\sigma = \{I_1, \ldots, I_m\}$ de $\{1, \ldots, n\}$ et $\{P_1, \ldots, P_m\}$ de P telles que pour tout $i \in \{1, \ldots, m\}$, σ_i est une permutation sur I_i et $P_i = \{s_j \mid j \in I_j\}$. On a alors que $CC(P, \sigma) = \{Circular(P_1, \sigma_1), \ldots, Circular(P_m, \sigma_m)\}$.

Pour $I = (i_1, \ldots, i_m)$ tel que pour tout $j \in \{1, \ldots, m\}$, $i_j \in P_j$ et pour $\sigma^\star$ une permutation sur $\{1, \ldots, m\}$, on note

$$ConcatCycles(CC(P, \sigma), I, \sigma^\star) = \bigoplus_{j=1}^{m} Linear(P_{\sigma^\star(j)}, \sigma_{\sigma^\star(j)}, i_{\sigma^\star(j)}).$$

À partir d'un ensemble de mots P, $ConcatCycles(CC(P, \sigma), I, \sigma^\star)$ revient à prendre une couverture circulaire de mots de P, de rendre linéaire chaque composante et ensuite de les fusionner. I correspond à l'endroit où on coupe chaque composante et $\sigma^\star$ l'ordre dans lequel on fusionne les composantes que l'on a rendues linéaires.

En reprenant le résultat de Blum et al. [11], on obtient la proposition suivante :

Proposition 2.40 Soient σ une permutation telle que $CC(P, \sigma) \in OPT_{\mathsf{SCCS}}(P)$ et σ_l une permutation circulaire telle qu'il existe $j \in \{1, \ldots, n\}$ tel que $Linear(P, \sigma_l, j) \in OPT_{\mathsf{SLS}}(P)$. Pour tout $I = (i_1, \ldots, i_m)$ tel que pour tout $j \in \{1, \ldots, m\}$, $i_j \in P_j$ et pour tout $\sigma^\star$ une permutation sur $\{1, \ldots, m\}$ on a :

$$|Linear(P, \sigma_l, j)| \leq |ConcatCycles(CC(P, \sigma), I, \sigma^\star)| \leq 4 \times |Linear(P, \sigma_l, j)|$$

où la décomposition de σ en permutation circulaire est $\sigma_1 \ldots \sigma_m$ et les partitions $Part_\sigma = \{I_1, \ldots, I_m\}$ de $\{1, \ldots, n\}$ et $\{P_1, \ldots, P_m\}$ de P sont telles que pour tout $i \in \{1, \ldots, m\}$, σ_i est une permutation sur I_i et $P_i = \{s_j \mid j \in I_j\}$.

Dans la preuve de cette proposition, on utilise le lemme suivant :

Lemme 2.41 Soient σ une permutation telle que

$$CC(P, \sigma) = \{Circular(P_1, \sigma_1), \ldots, Circular(P_m, \sigma_m)\} \in OPT_{SCCS}(P)$$

et $x_i \in P_i$ et $x_j \in P_j$ avec $i \neq j$. On a alors que :

$$|ov(x_i, x_j)| \leq |Circular(P_i, \sigma_i)| + |Circular(P_j, \sigma_j)|$$

Preuve ***Preuve du Lemme 2.41*** Pour simplifier les notations, on pose $C_i = Circular(P_i, \sigma_i)$ et $C_j = Circular(P_j, \sigma_j)$.

Comme $i \neq j$, $Lin(C_i) \neq Lin(C_j)$. De plus $|C_i|$ n'a aucune période à part elle-même, car si c'était le cas, on pourrait trouver une superchaîne circulaire de P_i de taille strictement plus petite, ce qui nous donnerait une couverture circulaire de mots strictement plus petite, ce qui est absurde.

Supposons qu'il existe u un chevauchement de x_i vers x_j tel que $|u| > |C_i| + |C_j|$. On a alors que $u[1, |C_i|] \in Lin(C_i)$ et $u[1, |C_j|] \in Lin(C_j)$. Comme $Lin(C_i) \neq Lin(C_j)$, on a que $|C_i| \neq |C_j|$. On peut supposer sans perte de généralité que $|C_i| > |C_j|$. On a alors

$$\begin{aligned} u[1, |C_i|] &= u[1 + |C_j|, |C_i| + |C_j|] \\ &= u[1 + |C_j|, |C_i|].u[|C_i| + 1, |C_i| + |C_j|] \\ &= u[1 + |C_j|, |C_i|].u[1, |C_j|] \end{aligned}$$

On a alors que $|C_j|$ est une période de C_i ce qui est absurde. ■

Preuve ***Preuve de la Proposition 2.40*** On sait que la couverture circulaire de mots $CC(P, \sigma) = \{Circular(P_1, \sigma_1), \ldots, Circular(P_m, \sigma_m)\}$, pour tout $i \in \{1, \ldots, m\}$, on pose l_i le plus long mot de P_i et $d_i = |Circular(P_i, \sigma_i)|$.

On a que $\sum_{i=1}^{m} d_i = ||CC(P, \sigma)|| \leq |Linear(P, \sigma_l, j)|$ et d'après le lemme 2.41, la somme des chevauchements maximaux est au plus de $\sum_{i=1}^{m} 2 \times d_i$ et donc la plus petite superchaîne de P est de taille au moins $\sum_{i=1}^{m} l_i - 2 \times d_i$ et donc $\sum_{i=1}^{m} l_i - 2 \times d_i \leq |Linear(P, \sigma_l, j)|$.

On a donc

$$\begin{aligned} |ConcatCycles(CC(P, \sigma), I, \sigma^\star)| &\leq \sum_{i=1}^{m} l_i + d_i \\ &= \sum_{i=1}^{m} l_i - 2 \times d_i + \sum_{i=1}^{m} 3 \times d_i \\ &\leq |Linear(P, \sigma_l, j)| + 3 \times |Linear(P, \sigma_l, j)| \\ &= 4 \times |Linear(P, \sigma_l, j)| \end{aligned}$$

`Q : Peut-on trouver une instance où on a égalité pour le Lemme 2.41 ?`

`Q : La borne de 4-approximation de la proposition 2.40 est-elle stricte ?`

2.4 Couvertures linéaire et mixte de mots

Dans cette section, on va parler des problèmes de la plus petite couverture linéaire de mots et de la plus petite couverture mixte de mots. Ce sont deux problèmes que nous avons rencontrés en étudiant des algorithmes liés à l'assemblage et que nous avons proposés. Dans la section 5.3, on va définir des variantes de ces deux problèmes. On va ici montrer que ces deux problèmes sont des généralisations de deux problèmes que l'on a déjà présentés : le problème de la plus petite superchaîne et le problème de la plus petite couverture circulaire de mots.

2.4.1 Couverture linéaire de mots

Une *couverture linéaire de mots* (« *Linear Cover of Strings* » en anglais) de P est un ensemble de mots $L = \{w_1, \ldots, w_m\}$ tel que tout mot s de P soit sous-chaîne d'un mot

w_i de L, *i.e.* $\exists i \in \{1, \dots, m\}$ tel que $s \underset{\text{sub}}{\subset} w_i$.

On peut alors définir le problème de minimisation associé.

Problèmme 2.42 ***Shortest Linear Cover of Strings*** *(*SLCS*) (voir Figure 2.12)* Soit P un ensemble de mots. On cherche une couverture linéaire de mots de P de norme minimale.

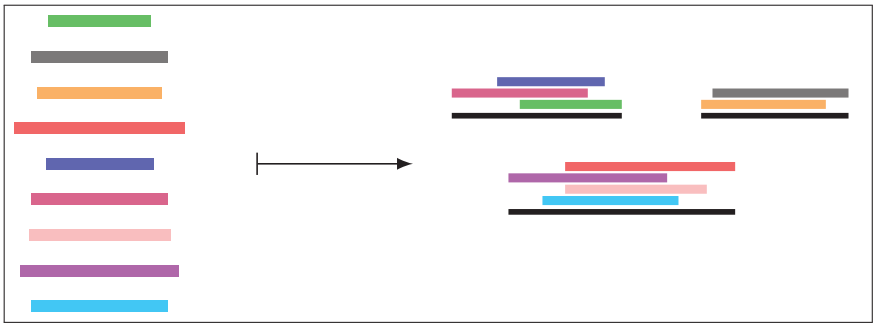


FIGURE 2.12 – Schéma du problème SLCS. Les rectangles de couleur représentent les mots de l'instance et le rectangle noir correspond à une couverture linéaire de mots de l'instance.

On peut comme on l'a fait avec les problèmes SLS, SCS et SCCS, redéfinir les solutions de SLCS à l'aide d'une permutation. On aura en plus besoin d'un sous-ensemble de $\{1, \dots, n\}$ qui prend une position dans chaque élément de la partition $Part_\sigma$. On pose $Coupe_\sigma$ l'ensemble de tous ces sous-ensembles, c'est-à-dire

$$Coupe_\sigma = \{\{e_1, \dots, e_m\} \mid \forall j \in \{1, \dots, m\},\ e_j \in P_j \text{ où } Part_\sigma = \{P_1, \dots P_m\}\}.$$

Soient $Part_\sigma = \{P_1, \dots P_m\}$ et $E \in Coupe_\sigma$. On a alors que pour tout j entre 1 et m, $|P_j \cap E| = 1$. On donne alors la définition suivante :

Définition 2.43 ***Couverture linéaire de mots issue d'une permutation et d'un ensemble*** La *couverture linéaire de mots issue* de la permutation σ et de l'ensemble $E \in Coupe_\sigma$, notée $LC(P, \sigma, E)$, est telle que :

$$LC(P, \sigma, E) = \{Linear(P_1, \sigma_1, e_1), \dots, Linear(P_m, \sigma_m, e_m)\}$$

où la décomposition de σ en permutations circulaires est $\sigma_1 \dots \sigma_m$, les partitions $Part_\sigma = \{I_1, \dots, I_m\}$ de $\{1, \dots, n\}$ et $\{P_1, \dots, P_m\}$ de P sont telles que pour tout $i \in \{1, \dots, m\}$, σ_i est une permutation sur I_i et $P_i = \{s_j \mid j \in I_j\}$ et $E = \{e_1, \dots, e_m\}$ est tel que pour tout j entre 1 et m, $P_j \cap E = \{e_j\}$.

Pour un ensemble de mots P, on pose $PSLCS(P)$ l'ensemble des couvertures linéaires de mots issue d'une permutation et d'un ensemble.

Problèmme 2.44 ***Shortest Linear Cover of Strings*** *(*SLCS*) version 2* Soit P un ensemble de mots. On cherche une couverture linéaire de mots de $PSLCS(P)$ de norme minimale.

On peut remarquer que pour tout ensemble de mots P, $\{\{x\} \mid x \in SOL_{\mathsf{SLS}}(P)\} \subseteq SOL_{\mathsf{SLCS}}(P)$. On va poser l'application T qui va de $SOL_{\mathsf{SLCS}}(P)$ vers $SOL_{\mathsf{SLS}}(P)$ telle que

$$T(L) := \{\bigoplus_{w \in I} w \mid I = L\}.$$

Remarque 2.45 L'ensemble $T(L)$ réunit toutes les fusions possibles données par tous les ordres sur L.

Proposition 2.46 Soient P un ensemble de mots, $w \in OPT_{SLS}(P)$, $L \in OPT_{SLCS}(P)$ et $X \in T(L)$. On a alors que $||L|| = |X| = |w|$.

Preuve On a de manière immédiate que $|X| \leq ||L||$ car pour tous mots w_1 et w_2 de Σ^*, $|w_1 \oplus w_2| \leq |w_1| + |w_2|$. De plus, comme $L \in OPT_{\mathsf{SLCS}}(P)$ et que $\{w\} \in SOL_{\mathsf{SLCS}}(P)$, on a que $||L|| \leq ||\{w\}|| = |w|$. Comme $X \in SOL_{\mathsf{SLS}}(P)$ et $w \in OPT_{\mathsf{SLS}}(P)$, on a aussi que $|w| \leq |X|$. Enfin, en rassemblant toutes ces inégalités on obtient :

$$|X| \leq ||L|| \leq |w| \leq |X|.$$

Le problème SLCS, même s'il ressemble à SCCS, est un problème difficile. En effet, soit SLCS_d le problème de décision associé à SLCS, *i.e.* pour un ensemble de mots P et un entier k, on cherche à savoir s'il existe une couverture linéaire de mots de P dont la norme est plus petite ou égale à k. On a alors la proposition suivante :

Proposition 2.47 Le problèmes SLCS_d est **NP-complet**.

Preuve Pour montrer la NP-complétude de SLCS_d, il suffit de montrer qu'il est aussi difficile que le problème SLS_d. D'après la proposition 2.46, on a que pour une solution optimale L de SLCS, tout élément de $T(L)$ est une solution optimale de SLS. SLCS_d est donc aussi difficile que SLS_d. ■

On peut même aller un peu plus loin sur les résultats d'un algorithme de SLS sur SLCS. En effet, si on a un algorithme sur SLS, il donne aussi une solution de SLCS et comme la taille des solutions optimales de SLS est égale à la taille des solutions optimales de SLCS, le ratio d'optimisation est conservé. On obtient alors le corollaire suivant :

Corollaire 2.48 Soit $\mathcal{A}$ un algorithme sur SLS avec un ratio d'approximation optimal α. On a alors que $\mathcal{A}$ est aussi un algorithme sur SLCS avec le même ratio d'approximation optimal et la même complexité.

2.4.2 Couverture mixte de mots

Une *couverture mixte de mots* (« *Mixed Cover of Strings* » en anglais) de P est un ensemble de mots linéaires et de mots circulaires $M = \{x_1, \ldots, x_m\}$ tel que tout mot s de P soit sous-chaîne d'un mot x_i de M, *i.e.* $\exists i \in \{1, \ldots, m\}$ tel que $s \underset{\text{sub}}{\subset} x_i$.

On a alors le problème suivant :

Problèmme 2.49 ***Shortest Mixed Cover of Strings*** *(*SMCS*)* ***(voir Figure 2.13)*** Soit P un ensemble de mots. On cherche la couverture mixte de mots de P de norme minimale.

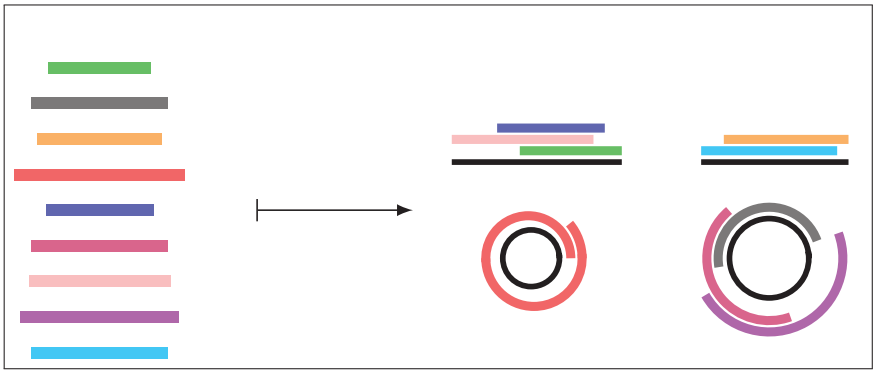


FIGURE 2.13 – Schéma du problème SMCS. Les rectangles de couleur représentent les mots de l'instance et le rectangle noir correspond à une couverture mixte de mots de l'instance.

Comme on l'a fait pour le problème SLCS, on va définir l'ensemble $Presque - Coupe_\sigma$ des sous-ensembles des ensembles de $Coupe_\sigma$, *i.e.*

$$Presque - Coupe_\sigma = \{X \mid \exists Y \in Coupe_\sigma \text{ tel que } X \subseteq Y\}$$

On peut alors définir une couverture mixte de mots issue d'une permutation et d'un ensemble.

Définition 2.50 ***Couverture mixte de mots issue d'une permutation et d'un ensemble*** La *couverture mixte de mots issue* de la permutation σ et de l'ensemble $E \in Presque - Coupe_\sigma$, notée $MC(P, \sigma, E)$, est telle que :

$$MC(P, \sigma, E) = \{w_1, \ldots, w_m\}$$

où pour j entre 1 et m,

$$w_j = \begin{cases} Linear(P_j, \sigma_j, e_j) & \text{si } |P_j \cap E| = 1 \text{ où } P_j \cap E = \{e_j\} \\ Circular(P_j, \sigma_j) & \text{si } |P_j \cap E| = 0 \end{cases}$$

et la décomposition de σ en permutations circulaires est $\sigma_1 \ldots \sigma_m$, les partitions $Part_\sigma = \{I_1, \ldots, I_m\}$ de $\{1, \ldots, n\}$ et $\{P_1, \ldots, P_m\}$ de P sont telles que pour tout $i \in \{1, \ldots, m\}$, σ_i est une permutation sur I_i et $P_i = \{s_j \mid j \in I_j\}$.

Pour un ensemble de mots P, on pose $PSMCS(P)$ l'ensemble des couvertures mixtes de mots issue d'une permutation et d'un ensemble et on peut redéfinir le problème SMCS.

Problèmme 2.51 ***Shortest Mixed Cover of Strings*** *(*SMCS*)* ***version 2*** Soit P un ensemble de mots. On cherche une couverture mixte de mots de $PSMCS(P)$ de norme minimale.

On peut alors faire un lien entre SMCS et SCCS : $SOL_{\mathsf{SCCS}}(P) \subseteq SOL_{\mathsf{SMCS}}(P)$. En effet, par définition toutes les couvertures circulaires de mots sont des couvertures mixtes

de mots. On peut de plus comparer les solutions optimales de SMCS et les solutions optimales de SCCS. Pour réaliser cela, on va alors définir l'application L de $SOL_{\mathsf{SMCS}}(P)$ vers $SOL_{\mathsf{SCCS}}(P)$ telle que

$$L(M) = \{\langle w\rangle \mid \langle w\rangle \text{ un mot circulaire de } M\} \bigcup \{\langle pr(w,w)\rangle \mid w \text{ un mot linéaire de } M\}.$$

On a alors la proposition suivante :

Proposition 2.52 Soient P un ensemble de mots, $C \in OPT_{\mathsf{SCCS}}(P)$ et $M \in OPT_{\mathsf{SMCS}}(P)$. On a alors que $||M|| = ||L(M)|| = ||C||$.

Preuve On a de manière immédiate que $||L(M)|| \leq ||M||$ car pour tout w mot de Σ^*, $|\langle pr(w,w)\rangle| \leq |w|$. De plus, comme $M \in OPT_{\mathsf{SMCS}}(P)$ et que $C \in OPT_{\mathsf{SCCS}}(P) \subseteq SOL_{\mathsf{SMCS}}(P)$, on a que $||M|| \leq ||C||$. Comme $L(M) \in SOL_{\mathsf{SCCS}}(P)$ et $C \in OPT_{\mathsf{SCCS}}(P)$, on a aussi que $||C|| \leq ||L(M)||$. Enfin, en rassemblant toutes ces inégalités, on obtient :

$$||L(M)|| \leq ||M|| \leq ||C|| \leq ||L(M)||.$$

Grâce à la proposition 2.52, on peut alors faire un lien entre la complexité de SCCS et de SMCS.

Corollaire 2.53 Soit $\mathcal{A}$ un algorithme sur SCCS avec un ratio d'approximation α. On a alors que $\mathcal{A}$ est aussi un algorithme sur SMCS avec le même ratio d'approximation et la même complexité.

En reprenant l'algorithme de Papadimitriou et al. [69], grâce au corollaire 2.53, on sait qu'il existe pour un ensemble de mots P, un algorithme en $|P|^3$ pour résoudre SMCS. On a alors que SMCS est un problème de la classe de complexité **P**. On montrera dans la sous-section 7.3.3 que l'on peut résoudre en temps $O(||P||)$ le problème SCCS, on peut alors résoudre de la même manière en temps $O(||P||)$ le problème SMCS.

Conclusion de la section Même si de manière générale, les problèmes SLCS et SMCS sont intéressants par eux-mêmes, dans la suite, on ne va pas s'intéresser directement à ces problèmes. En effet, d'après le corollaire 2.48 (respectivement 2.53), si on trouve un algorithme d'approximation pour SLS (respectivement SCCS), on peut l'appliquer au problème SLCS (respectivement SMCS).

Par contre, dans la section 5.3, on va s'intéresser au problème des superchaînes auquel on ajoute un ensemble de chevauchements qui sont interdits dans la superchaîne. Pour ce genre de problème et pour certaines instances, on a qu'aucune superchaîne linéaire ne peut résoudre le problème, mais il existe des couvertures linéaires de mots qui peuvent le résoudre. De même, quand aucune couverture circulaire de mots ne peut être trouvée, on va chercher une couverture mixte de mots. Dans ces conditions, on va étudier des variantes des problèmes SLCS et SMCS.

2.5 Graphe des préfixes et liens avec les superchaînes

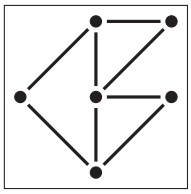
Dans cette dernière partie, on va redonner la définition du graphe des préfixes ainsi que quelques propriétés puis on va s'intéresser aux liens entre les types de parcours de ce graphe et les problèmes que l'on a énoncés dans les parties précédentes.

2.5.1 Notation sur les graphes

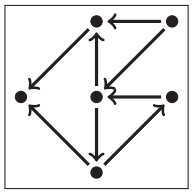
On va commencer par poser quelques notations sur les graphes. Dans la théorie des graphes, on fait la différence entre un graphe (ou graphe non orienté) et un graphe orienté. On a à peu près les mêmes notions sur les deux types de graphe, on va les expliciter ici.

Remarque 2.54 On utilise les notations sur les graphes pour la preuve de la proposition 2.32. En outre celle pour le degré diffère.

Soit $G = (V, E)$ un *graphe*, V est l'ensemble des *nœuds* et E est l'ensemble des *arêtes* (voir Figure 2.14a). Un *graphe orienté* $G = (V, A)$ est un graphe où on a orienté les arêtes, on parle alors d'*arcs* (voir Figure 2.14b). On pose V_G (resp. E_G ou A_G) l'ensemble des nœuds (resp. des arêtes ou des arcs) de G. Un graphe que n'est pas orienté est dit *non-orienté*. On a alors pour un arc un *début* et une *fin*. Le début et la fin d'un arc sont appelés les *extrémités* de cet arc. Même si une arête n'a pas de début ni de fin, elle est composée de deux sommets, que l'on va appeler *extrémités* de l'arête.



(a) Exemple de graphe



(b) Exemple de graphe orienté

FIGURE 2.14 – Exemple de graphe et de graphe orienté. Ici et dans la suite, on représente les nœuds par des points, les arêtes par des traits entre ces points et les arcs par des flèches d'un point qui correspond au début de l'arc vers un autre point qui correspond à la fin de l'arc.

On dit que deux arcs (respectivement arêtes) a_1 et a_2 sont *adjacents* s'ils partagent une extrémité. On dit qu'un arc a_1 est *suivi* par un autre arc a_2 si la fin de a_1 est le début de a_2.

Pour un graphe non-orienté, une *chaîne* entre deux nœuds x et y est un n-uplet $(a_1, \ldots, a_n)$ d'arêtes de A tel que x est une extrémité de a_1, y une extrémité de a_n et pour tout $i \in \{1, \ldots, n-1\}$, a_i et a_{i+1} sont adjacents. De même un *cycle* est un n-uplet circulaire $(a_1, \ldots, a_n)$ d'arêtes de A tel que pour tout $i \in [1, n]$, a_i et $a_{i+_n 1}$ sont adjacents. Un cycle est une chaîne entre un nœud et lui-même.

Pour un graphe orienté, un *chemin* entre deux nœuds x et y est un n-uplet $(a_1, \ldots, a_n)$ d'arcs de A tel que x est le début de a_1, y est la fin de a_n et pour tout $i \in [1, n-1]$, a_i est suivie par a_{i+1}. De même un *circuit* est un n-uplet circulaire $(a_1, \ldots, a_n)$ d'arcs de A tel que pour tout $i \in [1, n]$, a_i est suivi par $a_{i+_n 1}$. Un circuit est un chemin entre un nœud et lui-même.

Si toute les arêtes du chemin (respectivement de la chaîne) sont distinctes, alors le chemin (respectivement la chaîne) est dit(e) *simple*. La *taille* d'une chaîne, circuit, chemin ou cycle est son nombre d'arêtes. On dit qu'une arête e est dans une chaîne p (respectivement cycle, chemin ou circuit), notée $e \in p$, si e est un élément pris par la chaîne (e est un élément du n-uplet).

Soient u et v deux nœuds de G, on dit que v est un *successeur* de u s'il existe un arc dans A qui commence sur u et qui finit sur v. Le nœud u est alors un *prédécesseur* de v. Si u est un prédécesseur ou un successeur de v, on dit que u et v sont *adjacents*. Le *degré entrant* d'un nœud v, noté $d^{in}(v)$, est le nombre de prédécesseurs de v et le *degré sortant*, noté $d^{out}(v)$, est le nombre de successeurs de v. Pour un graphe orienté G, on définit l'*ordre* de G, noté $Ord(G)$, comme le maximum entre le degré entrant et le degré sortant de tous les nœuds de G.

On dit qu'un graphe est *connexe* si pour tout couple de nœuds (x, y), il existe une chaîne entre x et y. Une *composante connexe* d'un graphe (V, E) a un sous-ensemble X de V tel que $\forall (x, y) \in X^2$, il existe une chaîne entre x et y et $\forall (x, y) \in X \times (E \setminus X)$, il n'existe pas de chaîne entre x et y. Un graphe orienté est dit *fortement connexe* si pour tout couple de sommets (x, y), il existe un chemin de x vers y.

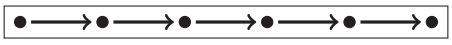
On dit qu'un chemin simple est *eulérien* sur G s'il passe une et une seule fois par tous les arcs de G. Un *multi-chemin* (simple) d'un graphe G est un ensemble de chemins (simples) sur les composantes connexes de G. On dit qu'un multi-chemin simple p est *semi-eulérien* sur G si pour chaque composante connexe G_i de G, il existe un et un seul chemin simple p_i du multi-chemin simple p tel que p_i est eulérien sur G_i. S'il existe un chemin eulérien (respectivement multi-chemin semi-eulérien) sur un graphe, on dit que ce graphe est *eulérien* (respectivement *semi-eulérien*).

De même, on dit qu'un chemin simple est *hamiltonien* sur G s'il passe une et une seule fois par tous les nœuds de G. On dit alors qu'un multi-chemin simple p est *semi-hamiltonien* sur G si pour chaque composante connexe G_i de G, il existe un et un seul chemin simple p_i du multi-chemin simple p tel que p_i est hamiltonien sur G_i. S'il existe un chemin hamiltonien (respectivement multi-chemin semi-hamiltonien) sur un graphe, on dit que ce graphe est *hamiltonien* (respectivement *semi-hamiltonien*).

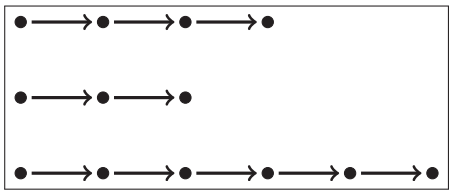
Un graphe est dit *acyclique* si on ne peut trouver aucun cycle dans le graphe. Un graphe non orienté acyclique est appelé une *forêt*. Un *arbre* est une forêt connexe. Un *arbre enraciné* est un arbre où on a spécifié un nœud particulier que l'on appelle la *racine*. Comme un arbre n'a aucun cycle, pour un autre nœud de l'arbre que la racine, il existe une unique chaîne simple entre ce nœud et la racine. La *profondeur* d'un nœud est la longueur de sa chaîne simple entre ce nœud et la racine. Soit v un nœud de l'arbre différent de la racine, on a alors que v a un unique nœud adjacent de profondeur plus petite, c'est le *parent* de v dans l'arbre et les nœuds adjacents à v de profondeurs plus grandes sont les *enfants* de v dans l'arbre. On pose $p_T(v)$ le parent de v dans l'arbre T et $Children_T(v)$ l'ensemble des enfants de v dans l'arbre T. Un nœud qui n'a pas d'enfants est appelé une *feuille* et un nœud qui a au moins un enfant est appelé un *nœud interne*. Un nœud u est un *ancêtre* d'un autre nœud v s'il existe un ensemble $\{x_1, \ldots, x_m\}$ de nœuds de l'arbre tel que $x_1 = u$, $x_m = v$ et pour tout $i \in [1, m-1]$, x_{i+1} est un enfant de x_i. La notion d'ancêtre nous donne un sous-ordre sur l'arbre et peut être vue comme une orientation des arêtes où on met un arc de u vers v si v est un fils de u. On confondra alors souvent l'arbre non orienté enraciné avec l'arbre orienté à partir de la racine vers ses feuilles ainsi que les notions de chaîne et de chemin dans un arbre.

On peut alors définir quelques graphes orientés particuliers :

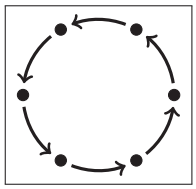
— Un *graphe chemin* est un graphe orienté acyclique connexe où chaque nœud a au plus un successeur et un prédécesseur.



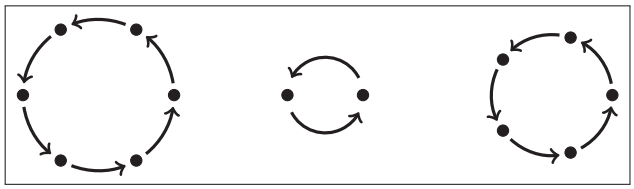
— Un *graphe multi-chemin* est un graphe orienté acyclique où chaque nœud a au plus un successeur et un prédécesseur.



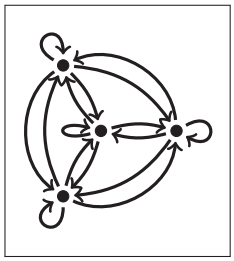
— Un *graphe circuit* est un graphe orienté connexe où chaque nœud a exactement un successeur et un prédécesseur.



— Un *graphe multi-circuit* est un graphe orienté où chaque nœud a exactement un successeur et un prédécesseur.



— Un *graphe orienté complet* est un graphe où entre chaque nœud u et v, il existe un arc de u vers v et un arc de v vers u.



Soit M un ensemble de chemins et de circuits d'un graphe orienté G. On dit que M *couvre* G si tout nœud de G est une extrémité d'un chemin ou d'un circuit de M. Si M couvre G alors M est une *couverture mixte* de G. Si M n'est composé que de chemins, on parle alors de *couverture linéaire*, et si au contraire, M n'est composé que de circuits, on parle de *couverture cyclique*.

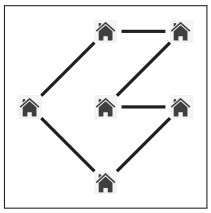
2.5.2 Problèmes sur les graphes

On va maintenant définir quelques problèmes sur les graphes.

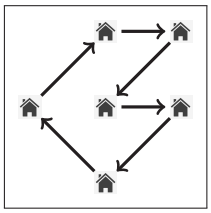
Problèmme 2.55 ***Minimum Hamiltonian Directed Path*** *(*Min-HDP*)* Soit G un graphe orienté complet et w une pondération sur les arcs de G, on cherche un chemin hamiltonien p de G qui minimise le poids p, où $|p| = \sum_{e \in p} w(e)$.

Remarque 2.56 Quand on parle ici de taille d'un chemin hamiltonien sur un graphe orienté muni d'une pondération w, on parle de la somme des poids du chemin. On peut alors généraliser cette notation à la taille d'un circuit et à la norme d'un ensemble de chemins et de circuits.

Le problème Min-HDP est aussi connu sous le nom de Min-ATSP dans la littérature (on peut retrouver ce problème par exemple dans Papadimitriou et al. [69]). ATSP (« *Asymmetric Traveling Salesman Problem* » en anglais), est traduit en français par le problème du voyageur de commerce asymétrique. Le problème TSP (« *Traveling Salesman Problem* » en anglais) est une référence à un problème classique où un voyageur de commerce doit visiter plusieurs villes et il cherche le trajet le plus court passant par toutes les villes en utilisant les routes. On peut alors modéliser le problème comme un graphe où les villes sont les nœuds, les arêtes sont les routes pour aller d'une ville à l'autre et le poids d'une arête est la distance entre les deux villes. On cherche alors la chaîne hamiltonienne de poids minimal sur ce graphe pour résoudre le problème Min-TSP. Le problème Min-ATSP est le même problème du voyageur de commerce mais où le voyageur de commerce ne veut plus minimiser la distance mais le temps. On a alors que le temps pour aller d'une ville A vers une ville B est différent du temps pour aller de la ville B jusqu'à la ville A. On modélise alors ce problème avec un graphe orienté et on cherche le chemin hamiltonien de poids minimal sur ce graphe orienté pour résoudre Min-ATSP. Sahni et al. [79] a montré que le problème Min-ATSP était **APX-Difficile**.



(a) Modélisation de TSP



(b) Modélisation de ATSP

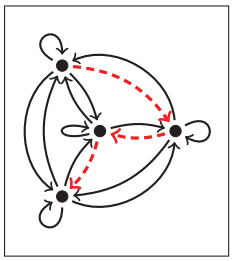
FIGURE 2.15 – Modélisation de TSP et ATSP sous forme de graphe et de graphe orienté.

Problèmme 2.57 ***Minimum Hamiltonian Directed Cycle*** *(*Min-HDC*)* Soient G un graphe orienté complet et w une pondération sur les arcs de G. On cherche un circuit hamiltonien c de G qui minimise la taille de c, où $|c| = \sum_{e \in c} w(e)$.

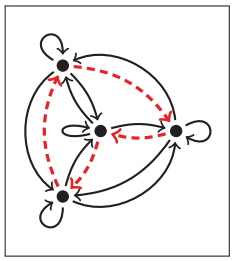
Problèmme 2.58 ***Minimum Hamiltonian Directed Cyclic Cover*** *(*Min-HDCC*)* Soit G un graphe orienté complet et w une pondération sur les arcs de G, on cherche une couverture cyclique C de G qui minimise la norme de C.

Pour chaque problème de minimisation Min-$\mathcal{P}$, il existe un problème de maximisation Max-$\mathcal{P}$. Par exemple, on peut définir Max-HDP :

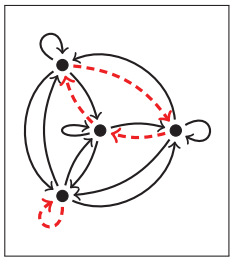
Problèmme 2.59 ***Maximum Hamiltonian Directed Path*** *(*Max-HDP*)* Soit G un graphe orienté complet et w une pondération sur les arcs de G, on cherche un chemin hamiltonien p de G qui maximise la taille de p, où $|p| = \sum_{e \in p} w(e)$.



(a) Chemin hamiltonien



(b) Circuit hamiltonien



(c) Couverture cyclique

FIGURE 2.16 – Exemple de chemin hamiltonien, de circuit hamiltonien et de couverture cyclique sur un graphe orienté complet.

On peut définir de même les problèmes Max-HDC et Max-HDCC. On a alors que :

— Min-HDP, Max-HDP, Min-HDC et Max-HDC sont **NP-Difficiles**,

— Min-HDCC et Max-HDCC sont de complexité cubique. On obtient cette complexité en utilisant l'algorithme hongrois définit dans Papadimitriou et al. [69] qui a une complexité en $O(|V|^3)$ où V est l'ensemble des nœuds de l'instance.

2.5.3 Graphe des préfixes

On va présenter un graphe qui fait le lien entre ces problèmes de graphes et les problèmes de superchaînes : le graphe des préfixes. On trouve une définition de ce graphe dans l'article de Turner [90].

Définition 2.60 ***Graphe des préfixes (voir Figure 2.17)*** Soit P un ensemble de mots. Le *graphe des préfixes* de P est le graphe orienté pondéré sur ses arcs $G = (V, A, w)$ où :

$$V := P \cup \{p\}$$

$$A := (P \times P) \cup (P \times \{p\})$$

$$\begin{array}{rccc} w : & A & \to & \mathbb{N} \\ & (u,v) & \mapsto & \begin{cases} |pr(u,v)| & \text{Si } (u,v) \in P \times P \\ |u| & \text{Si } (u,v) \in P \times \{p\} \end{cases} \end{array}$$

Dans le graphe des préfixes, on peut voir p qui correspond au puits comme un mot vide : en effet pour tout mot u de P, $w((u,p)) = |pr(u,\varepsilon)| = |u|$. De la définition du graphe des préfixes, on confond les mots de P et les nœuds du graphe des préfixes de P.

On va alors définir trois propriétés que peut avoir un graphe. La première consiste à savoir s'il est plus rapide d'aller directement d'un nœud vers un autre que de passer par un troisième. Les deux suivantes viennent de Monge [62]. Soit $G = (V, A, w)$ un graphe orienté pondéré sur ses arcs.

— On dit que G satisfait l'*inégalité triangulaire* si et seulement si pour tout quadruplet de sommets x, y et z, on a

$$w((x,y)) + w((y,z)) \geq w((x,z))$$

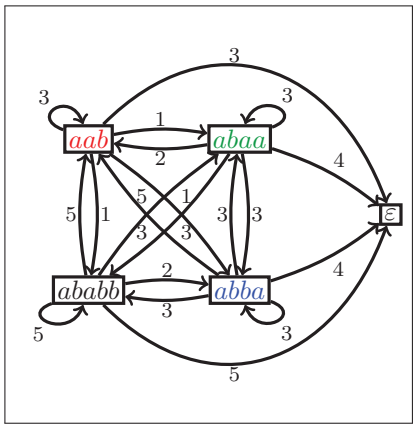


FIGURE 2.17 – Exemple de graphe des préfixes pour $P = \{aab, abaa, ababb, abba\}$. On a représenté le nœud p qui correspond au puits par ε.

— On dit que G satisfait l'*inégalité inférieure de Monge* si et seulement si pour tout quadruplet de sommets u, v, x et y tels que si $w((u,v)) \leq w((u,x))$ et $w((u,v)) \leq w((y,v))$ on a

$$w((u,v)) + w((y,x)) \leq w((u,x)) + w((y,v))$$

— On dit que G satisfait l'*inégalité supérieure de Monge* si et seulement si pour tous sommets u, v, x et y tels que si $w((u,v)) \geq w((u,x))$ et $w((u,v)) \geq w((y,v))$ on a

$$w((u,v)) + w((y,x)) \geq w((u,x)) + w((y,v))$$

On peut alors trouver quelques propriétés connues du graphe des préfixes :

Proposition 2.61 Soit P un ensemble de mots. Le graphe des préfixes de P satisfait l'inégalité triangulaire et l'inégalité inférieure de Monge.

Preuve

Inégalité triangulaire : Soient x, y et z trois mots. On veut montrer que

$$|pr(x,y)| + |pr(y,z)| \geq |pr(x,z)|.$$

Posons $u := ov(ov(x,y),z)$, on a alors que u est un chevauchement de x vers z. On a donc que $|u| \leq |ov(x,z)|$.

$$\begin{aligned} |pr(x,z)| &= |x| - |ov(x,z)| \\ &\leq |x| - |ov(ov(x,y),z)| \\ &= |x| - |ov(x,y)| + |pr(ov(x,y),z)| \\ &= |x| - |x| + |pr(x,y)| + |pr(y,z)| \\ &= |pr(x,y)| + |pr(y,z)|. \end{aligned}$$

Inégalité inférieure de Monge : Soient u, v, x et y quatre mots tels que $|pr(u,v)| \leq |pr(u,x)|$ et $|pr(u,v)| \leq |pr(y,v)|$ on veut montrer que

$$|pr(u,v)| + |pr(y,x)| \leq |pr(u,x)| + |pr(y,v)|.$$

On a de plus que

$$\begin{array}{rrcl} & |pr(u,v)| & \leq & |pr(u,x)| \\ \Leftrightarrow & |u| - |pr(u,v)| & \geq & |u| - |pr(u,x)| \\ \Leftrightarrow & |ov(u,v)| & \geq & |ov(u,x)|. \end{array}$$

De même, on a que $|ov(u,v)| \geq |ov(y,v)|$ et donc d'après la proposition 3.33, on obtient

$$|ov(u,v)| + |ov(y,x)| \geq |ov(u,x)| + |ov(y,v)|.$$

On en déduit l'inégalité

$$\begin{aligned} |pr(u,v)| + |pr(y,x)| &= |u| - |ov(u,v)| + |y| - |ov(y,x)| \\ &\leq |u| + |y| - |ov(u,x)| - |ov(y,v)| \\ &= |pr(u,x)| + |pr(y,v)|. \end{aligned}$$

Attention, l'inégalité triangulaire et l'inégalité inférieure de Monge ne suffisent pas pour définir un graphe des préfixes. En effet, un graphe qui satisfait ces deux conditions n'est pas forcément un graphe des préfixes issu d'un ensemble de mots (voir Figure 2.18).

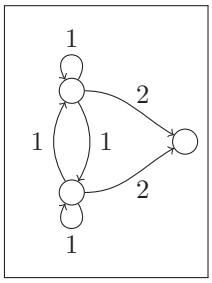


FIGURE 2.18 – Exemple de graphe satisfaisant l'inégalité triangulaire et l'inégalité inférieure de Monge, mais qui n'est pas un graphe des préfixes. En effet, il faudrait, pour obtenir un tel graphe, un ensemble de mots $\{aa, aa\}$ qui n'est pas un ensemble.

2.5.4 Liens entre le graphe des préfixes et SLS

En reprenant les travaux de Turner [90], on peut montrer qu'il existe un lien entre les problèmes Min-HDP et SLS. En effet, il se base sur ce lien pour utiliser les algorithmes pour Min-HDP pour trouver une solution de SLS. On va présenter formellement ce lien, ce qui va nous permettre dans un second temps de généraliser ce lien aux problèmes SCS et SCCS.

On va montrer comment passer d'une superchaîne d'un ensemble de mots P à un chemin hamiltonien sur le graphe des préfixes de P, et inversement. Grâce à la proposition 2.12, on ne va regarder que les éléments de $PSLS(P)$. À partir d'un mot

$w \in PSLS(P)$, on va définir le chemin simple $sol_{LS\to HP}(w)$ sur le graphe des préfixes de P : comme $w \in PSLS(P)$, il existe une permutation σ de $\{1,\dots,n\}$ telle que $w = P_l(\sigma)$ et on pose alors

$$sol_{LS\to HP}(w) := \big((s_{\sigma(1)}, s_{\sigma(2)}), \dots, (s_{\sigma(n-1)}, s_{\sigma(n)})\big).$$

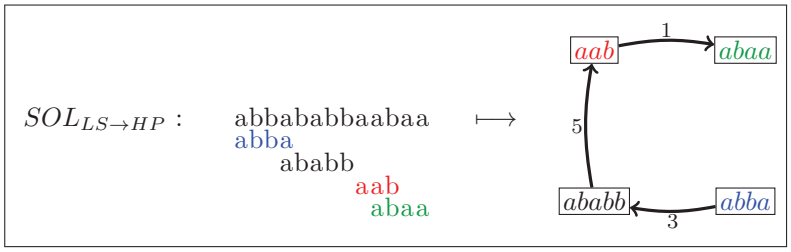


FIGURE 2.19 – Exemple de l'application $sol_{LS\to HP}$ pour $w = abbababbaabaa$ sur $P = \{aab, abaa, ababb, abba\}$.

Remarque 2.62 Soit w un mot de $PSLS(P)$. Le chemin simple $sol_{LS\to HP}(w)$ passe par tous les mots de P dans le graphe des préfixes de P. On a alors que $sol_{LS\to HP}(w)$ est un chemin hamiltonien du graphe qui correspond au graphe des préfixes de P où on ne passe pas par le nœud p du graphe des préfixes de P (*i.e.* le graphe des préfixes réduit de P). On en déduit que $sol_{LS\to HP}(w).(s_{\sigma(n)}, p)$ est un chemin hamiltonien du graphe des préfixes (où σ est la permutation de $\{1,\dots,n\}$ telle que $w = P_l(\sigma)$).

Dans l'autre sens, à partir d'un chemin hamiltonien $q := ((u_1,u_2),\dots,(u_n,u_{n+1}))$ du graphe des préfixes de P, on va définir la superchaîne $sol_{HP\to LS}(q)$ de P : on pose alors

$$sol_{HP\to LS}(q) := \bigoplus_{j=1}^{n} pr(u_j, u_{j+1})$$

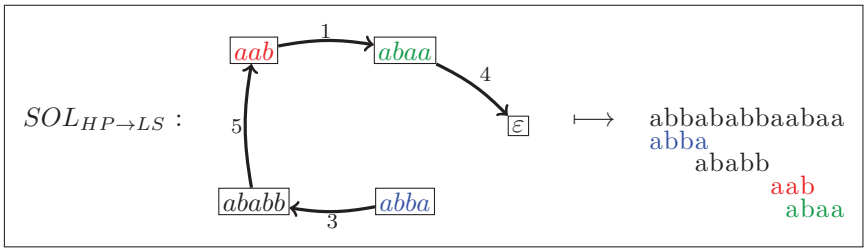


FIGURE 2.20 – Exemple de l'application $sol_{HP\to LS}$ pour $q = \big((abba, ababb), (ababb, aab), (aab, abaa), (abaa, p)\big)$ sur le préfixe graphe de $P = \{aab, abaa, ababb, abba\}$.

Remarque 2.63 Le nœud du graphe des préfixes u_{n+1} est en fait le puits p et comme on a associé p au mot vide, on a alors $pr(u_n, u_{n+1}) = u_n$.

Proposition 2.64 ***Lien entre*** Min-HDP ***et*** SLS Soient $P := \{s_1,\dots,s_n\}$ un ensemble de mots, G le graphe des préfixes de P et p le puits de G.

Soient $q \in OPT_{\text{Min-HDP}}(G)$ et $w \in OPT_{\text{SLS}}(P)$. On a donc que $w \in PSLS(P)$ et donc il existe une permutation de $\{1,\dots,n\}$ telle que $w := P_l(\sigma)$. On a alors :

1. $|q| = |w|$,
2. $sol_{HP\to LS}(q) \in OPT_{\mathsf{SLS}}(P)$,
3. $sol_{LS\to HP}(w).(s_{\sigma(n)}, p) \in OPT_{\mathsf{Min\text{-}HDP}}(G)$.

Preuve En reprenant les définitions, pour $q := ((u_1, u_2), \ldots, (u_n, u_{n+1}))$ on a que

$$\begin{aligned} |sol_{HP\to LS}(q)| &= |\bigoplus_{j=1}^{n} pr(u_j, u_{j+1})| \\ &= |\big((u_1, u_2), \ldots, (u_n, u_{n+1})\big)| \\ &= |q| \end{aligned}$$

et que

$$\begin{aligned} |sol_{LS\to HP}(w).(s_{\sigma(n)}, p)| &= |\big((s_{\sigma(1)}, s_{\sigma(2)}), \ldots, (s_{\sigma(n-1)}, s_{\sigma(n)})\big)| + |(s_{\sigma(n)}, p)| \\ &= |\bigoplus_{j=1}^{n-1} pr(s_{\sigma(j)}, s_{\sigma(j+1)})| + |s_{\sigma(n)}| \\ &= |w|. \end{aligned}$$

Comme $w \in OPT_{\mathsf{SLS}}(P)$, on a que $|w| \leq |sol_{HP\to LS}(q)| = |q|$. Comme $q \in OPT_{\mathsf{Min\text{-}HDP}}(G)$, on a que $|q| \leq |sol_{LS\to HP}(w).(s_{\sigma(n)}, p)| = |w|$. On a donc que $|q| = |w|$ et donc que $sol_{HP\to LS}(q) \in OPT_{\mathsf{SLS}}(P)$ et que $sol_{LS\to HP}(w).(s_{\sigma(n)}, p) \in OPT_{\mathsf{Min\text{-}HDP}}(G)$. ■

Conclusion de la sous-section Grâce à la proposition 2.64, on a montré formellement qu'il y a équivalence à trouver une solution optimale de SLS d'un ensemble de mots et à trouver une solution optimale de Min-HDP dans le graphe des préfixes de cet ensemble de mots.

2.5.5 Liens entre le graphe des préfixes et SCS

On peut faire de même un lien entre Min-HDC et SCS (voir Proposition 2.65) et entre Min-HDCC et SCCS (voir Proposition 2.66). Pour ce faire, on va définir le *graphe des préfixes réduit*, c'est le sous-graphe du graphe des préfixes privé du nœud p qui correspond au puits p et des arcs finissant en p.

On va montrer qu'on peut passer d'une superchaîne circulaire d'un ensemble de mot P à un circuit hamiltonien sur le graphe des préfixes réduit de P, et inversement. Grâce à la proposition 2.25, on ne va regarder que les éléments de $PSCS(P)$. Soit $\langle w\rangle \in PSCS(P)$, il existe alors une permutation circulaire de $\{1, \ldots, n\}$ telle que $\langle w\rangle := Circular(P, \sigma)$. On pose alors

$$sol_{CS\to HC}(\langle w\rangle) := ((s_1, s_{\sigma^1(1)}), \ldots, (s_{\sigma^n(1)}, s_1)).$$

Dans l'autre sens, soit $c := ((u_1, u_2), \ldots, (u_n, u_1))$ un circuit hamiltonien dans le graphe des préfixes réduit de P, on pose alors

$$sol_{HC\to CS}(c) := \langle \bigoplus_{j=1}^{n} pr(u_j, u_{j+_n 1})\rangle$$

On peut alors donner le pendant de la proposition 2.64 pour le problème SCS.

Proposition 2.65 ***Lien entre Min-HDC et SCS*** Soient $P := \{s_1, \ldots, s_n\}$ un ensemble de mots et G^- le graphe des préfixes réduit de P.

Soient $c \in OPT_{Min-HDC}(G^-)$ et $\langle w\rangle \in OPT_{SCS}(P)$, on a alors

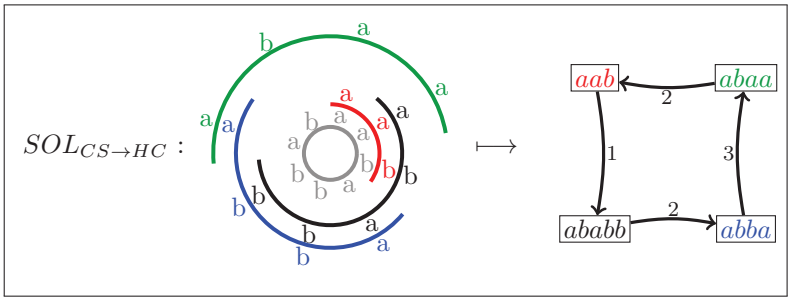


FIGURE 2.21 – Exemple de l'application $sol_{CS\to HC}$ pour $\langle w\rangle = \langle aababbab\rangle$ sur $P = \{aab, abaa, ababb, abba\}$.

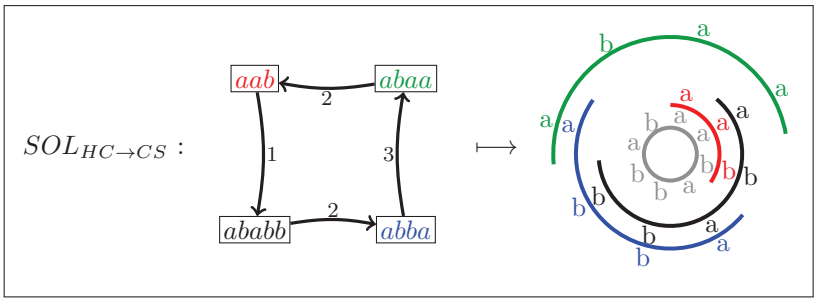


FIGURE 2.22 – Exemple de l'application $sol_{HC\to CS}$ pour $c = \big((aab, ababb), (ababb, abba), (abba, abaa), (abaa, aab)\big)$ sur le graphe des préfixes réduit de $P = \{aab, abaa, ababb, abba\}$.

— $|c| = |\langle w\rangle|$,
— $sol_{HC\to CS}(c) \in OPT_{SCS}(P)$,
— $sol_{CS\to HC}(\langle w\rangle) \in OPT_{Min-HDC}(G^-)$.

Conclusion de la sous-section Grâce à la proposition 2.65, on a montré formellement qu'il y a équivalence à trouver une solution optimale de SCS d'un ensemble de mots et à trouver une solution optimale de Min-HDC sur le graphe des préfixes réduit de cet ensemble de mots. En généralisant le lien connu entre SLS et Min-HDP, on a montré que l'on pouvait utiliser une version réduit du graphe des préfixes pour trouver un lien entre SCS et Min-HDC. On peut donc utiliser des algorithmes de Min-HDC pour trouver des solutions de SCS.

2.5.6 Liens entre le graphe des préfixes et SCCS

On va montrer qu'on peut passer d'une couverture de mots circulaire d'un ensemble de mot P à une couverture cyclique sur le graphe des préfixes réduit de P, et inversement.

Soit $W \in PSCCS(P)$, on a alors que $W = \{\langle w_1\rangle, \ldots, \langle w_m\rangle\}$ avec pour tout i entre 1 et m, $\langle w_i\rangle \in PSCCS(P_i)$ où $\{P_1, \ldots, P_m\}$ est la partition de P donnée par la permutation σ telle que $W = CC(P, \sigma)$. On pose alors

$$sol_{CCS\to HCC}(W) := \{sol_{CS\to HC}(\langle w_1\rangle), \ldots, sol_{CS\to HC}(\langle w_m\rangle)\}.$$

Dans l'autre sens, soit $C := \{c_1, \ldots, c_m\}$ une couverture cyclique du graphe des préfixes

réduit de P, on pose alors

$$sol_{HCC \to CCS}(C) := \{sol_{HC \to CS}(c_1), \ldots, sol_{HC \to CS}(c_m)\}.$$

Proposition 2.66 ***Lien entre Min-HDCC et SCCS*** Soient $P := \{s_1, \ldots, s_n\}$ un ensemble de mots et G^- le graphe des préfixes réduit de P.

Soient $C \in OPT_{Min-HDCC}(G^-)$ et $W \in OPT_{SCCS}(P)$, on a alors

— $||C|| = ||W||$,

— $sol_{HCC \to CCS}(C) \in OPT_{SCCS}(P)$,

— $sol_{CCS \to HCC}(W) \in OPT_{Min-HDCC}(G^-)$.

Conclusion de la sous-section Grâce à la proposition 2.66, on a montré formellement qu'il y a équivalence à trouver une solution optimale de SCCS d'un ensemble de mots et à trouver une solution optimale de Min-HDCC sur le graphe des préfixes de cet ensemble de mots. On a montré notamment que l'on pouvait utiliser une version réduite du graphe des préfixes pour trouver un lien entre SCCS et Min-HDCC. On peut donc utiliser des algorithmes de Min-HDCC pour trouver des solutions de SCCS. Même si ce résultat était déjà connu comme par exemple dans Turner [90], on a apporté ici un formalisme qui nous sera utile dans les chapitres suivants.

Conclusion du chapitre On a montré formellement dans ce chapitre que pour les problèmes des superchaînes, on pouvait se restreindre à prendre en entrée un ensemble de mots factor-free et que cette modalité nous permettait de redéfinir les problèmes classiques en terme de permutation. En effet, toutes les solutions optimales sont de la forme d'une superchaîne obtenue par la fusion des mots dans l'ordre donné par une permutation. Cette redéfinition des problèmes classiques permet de restreindre le nombre de solutions possibles en un nombre fini dépendant de la taille de l'ensemble de mots en entrée. On a montré ce résultat pour les deux problèmes classiques SLS et SCCS. On a, de plus, défini trois problèmes qui vérifient aussi ce résultat et qui nous paraissent importants : les problèmes SCS, SLCS et SMCS. Pour chacun de ces problèmes nous avons montré leur complexité.

Deuxième partie

Compression et algorithme glouton

Chapitre 3

Variante compression

Pour les problèmes des superchaînes, il existe deux mesures d'optimisation naturelles : soit on peut regarder la longueur de la superchaîne, soit on peut regarder la différence entre la norme de l'ensemble de mots en entrée et la longueur de la superchaîne, que l'on appelle la compression de la superchaîne. Avec la première mesure, on veut alors trouver une des plus petites superchaînes associée à un ensemble de mots, alors qu'avec la deuxième mesure, on veut trouver une des superchaînes qui ait la plus grande compression. On a donc à faire à deux problèmes d'optimisation différents, un problème de minimisation (SLS voir Problème 2.1) et un problème de maximisation (SLS_c voir Problème 3.1). Il est connu que les solutions optimales de SLS et de SLS_c sont exactement les mêmes [90]. D'un autre coté, on sait qu'en terme d'approximation, on perd la correspondance. En effet, pour une superchaîne d'un ensemble de mots, la superchaîne peut avoir une bonne compression mais pas une bonne longueur.

Dans ce chapitre, on va présenter un encadrement de la longueur d'une superchaîne en fonction de sa compression, et inversement. Ce résultat, va nous permettre de faire le lien entre la longueur et la compression d'une superchaîne. Ensuite, nous allons appliquer cet encadrement au cas des problèmes des superchaînes où on a un ensemble de mots de taille fixé en instance (on pourra trouver ce résultat dans (IX)). Enfin, nous allons présenter une structure bien connue, le graphe des chevauchements qui est le pendant du graphe des préfixes pour la compression.

Sommaire

3.1 Variante compression pour les problèmes des superchaînes

Comme on l'a vu dans la section 1.8, pour un problème $\mathcal{P}$, on peut définir la version compression $\mathcal{P}_c$ du problème $\mathcal{P}$. Par exemple, on peut définir la version compression du problème SLS.

Problèmme 3.1 SLS_c Soit P un ensemble de mots, on cherche la superchaîne linéaire w de P qui maximise $||P|| - |w|$.

On peut alors définir de même les problèmes SCS_c et SCCS_c.

D'après les propositions 2.12, 2.25 et 2.37, on peut se restreindre aux problèmes SLS (Problème 2.15), SCS (Problème 2.26) et SCCS (Problème 2.38) où l'on ne considère que les superchaînes issues ou générées par des permutations.

Comme pour un problème d'optimisation $\mathcal{P}$, les solutions optimales de $\mathcal{P}$ sont aussi des solutions optimales de $\mathcal{P}_c$ et inversement (voir proposition 3.7), Alors pour les problèmes des superchaînes, les solutions optimales de la version compression du problème sont des superchaînes issues ou générées par des permutations. On a par exemple la proposition suivante pour le problème SLS que l'on peut généraliser à SCS et à SCCS.

Proposition 3.2 Soit P un ensemble de mots factor-free. On a que $OPT_{\mathsf{SLS}_c}(P) \subseteq PSLS(P)$.

Preuve Pour un ensemble de mots P, on a que $OPT_{\mathsf{SLS}}(P) \subseteq PSLS(P)$. Comme les solutions optimales de SLS et de SLS_c se correspondent, on a que $OPT_{\mathsf{SLS}}(P) = OPT_{\mathsf{SLS}_c}(P)$. On a donc que $OPT_{\mathsf{SLS}_c}(P) \subseteq PSLS(P)$. ■

On peut alors redéfinir les problèmes SLS_c, SCS_c et $SCCS_c$.

Problèmme 3.3 SLS_c ***version 2*** Soit P un ensemble de mots factor-free, on cherche la superchaîne linéaire $P_l(\sigma)$ de P, où σ est une permutation de $\{1, \dots, |P|\}$, qui maximise

$$||P|| - |P_l(\sigma)| = \sum_{i=1}^{|P|-1} |ov(s_{\sigma(i)}, s_{\sigma(i+1)})|.$$

Problèmme 3.4 SCS_c ***version 2*** Soit P un ensemble de mots factor-free, on cherche la superchaîne circulaire $Circular(P, \sigma)$ de P, où σ est une permutation circulaire de $\{1, \dots, |P|\}$, qui maximise

$$||P|| - |Circular(P, \sigma)| = \sum_{i=1}^{|P|} |ov(s_{\sigma^i(1)}, s_{\sigma^{i+_{|P|}1}(1)})|.$$

Problèmme 3.5 SCCS_c ***version 2*** Soit P un ensemble de mots factor-free, on cherche la couverture circulaire de mots $CC(P, \sigma) = \{Circular(P_1, \sigma_1), \dots, Circular(P_m, \sigma_m)\}$

de P, où σ est une permutation de $\{1, \ldots, |P|\}$, qui maximise

$$||P|| - |CC(P, \sigma)| = \sum_{j=1}^{m} \big(||P_j|| - |Circular(P_j, \sigma_j)|\big).$$

Remarque 3.6 Pour les superchaînes linéaires de $PSLS(P)$, les superchaînes circulaires de $PSCS(P)$ et les couvertures circulaires de mots de $PSCCS(P)$, la mesure maximisée par la variante compression correspond à la somme des chevauchements maximaux.

Il est important dans les définitions 3.3, 3.4 et 3.5 de SLS_c, SCS_c et SCCS_c que l'ensemble de mots en entrée soit factor-free et que la solution soit issue d'une permutation. Cette mention est bien souvent oubliée dans la littérature quand ces restrictions de SLS_c, SCS_c et SCCS_c sont données.

Conclusion de la section Grâce aux liens entre un problème d'optimisation et sa version compression, on a pu redéfinir en terme de permutation les problèmes SLS_c (Problème 3.3), SCS_c (Problème 3.4) et SCCS_c (Problème 3.5) et donner une définition plus précise des mesures à maximiser.

3.2 Lien entre problème d'optimisation et variante compression

On va donc s'intéresser par la suite aux problèmes SLS_c (Problème 3.3), SCS_c (Problème 3.4) et SCCS_c (Problème 3.5 et plus particulièrement au lien qu'il va y avoir avec les problèmes SLS (Problème 2.15), SCS (Problème 2.26) et SCCS (Problème 2.38). Pour réaliser cela, on va regarder dans le cas général le lien entre un problème d'optimisation $\mathcal{P}$ et sa version compression $\mathcal{P}_c$. On va commencer par voir le lien entre les solutions optimales de $\mathcal{P}$ et les solutions optimales de $\mathcal{P}_c$. On va ensuite regarder, pour un algorithme, le lien entre son ratio d'approximation sur $\mathcal{P}$ et son ratio d'approximation sur $\mathcal{P}_c$.

3.2.1 Solutions optimales

Pour commencer, on va se demander quel est le lien entre les solutions optimales de chacun : on aboutit au résultat que l'ensemble des solutions optimales de $\mathcal{P}$ est identique à l'ensemble des solutions optimales de $\mathcal{P}_c$.

Proposition 3.7 Soit S une entrée du problème $\mathcal{P}$. On a que $OPT_{\mathcal{P}_c}(S) = OPT_{\mathcal{P}}(S)$.

Preuve SoiT $w \in OPT_{\mathcal{P}}(S)$. Montrons que $w \in OPT_{\mathcal{P}_c}(S)$. Supposons que $w \notin OPT_{\mathcal{P}_c}(S)$, alors il existe $w_c \in OPT_{\mathcal{P}_c}(S)$ tel que $||S|| - |w_c| > ||S|| - |w|$. On a donc que $|w_c| < |w|$, ce qui est impossible car w est optimal. On a donc que $OPT_{\mathcal{P}}(S) \subseteq OPT_{\mathcal{P}_c}(S)$. On montre de même que $OPT_{\mathcal{P}_c}(S) \subseteq OPT_{\mathcal{P}}(S)$. ■

3.2.2 Ratios d'approximation

On peut ensuite se demander, comment un algorithme d'approximation sur $\mathcal{P}_c$, verrait son approximation varier sur $\mathcal{P}$ et inversement. Dans cette partie, on va prendre $\mathcal{P}$ un problème de minimisation. On aura alors que $\mathcal{P}_c$ est un problème de maximisation. Comme $(\mathcal{P}_c)_c = \mathcal{P}$, on peut généraliser ce résultat au problème $\mathcal{P}'$ de maximisation en prenant $\mathcal{P}' = \mathcal{P}_c$.

Soit $\mathcal{A}$ un algorithme sur $\mathcal{P}$. On pose $super(\mathcal{A})$ le ratio d'approximation optimal de $\mathcal{A}$ sur $\mathcal{P}$ et $comp(\mathcal{A})$ le ratio d'approximation optimal de $\mathcal{A}$ sur $\mathcal{P}_c$. De plus, on pose

$$super(S, \mathcal{A}) = max(\{\frac{|w_{\mathcal{A}}|}{|w_{opt}|} \mid w_{\mathcal{A}} \text{ une solution de } \mathcal{A} \text{ sur } S \text{ et } w_{opt} \text{ une solution optimale de } \mathcal{P} \text{ sur } S\})$$

et

$$comp(S, \mathcal{A}) = min(\{\frac{||S||-|w_{\mathcal{A}}|}{||S||-|w_{opt}|} \mid w_{\mathcal{A}} \text{ une solution de } \mathcal{A} \text{ sur } S \text{ et } w_{opt} \text{ une solution optimale de } \mathcal{P} \text{ sur } S\}).$$

On a alors que

$$super(\mathcal{A}) = max(\{super(S, \mathcal{A}) \mid S \text{ entrée de } \mathcal{P}\})$$

et que

$$comp(\mathcal{A}) = min(\{comp(S, \mathcal{A}) \mid S \text{ entrée de } \mathcal{P}\}).$$

On va supposer de plus que la taille de toute solution optimale w_{opt} de $\mathcal{P}$ pour une entrée S est plus petite que la taille de S, *i.e.* $|w_{opt}| \leq ||S||$. Enfin, on ne va prendre que des algorithmes d'approximation $\mathcal{A}$ qui donnent des solutions de taille plus petite que la taille de S, *i.e.* $|w_{\mathcal{A}}| \leq ||S||$.

Remarque 3.8 Dans le cas des problèmes des superchaînes, un algorithme $\mathcal{A}$ qui donne des solutions $w_{\mathcal{A}}$ issues de permutations pour un ensemble de mots S vérifie toujours la propriété $|w_{\mathcal{A}}| \leq ||S||$. En effet, la restriction aux permutations permet de ne considérer que ces cas là. On n'a pas de solution où l'on ajoute des caractères non imposés par les mots de départ.

On va définir dans la suite deux types d'encadrement : le premier en fonction d'un encadrement de la taille de la solution optimale sur la norme de l'instance (Théorème 3.13) et le deuxième en fonction d'un encadrement de la taille d'une solution de l'algorithme sur la norme de l'instance (Théorème 3.18).

3.2.2.1 Encadrement à l'aide de la taille d'une solution optimale

On va commencer par donner un lien entre les ratios d'approximation optimaux de $\mathcal{P}$ et $\mathcal{P}_c$ (respectivement de $\mathcal{P}_c$ et $\mathcal{P}$) en fonction d'une borne inférieure de $\frac{|w_{opt}|}{||S||}$ (respectivement d'une borne inférieure de $\frac{||S||-|w_{opt}|}{||S||}$), où w_{opt} est une solution optimale pour le problème $\mathcal{P}$ de minimisation sur une entrée S de ce problème.

Nous démontrons alors la proposition suivante qui nous permet d'avoir une borne supérieure pour $super(S, \mathcal{A})$ en fonction d'une borne inférieure de $\frac{|w_{opt}|}{||S||}$.

Proposition 3.9 Soient $\mathcal{P}$ un problème de minimisation, S une entrée du problème $\mathcal{P}$, $\mathcal{A}$ un algorithme sur $\mathcal{P}$ et $w_{opt} \in OPT_{\mathcal{P}}(S)$. Pour tout réel γ tel que $\gamma \leq \frac{|w_{opt}|}{||S||}$, on a :

$$super(S, \mathcal{A}) \leq \frac{(\gamma - 1) \times comp(\mathcal{A}) + 1}{\gamma}.$$

Preuve Soient $\alpha = \frac{(\gamma-1)\times comp(\mathcal{A})+1}{\gamma}$ et la fonction $f : x \mapsto \frac{(x-1)\times comp(\mathcal{A})+1}{x}$. Sa dérivée est $f' : x \mapsto \frac{comp(\mathcal{A})-1}{x^2}$ qui est négative car $0 < comp(\mathcal{A}) \leq 1$. f est donc une fonction strictement décroissante et comme $\gamma < 1$, on a $\alpha = f(\gamma) > f(1) = 1$. On obtient que $\gamma = \frac{1-comp(\mathcal{A})}{\alpha-comp(\mathcal{A})}$. Il s'ensuit que :

$$\begin{array}{rrcl} & \gamma \times ||S|| & \leq & |w_{opt}| \\ \Leftrightarrow & \frac{1-comp(\mathcal{A})}{\alpha-comp(\mathcal{A})} \times ||S|| & \leq & |w_{opt}| \\ \Leftrightarrow & (1 - comp(\mathcal{A})) \times ||S|| & \leq & (\alpha - comp(\mathcal{A})) \times |w_{opt}| \\ \Leftrightarrow & comp(\mathcal{A}) \times |w_{opt}| + (1 - comp(\mathcal{A})) \times ||S|| & \leq & \alpha \times |w_{opt}|. \end{array}$$

Or $\mathcal{A}$ a un ratio d'approximation de $comp(\mathcal{A})$ pour $\mathcal{P}_c$. Pour $w_{\mathcal{A}}$ une solution de l'algorithme $\mathcal{A}$, on a donc

$$\begin{array}{rrcl} & ||S|| - w_{\mathcal{A}} & \geq & comp(\mathcal{A}) \times (||S|| - w_{opt}) \\ \Rightarrow & |w_{\mathcal{A}}| & \leq & comp(\mathcal{A}) \times |w_{opt}| + (1 - comp(\mathcal{A})) \times ||S|| \leq \alpha \times |w_{opt}|. \end{array}$$

Comme le problème $\mathcal{P}$ est un problème de minimisation, on obtient que

$$super(S, \mathcal{A}) = min(\{\beta \mid \beta \geq \frac{|w_{\mathcal{A}}|}{|w_{opt}|}\})$$

et donc que $super(S, \mathcal{A}) \leq \alpha = \frac{(\gamma-1)\times comp(\mathcal{A})+1}{\gamma}$. ■

Remarque 3.10 La fraction $\frac{|w_{opt}|}{||S||}$ est indépendante d'un algorithme sur le problème $\mathcal{P}$. En effet, cette fraction correspond à la « capacité » à l'ensemble S à pouvoir être bien résolu par le problème $\mathcal{P}$. Dans le cas par exemple où $|w_{opt}| = ||S||$, on a alors que $\gamma = 1$ et donc $\frac{(\gamma-1)\times comp(\mathcal{A})+1}{\gamma} = 1$ et $super(S, \mathcal{A}) \leq 1$, c'est-à-dire que tout algorithme $\mathcal{A}$ est exact pour cet ensemble de mots S. En effet si $|w_{opt}| = ||S||$, on en déduit que pour tout algorithme $\mathcal{A}$, $|w_{\mathcal{A}}| = |w_{opt}|$, car on a supposé que $|w_{\mathcal{A}}| \leq ||S||$.

Nous démontrons ensuite la proposition suivante qui nous permet d'avoir une borne inférieure pour $comp(S, \mathcal{A})$ en fonction d'une borne inférieure de $\frac{||S||-|w_{opt}|}{||S||}$. Cette proposition est le symétrique de la proposition 3.9 où *super* et *comp* ont échangé leurs rôles.

Proposition 3.11 Soient $\mathcal{P}$ un problème de minimisation, S une entrée du problème $\mathcal{P}$, $\mathcal{A}$ un algorithme sur $\mathcal{P}$ et $w_{opt} \in OPT_{\mathcal{P}}(S)$. Pour tout réel γ tel que $\gamma \leq \frac{||S||-|w_{opt}|}{||S||}$, on a :

$$comp(S, \mathcal{A}) \geq \frac{(\gamma - 1) \times super(\mathcal{A}) + 1}{\gamma}.$$

Preuve Soient $\alpha = \frac{(\gamma-1)\times super(\mathcal{A})+1}{\gamma}$ et la fonction $f : x \mapsto \frac{(x-1)\times super(\mathcal{A})+1}{x}$. Sa dérivée est $f' : x \mapsto \frac{comp(\mathcal{A})-1}{x^2}$ qui est positive car $1 \leq super(\mathcal{A})$. f est donc une fonction strictement croissante et comme $\gamma < 1$, on a $\alpha = f(\gamma) < f(1) = 1$. On obtient que $\gamma = \frac{super(\mathcal{A})-1}{super(\mathcal{A})-\alpha}$. Il s'ensuit que :

$$\begin{array}{rrcl} & \gamma \times ||S|| & \leq & ||S|| - |w_{opt}| \\ \Leftrightarrow & \frac{super(\mathcal{A})-1}{super(\mathcal{A})-\alpha} \times ||S|| & \leq & ||S|| - |w_{opt}| \\ \Leftrightarrow & (super(\mathcal{A}) - 1) \times ||S|| & \leq & (super(\mathcal{A}) - \alpha) \times (||S|| - |w_{opt}|) \\ \Leftrightarrow & ||S|| - super(A) \times |w_{opt}| & \geq & \alpha \times (||S|| - |w_{opt}|). \end{array}$$

Or $\mathcal{A}$ a un ratio d'approximation de $super(\mathcal{A})$ pour $\mathcal{P}$, soit $w_{\mathcal{A}}$ une solution de l'algorithme $\mathcal{A}$, on a donc

$$\begin{array}{rrcl} & w_{\mathcal{A}} & \geq & super(\mathcal{A}) \times w_{opt} \\ \Rightarrow & ||S|| - |w_{\mathcal{A}}| & \geq & ||S|| - super(\mathcal{A}) \times w_{opt} \geq \alpha \times (||S|| - |w_{opt}|). \end{array}$$

Comme le problème $\mathcal{P}$ est un problème de minimisation, on a alors que

$$comp(S, \mathcal{A}) = max(\{\beta \mid \beta \leq \frac{||S|| - |w_{\mathcal{A}}|}{||S|| - |w_{opt}|}\})$$

et donc que $comp(S, \mathcal{A}) \geq \alpha = \frac{(\gamma-1)\times super(\mathcal{A})+1}{\gamma}$. ■

Remarque 3.12 Les preuves des propositions 3.9 et 3.11 sont très similaires. On donne néanmoins les deux preuves pour comprendre les points délicats.

On peut alors encadrer $super(\mathcal{A})$ à l'aide de $comp(\mathcal{A})$ et inversement.

Théorème 3.13 Soient $\mathcal{P}$ un problème de minimisation et $\mathcal{A}$ un algorithme sur $\mathcal{P}$.

1. Pour tous réels γ et γ' tels que pour toute entrée S du problème $\mathcal{P}$, $\gamma \leq \frac{|w_{opt}|}{||S||} \leq \gamma'$ où $w_{opt} \in OPT_{\mathcal{P}}(S)$, on a :
$$\frac{(\gamma' - 1) \times comp(\mathcal{A}) + 1}{\gamma'} \leq super(\mathcal{A}) \leq \frac{(\gamma - 1) \times comp(\mathcal{A}) + 1}{\gamma}.$$
2. Pour tous réels γ et γ' tels que pour toute entrée S du problème $\mathcal{P}$, $\gamma \leq \frac{||S||-|w_{opt}|}{||S||} \leq \gamma'$ où $w_{opt} \in OPT_{\mathcal{P}}(S)$, on a :
$$\frac{(\gamma - 1) \times super(\mathcal{A}) + 1}{\gamma} \leq comp(\mathcal{A}) \leq \frac{(\gamma' - 1) \times super(\mathcal{A}) + 1}{\gamma'}.$$

Preuve On ne va montrer ici que le point 1. car la preuve du point 2. est identique. D'après la proposition 3.9, on a que pour tout réel γ tel que $\gamma \leq \frac{|w_{opt}|}{||S||}$, on a :

$$super(S, \mathcal{A}) \leq \frac{(\gamma - 1) \times comp(\mathcal{A}) + 1}{\gamma}.$$

On a donc pour tout réel γ tel que pour toute entrée S du problème $\mathcal{P}$, $\gamma \leq \frac{|w_{opt}|}{||S||}$ et on a donc que

$$super(\mathcal{A}) \leq \frac{(\gamma - 1) \times comp(\mathcal{A}) + 1}{\gamma}.$$

En effet, si pour tout x d'un ensemble E, il existe un réel a tel que $x \leq a$, on a alors que $max(\{x \in E\}) \leq a$.

D'après la proposition 3.11, on a que pour tout réel γ'' tel que $\gamma'' \leq \frac{||S||-|w_{opt}|}{||S||}$, on a :

$$comp(S, \mathcal{A}) \geq \frac{(\gamma'' - 1) \times super(\mathcal{A}) + 1}{\gamma''}.$$

On a donc pour tout réel γ'' tel que pour toute entrée S du problème $\mathcal{P}$, $\gamma'' \leq \frac{||S||-|w_{opt}|}{||S||}$ et on a donc que

$$comp(\mathcal{A}) \geq \frac{(\gamma'' - 1) \times super(\mathcal{A}) + 1}{\gamma''}.$$

En effet si pour tout x d'un ensemble E, il existe un réel a tel que $x \geq a$, on a alors que $min(\{x \in E\}) \geq a$. En posant $\gamma' = 1 - \gamma''$ on a que

$$\begin{array}{rrcl} & comp(\mathcal{A}) & \geq & \frac{(\gamma''-1)\times super(\mathcal{A})+1}{\gamma''} \\ \Leftrightarrow & comp(\mathcal{A}) & \geq & \frac{1-\gamma'\times super(\mathcal{A})}{1-\gamma'} \\ \Leftrightarrow & comp(\mathcal{A}) \times (1-\gamma') & \geq & 1 - \gamma' \times super(\mathcal{A}) \\ \Leftrightarrow & \frac{(\gamma'-1)\times comp(\mathcal{A})+1}{\gamma'} & \leq & super(\mathcal{A}). \end{array}$$

Remarque 3.14 Le théorème 3.13 nous donne un encadrement de $super(\mathcal{A})$ en fonction de $comp(\mathcal{A})$. Si on arrive à trouver un encadrement de $\frac{|w_{opt}|}{||S||}$ pour tout S, on peut alors encadrer $super(\mathcal{A})$. Soient γ et ϵ tels que $\gamma \leq \frac{|w_{opt}|}{||S||} \leq \gamma + \epsilon$, on a alors que

$$\frac{(\gamma - 1) \times comp(\mathcal{A}) + 1}{\gamma} - \frac{(\gamma + \epsilon - 1) \times comp(\mathcal{A}) + 1}{\gamma + \epsilon} = \frac{\epsilon \times (1 - comp(\mathcal{A}))}{\gamma^2 + \gamma \times \epsilon}.$$

On a alors que si on trouve γ et ϵ tels que $\gamma \leq \frac{|w_{opt}|}{||S||} \leq \gamma + \epsilon$ avec ϵ aussi petit que l'on veut, on obtient exactement le ratio d'approximation optimal $super(\mathcal{A})$: en effet on a que $\lim_{\epsilon \to 0} \frac{\epsilon \times (1-comp(\mathcal{A}))}{\gamma^2+\gamma\times\epsilon} = 0$.

3.2.2.2 Encadrement à l'aide de la taille d'une solution de l'algorithme

Dans la sous-section précédente, on a utilisé un encadrement de $\frac{|w_{opt}|}{||S||}$ (respectivement $\frac{||S||-|w_{opt}|}{||S||}$) pour tout S pour encadrer $super(\mathcal{A})$ à l'aide de $comp(\mathcal{A})$ (respectivement $comp(\mathcal{A})$ à l'aide de $super(\mathcal{A})$). On va ici utiliser un encadrement de $\frac{|w_{\mathcal{A}}|}{||S||}$ (respectivement $\frac{||S||-|w_{\mathcal{A}}|}{||S||}$) pour tout S pour encadrer $super(\mathcal{A})$ à l'aide de $comp(\mathcal{A})$ (respectivement $comp(\mathcal{A})$ à l'aide de $super(\mathcal{A})$).

Comme les preuves sont très proches de celles de la sous-section précédente, on ne va donner que la preuve de la première proposition. Cette proposition nous donne une borne

supérieure pour $super(S, \mathcal{A})$ en fonction d'une borne inférieure de $\frac{|w_{\mathcal{A}}|}{||S||}$:

Proposition 3.15 Soient S un ensemble de mots, $\mathcal{A}$ un algorithme d'approximation sur $\mathcal{P}$, $w_{\mathcal{A}}$ une solution de l'algorithme $\mathcal{A}$ et $w_{opt} \in OPT_{\mathcal{P}}(S)$.

Pour tout réel β tel que $1 - comp(\mathcal{A}) < \beta < 1$ et $\beta \leq \frac{|w_{\mathcal{A}}|}{||S||}$, on a :

$$super(S, \mathcal{A}) \leq \frac{\beta \times comp(\mathcal{A})}{\beta + comp(\mathcal{A}) - 1}.$$

Preuve On pose $\alpha = \frac{\beta \times comp(\mathcal{A})}{\beta + comp(\mathcal{A}) - 1}$. Comme $\beta > 1 - comp(\mathcal{A})$, on a que $\alpha > 0$ et donc $\alpha - comp(\mathcal{A}) = \frac{comp(\mathcal{A}) \times (1 - comp(\mathcal{A}))}{\beta + comp(\mathcal{A}) - 1} > 0$ et donc $\beta = \frac{\alpha \times (1 - comp(\mathcal{A}))}{\alpha - comp(\mathcal{A})}$.

On peut déterminer un peu mieux la borne inférieure de α :

$$\begin{array}{rrcl}
 & \beta & < & 1 \\
\Leftrightarrow & \frac{\alpha \times (1 - comp(\mathcal{A}))}{\alpha - comp(\mathcal{A})} & < & 1 \\
\Leftrightarrow & \alpha \times (1 - comp(\mathcal{A})) & < & \alpha - comp(\mathcal{A}) \\
\Leftrightarrow & \alpha \times comp(\mathcal{A}) & > & comp(\mathcal{A}) \\
\Leftrightarrow & \alpha & > & 1.
\end{array}$$

On a alors :

$$\begin{array}{rrcl}
 & \beta \times ||S|| & \leq & |w_{\mathcal{A}}| \\
\Leftrightarrow & \frac{\alpha \times (1 - comp(\mathcal{A}))}{\alpha - comp(\mathcal{A})} \times ||S|| & \leq & |w_{\mathcal{A}}| \\
\Leftrightarrow & \alpha \times (1 - comp(\mathcal{A})) \times ||S|| & \leq & (\alpha - comp(\mathcal{A})) \times |w_{\mathcal{A}}| \\
\Leftrightarrow & (\alpha \times comp(\mathcal{A}) - \alpha) \times ||S|| + \alpha \times |w_{\mathcal{A}}| & \geq & comp(\mathcal{A}) \times |w_{\mathcal{A}}| \\
\Leftrightarrow & (\alpha - \frac{\alpha}{comp(\mathcal{A})}) \times ||S|| + \frac{\alpha}{comp(\mathcal{A})} \times |w_{\mathcal{A}}| & \geq & |w_{\mathcal{A}}|.
\end{array}$$

Or $\mathcal{A}$ a un ratio d'approximation de $comp(\mathcal{A})$ pour $\mathcal{P}_c$, on a donc

$$\begin{array}{rrcl}
 & comp(\mathcal{A}) \times (||S|| - |w_{opt}|) & \leq & ||S|| - |w_{\mathcal{A}}| \\
\Leftrightarrow & ||S|| - |w_{opt}| & \leq & \frac{1}{comp(\mathcal{A})} \times (||S|| - |w_{\mathcal{A}}|) \\
\Leftrightarrow & \alpha \times (||S|| - |w_{opt}|) & \leq & \frac{\alpha}{comp(\mathcal{A})} \times (||S|| - |w_{\mathcal{A}}|) \\
\Leftrightarrow & \alpha \times |w_{opt}| & \geq & (\alpha - \frac{\alpha}{comp(\mathcal{A})}) \times ||S|| + \frac{\alpha}{comp(\mathcal{A})} \times |w_{\mathcal{A}}| \geq |w_{\mathcal{A}}| \\
\Rightarrow & \alpha & \geq & \frac{|w_{\mathcal{A}}|}{|w_{opt}|}.
\end{array}$$

Comme le problème $\mathcal{P}$ est un problème de minimisation, on a alors que

$$super(S, \mathcal{A}) = min(\{\lambda \mid \lambda \geq \frac{|w_{\mathcal{A}}|}{|w_{opt}|}\})$$

et donc que $super(S, \mathcal{A}) \leq \alpha = \frac{\beta \times comp(\mathcal{A})}{\beta + comp(\mathcal{A}) - 1}$. ■

Remarque 3.16 La proposition 3.15 nous permet à partir d'une solution d'un algorithme d'approximation sur $\mathcal{P}$ de donner une borne supérieure pour la distance entre cette solution et une solution optimale de $\mathcal{P}$ en fonction du ratio d'approximation optimal de $\mathcal{P}_c$.

On a de plus un résultat très intéressant qui nous dit que plus un algorithme avec un ratio d'approximation optimal constant sur $\mathcal{P}_c$ va donner des solutions proches en taille de $||S||$, c'est-à-dire pour β qui tend vers 1, plus les solutions optimales vont se rapprocher des solutions de l'algorithme. En effet, on a que $\lim_{\beta \to 1} \frac{\beta \times comp(\mathcal{A})}{\beta + comp(\mathcal{A}) - 1} = 1$.

Par des arguments similaires, on obtient alors la proposition 3.17 ainsi que le théorème 3.18.

Proposition 3.17 Soient S un ensemble de mots, $\mathcal{A}$ un algorithme d'approximation sur $\mathcal{P}$ et $w_{\mathcal{A}}$ une solution de l'algorithme $\mathcal{A}$.
Pour tout réel β tel que $\beta \leq \frac{||S|| - |w_{\mathcal{A}}|}{||S||}$, on a :

$$comp(S, \mathcal{A}) \geq \frac{\beta \times super(\mathcal{A})}{\beta + super(\mathcal{A}) - 1}.$$

Théorème 3.18 Soient $\mathcal{P}$ un problème de minimisation et $\mathcal{A}$ un algorithme sur $\mathcal{P}$.

1. Pour tous réels β et β' tels que $1 - comp(\mathcal{A}) < \beta < 1$ et pour toute entrée S du problème $\mathcal{P}$, $\beta \leq \frac{|w_{\mathcal{A}}|}{||S||} \leq \beta'$ où $w_{\mathcal{A}}$ est une solution de l'algorithme $\mathcal{A}$, on a :

$$\frac{\beta' \times comp(\mathcal{A})}{\beta' + comp(\mathcal{A}) - 1} \leq super(\mathcal{A}) \leq \frac{\beta \times comp(\mathcal{A})}{\beta + comp(\mathcal{A}) - 1}.$$

2. Pour tous réels β et β' tels que $0 < \beta' < comp(\mathcal{A})$ et pour toute entrée S du problème $\mathcal{P}$, $\beta \leq \frac{||S|| - |w_{\mathcal{A}}|}{||S||} \leq \beta'$ où $w_{\mathcal{A}}$ est une solution de l'algorithme $\mathcal{A}$, on a :

$$\frac{\beta \times super(\mathcal{A})}{\beta + super(\mathcal{A}) - 1} \leq comp(\mathcal{A}) \leq \frac{\beta' \times super(\mathcal{A})}{\beta' + super(\mathcal{A}) - 1}.$$

Conclusion de la section Dans cette section, on a montré qu'à l'aide d'un encadrement de $\frac{|w_{opt}|}{||S||}$ ou de $\frac{|w_{\mathcal{A}}|}{||S||}$, on pouvait encadrer la ratio d'approximation optimal d'un algorithme sur un problème $\mathcal{P}$ en fonction du ratio d'approximation optimal de ce même algorithme pour le problème $\mathcal{P}_c$. Ces deux types d'encadrements sont importants, en effet le théorème 3.13 nous donne un encadrement indépendant de l'algorithme choisi (on a juste besoin du ratio d'approximation de ce problème pour la version compression) alors que le théorème 3.18 nous donne un encadrement que l'on peut calculer avec une solution d'un algorithme.

3.3 Taille de mots de l'instance fixée

On a vu dans la section 3.2 qu'à partir d'un encadrement de $\frac{|w_{opt}|}{||S||}$, on peut utiliser le ratio d'approximation optimal d'un problème $\mathcal{P}_c$ pour encadrer le ratio d'approximation optimal du problème $\mathcal{P}$. On va regarder ici les problèmes SLS et SCS où tous les mots de l'instance ont la même longueur qui est fixée. Pour un entier r strictement plus grand que 1, on définit alors formellement ces deux problèmes (Problèmes 3.19 et 3.20).

Problèmme 3.19 r-SLS Soit P un ensemble de mots où chaque mot est de taille r, on cherche une superchaîne linéaire de P .

Problèmme 3.20 r-SCS Soit P un ensemble de mots où chaque mot est de taille r, on cherche une superchaîne circulaire de P de longueur minimale.

Crochemore et al. [16] montre que le problème r-SLS est dans **P** dans le cas où $r = 2$ (plus exactement il existe un algorithme en $O(||P||^2)$ où P est l'instance, c'est-à-dire un ensemble de mots). De plus, par Gallant et al. [25], on a une preuve que le problème r-SLS est **NP-Difficile** dans le cas où $r \geq 3$.

On va ici s'intéresser au ratio d'approximation optimal de l'algorithme glouton (que l'on va définir dans la section 4.1) sur r-SLS et r-SCS. On pourra trouver une partie de ces résultats dans (VI).

Remarque 3.21 On ne va pas s'intéresser au problème r-SCCS, car comme d'après le théorème 4.23, l'algorithme glouton est optimal pour SCCS, il est alors optimal pour r-SCCS (grâce à la proposition 3.22).

Les problèmes r-SLS et r-SCS ne sont pas des problèmes avec une contrainte sur la sortie mais avec une contrainte sur l'entrée, on a donc que s'il existe par exemple un algorithme de ratio d'approximation optimal α sur SLS, ce même algorithme donne un ratio d'approximation optimal plus petit que α sur r-SLS. En d'autres termes, on a la proposition suivante :

Proposition 3.22 Soient $\mathcal{P}$ un problème de minimisation, $\mathcal{A}$ un algorithme sur $\mathcal{P}$, $\mathcal{P}'$ le problème de minimisation $\mathcal{P}$ où l'on a restreint l'ensemble des entrées et $\mathcal{A}'$ le même algorithme que $\mathcal{A}$ mais sur $\mathcal{P}'$. On a alors que :

- $super(\mathcal{A}') \leq super(\mathcal{A})$,
- $comp(\mathcal{A}') \geq comp(\mathcal{A})$.

La proposition 3.22 nous donne des inégalités. En effet, il est possible que les contre exemples pour le problème $\mathcal{P}$ ne marchent plus pour le problème $\mathcal{P}'$ car ils ne sont plus de la forme souhaitée.

On va ici reprendre le résultat du théorème 3.13 que l'on va appliquer à notre problème. Pour la suite de cette section, on va prendre r un entier strictement plus grand que 1 et P un ensemble de mots de taille r. On peut alors borner la taille d'une superchaîne linéaire et la taille d'une superchaîne circulaire de P.

Proposition 3.23 Soient $P = \{s_1, \ldots, s_n\}$ un ensemble de mots de taille r et $t \in PSLS(P) \cup PSCS(P)$. On a

$$r + |P| - 1 \leq |t| \leq n \times |P|$$

Preuve Comme l'instance P est un sous-ensemble de Σ^r, nous avons que $||P|| = r \times n$. Comme chaque mot de P est différent, on aboutit à ce que chaque mot diffère d'un autre par au moins un symbole et que le chevauchement d'un mot sur un autre est d'au plus $r - 1$ symboles. ■

De cet encadrement de la taille d'une superchaîne (linéaire ou circulaire), on peut alors en dériver le théorème suivant.

Théorème 3.24 Soient r un entier tel que $r > 1$ et P un ensemble de mots de taille r.

1. Pour tout algorithme $\mathcal{A}_L$ de r-SLS :

$$1 \leq super(\mathcal{A}_L) \leq r - (r-1) \times comp(\mathcal{A}_L).$$

2. Pour tout algorithme $\mathcal{A}_C$ de r-SCS :

$$1 \leq super(\mathcal{A}_C) \leq r - (r-1) \times comp(\mathcal{A}_C).$$

Preuve On va prouver le point 1., la preuve du point 2. est similaire. D'après la proposition 3.23, on a que pour une superchaîne linéaire optimale $|w_{opt}|$ de r-SLS, $r + |P| - 1 \leq |w_{opt}| \leq r \times |P|$, ce qui implique que

$$\begin{array}{rccccc} & \frac{r+|P|-1}{||P||} & \leq & \frac{|w_{opt}|}{||P||} & \leq & \frac{r\times|P|}{||P||} \\ \Leftrightarrow & \frac{r+|P|-1}{r\times|P|} & \leq & \frac{|w_{opt}|}{||P||} & \leq & \frac{r\times|P|}{r\times|P|} \\ \Rightarrow & \frac{|P|}{r\times|P|} & \leq & \frac{|w_{opt}|}{||P||} & \leq & 1 \\ \Leftrightarrow & \frac{1}{r} & \leq & \frac{|w_{opt}|}{||P||} & \leq & 1 \end{array}$$

En utilisant le théorème 3.13 avec $\gamma = \frac{1}{r}$ et $\gamma' = 1$, on obtient

$$\begin{array}{rccccc} & \frac{1+(1-1)\times comp(\mathcal{A}_L)}{1} & \leq & super(\mathcal{A}_L) & \leq & \frac{1+(\frac{1}{r}-1)\times comp(\mathcal{A}_L)}{\frac{1}{r}} \\ \Leftrightarrow & 1 & \leq & super(\mathcal{A}_L) & \leq & r \times (1 + (\frac{1-r}{r}) \times comp(\mathcal{A}_L)) \\ \Leftrightarrow & 1 & \leq & super(\mathcal{A}_L) & \leq & r - (r-1) \times comp(\mathcal{A}_L). \end{array}$$

D'après Tarhio et al. [87], on sait que l'algorithme glouton pour SLS_c a un ratio d'approximation optimal de $\frac{1}{2}$. Dans la section 4.3, on va donner le théorème 4.23 qui nous donne notamment le ratio d'approximation optimal de l'algorithme glouton pour SCS_c qui est de $\frac{1}{2}$ (on a aussi une nouvelle preuve grâce à ce théorème que SLS_c a un ratio d'approximation optimal de $\frac{1}{2}$). On peut alors montrer que ce ratio d'approximation optimal pour l'algorithme glouton ne change ni pour r-SLS_c ni pour r-SCS_c.

Proposition 3.25 Les ratios d'approximation optimaux de l'algorithme glouton pour r-SLS_c et pour r-SCS_c sont $\frac{1}{2}$.

Preuve Grâce à la proposition 3.22, on a donc que $comp(glouton_{\text{r-SLS}}) \geq comp(glouton_{\mathsf{SLS}}) = \frac{1}{2}$ et que $comp(glouton_{\text{r-SCS}}) \geq comp(glouton_{\mathsf{SCS}}) = \frac{1}{2}$. D'après l'exemple 3.26, on a que $comp(glouton_{\text{r-SLS}}) \leq \frac{1}{2}$ et que $comp(glouton_{\text{r-SCS}}) \leq \frac{1}{2}$. ■

Exemple 3.26 Soit $P = \{ab^{r-1}, b^r, b^{r-1}c\}$ une instance de r-SLS et r-SCS.

— On a alors qu'une superchaîne linéaire optimale est ab^rc. On a de plus que $ab^{r-1}cb^r$ est une solution de l'algorithme glouton pour r-SLS.

On a alors que $super(glouton_{\text{r-SLS}}) \geq \frac{|ab^{r-1}cb^r|}{|ab^rc|} = \frac{2r+1}{r+2}$ et $comp(glouton_{\text{r-SLS}}) \leq \frac{||P||-|ab^{r-1}cb^r|}{||P||-|ab^rc|} = \frac{3\times r-2r+1}{3\times r-r+2} = \frac{r+1}{2r+2} = \frac{1}{2}$.

— On a alors qu'une superchaîne circulaire optimale est $\langle ab^rc\rangle$. On a de plus que $\langle ab^{r-1}cb^r\rangle$ est une solution de l'algorithme glouton pour r-SCS. On a alors que $super(glouton_{\text{r-SCS}}) \geq \frac{|\langle ab^{r-1}cb^r\rangle|}{|\langle ab^rc\rangle|} = \frac{2r+1}{r+2}$ et $comp(glouton_{\text{r-SCS}}) \leq \frac{||P||-|\langle ab^{r-1}cb^r\rangle|}{||P||-|\langle ab^rc\rangle|} = \frac{3\times r-2r+1}{3\times r-r+2} = \frac{r+1}{2r+2} = \frac{1}{2}$.

Pour l'algorithme glouton, la proposition 3.25 nous dit que les ratios d'approximation optimaux sont les mêmes entre SLS_c et r-SLS_c et entre SCS_c et r-SCS_c. On peut alors se demander si ce constat est identique entre SLS et r-SLS et entre SCS et r-SCS. La proposition suivante donne une borne inférieure pour r-SLS et r-SCS.

Proposition 3.27 Les ratios d'approximation optimaux de l'algorithme glouton pour r-SLS et pour r-SCS sont d'au moins $2 - \frac{1}{r}$.

Preuve Le théorème 3.24 donne une borne supérieure pour les ratios d'approximation de l'algorithme glouton pour r-SLS ainsi que r-SCS. Pour obtenir la borne inférieure désirée, on va produire une instance telle que $\frac{|w_{glouton}|}{|w_{opt}|} = 2 - \frac{1}{r}$ et cela pour $w_{glouton}$ une solution de l'algorithme glouton et w_{opt} une solution optimale pour r-SLS.

Considérons $P := \{a_1a_2^{r-1}, a_2^r, a_2^{r-1}a_3, a_2a_3^{r-1}, \ldots, a_{m-1}^r, a_{m-1}^{r-1}a_m\}$ sur l'alphabet $\Sigma = \{a_1, a_2, \ldots, a_m\}$. Pour r-SLS, dans le pire cas, on arrive à la solution gloutonne qui est

$$w_{glouton} = a_1a_2^{r-1}a_3^{r-1}\ldots a_{m-1}^{r-1}a_ma_2^ra_3^r\ldots a_{m-1}^r$$

quand une solution optimale est $w_{opt} = a_1a_2^ra_3^r\ldots a_{m-1}^ra_m$. On a alors

$$\begin{array}{rcl} \frac{|w_{glouton}|}{|w_{opt}|} & = & \frac{2+(r-1)(m-2)+r(m-2)}{2+r(m-2)} \\ & = & 2 - \frac{m}{2+r(m-2)} \\ & \xrightarrow[m\to\infty]{} & 2 - \frac{1}{r}. \end{array}$$

On prenant le même ensemble de mots P,

$$\langle w_{opt}\rangle = \langle a_1a_2^ra_3^r\ldots a_{m-1}^ra_m\rangle$$

et

$$\langle w_{glouton}\rangle = \langle a_1a_2^{r-1}a_3^{r-1}\ldots a_{m-1}^{r-1}a_ma_2^ra_3^r\ldots a_{m-1}^r\rangle,$$

on obtient exactement la même borne. ■

Remarque 3.28 La proposition 3.27, nous donne une nouvelle borne inférieure pour les ratios d'approximation optimaux de r-SLS et r-SCS. En effet, la borne inférieure classique de 2 pour SLS et SCS ne peut pas être appliquée, car l'exemple utilisé pour la borne inférieure fait croître la grandeur des mots, ce qui n'est pas possible ni dans r-SLS ni dans r-SCS.

Grâce à la proposition 3.27 et au théorème 3.24, et en utilisant les ratios d'approximation optimaux de l'algorithme glouton pour r-SLS_c et r-SCS_c donnés par la

proposition 3.25, on obtient une borne des ratios d'approximation optimaux de l'algorithme glouton pour r-SLS et r-SCS.

Théorème 3.29 Les ratios d'approximation optimaux de l'algorithme glouton peuvent être encadrés par :

— $2 - \frac{1}{r} \leq super(glouton_{r\text{-SLS}}) \leq \frac{r+1}{2}$,

— $2 - \frac{1}{r} \leq super(glouton_{r\text{-SCS}}) \leq \frac{r+1}{2}$.

	r	2	3	4	5	6
borne inférieure	$2 - \frac{1}{r}$	3/2	5/3	7/4	9/5	11/6
borne supérieure	$\frac{r+1}{2}$	3/2	2	5/2	3	7/2

TABLE 3.1 – Borne d'approximation de l'algorithme glouton pour r-SLS et r-SCS pour $r < 7$. Cela donne un ratio d'approximation optimal de 2 pour 3-SLS et 3-SCS.

Remarque 3.30 Même si le problème 2-SLS n'est pas **NP-difficile**, l'algorithme glouton n'est pas exact pour ce problème. En effet, il a un ratio d'approximation optimal de $\frac{3}{2}$. On a ici un exemple que la complexité d'un problème ne dépend pas de l'optimalité de l'algorithme glouton sur ce problème.

La table 3.1 montre les bornes données par le théorème 3.29 pour des petites valeurs de r. On peut remarquer que l'algorithme glouton obtient un ratio de $\frac{3}{2}$ pour 2-SLS et 2-SCS et un ratio de 2 pour 3-SLS et 3-SCS. Ceci résout la conjecture glouton pour 3-SLS ainsi que pour 3-SCS. Comme la précédente borne connue du ratio d'approximation optimal de l'algorithme glouton pour r-SLS est $\frac{7}{2}$ [44], le théorème 3.29 améliore cette borne pour toute valeur de r plus petite ou égale à 5. Notons que d'autres algorithmes d'approximation (qui sont plus complexes que l'algorithme glouton) donnent un meilleur ratio d'approximation pour des petits r (voir une table récente apparue dans [30]).

Remarque 3.31 Il existe de plus une preuve pour montrer que le ratio d'approximation pour 4-SLS est d'au plus 2 [50]. Cette preuve, qui est malheureusement un peu floue, ne peut pas être généralisée ni pour 3-SLS ni pour 5-SLS.

`Q : Les ratios d'approximation optimaux des problèmes` r-SLS `et de` r-SCS `sont-ils de` $2 - \frac{1}{r}$ `?`

Conclusion de la section On s'est intéressé dans cette section à l'algorithme glouton sur les problèmes de la plus petite superchaîne et de la plus petite superchaîne circulaire où on a en entrée un ensemble de mots avec des mots de même longueur. On a donné les ratios d'approximation optimaux pour r-SLS$_c$ et pour r-SCS$_c$ ainsi qu'un encadrement pour les ratios d'approximation optimaux pour r-SLS et pour r-SCS. Ces nouvelles bornes pour les ratios d'approximation optimaux améliorent les précédents résultats sur ces problèmes pour les valeurs petites de r. On a de plus mis en évidence que l'on pourrait mettre à jour la conjecture de 2 pour SLS à $2 - \frac{1}{r}$ pour r-SLS.

3.4 Graphe de chevauchements

On va s'intéresser dans cette section au lien qu'il peut y avoir entre les problèmes des graphes et les problèmes SLS_c, SCS_c et SCCS_c. On va commencer par définir le pendant du graphe des préfixes : le graphe des chevauchements. On peut retrouver dans Turner [90] et dans Kaplan [43] qu'il existe un lien entre le problème du chemin hamiltonien sur le graphe des chevauchements et le problème de la plus petite superchaîne. On va redonner ici formellement ce lien ainsi que les liens similaires qui existent entre les problèmes sur les graphes et les problèmes SCS_c et SCCS_c.

3.4.1 Définition du graphe des chevauchements

Dans la section 3.1, on a pu donner une définition de ces problèmes en termes de permutation et on a vu que pour les superchaînes linéaires de $PSLS(P)$, les superchaînes circulaires de $PSCS(P)$ et les couvertures circulaires de mots de $PSCCS(P)$, la mesure maximisée par la variante compression correspond à la somme des chevauchements maximaux.

À l'image du graphe des préfixes, on va définir un graphe qui va nous permettre de faire les liens entre des problèmes sur les graphes Max-HDP, Max-HDC et Max-HDCC et les problèmes sur les mots SLS_c, SCS_c et SCCS_c. Ce graphe est déjà connu dans la littérature, un des premiers à définir et utiliser le graphe des chevauchements est Gallant dans sa thèse [24]. On peut se référer au livre de Gusfield [35] pour de plus amples explications.

Définition 3.32 ***Graphe des chevauchements (voir Figure 3.1)*** Soit P un ensemble de mots. Le *graphe des chevauchements* de P est le graphe orienté complet pondéré sur ses arcs $G = (V, A, w)$ où :

$$V := P$$

$$A := (P \times P)$$

$$\begin{array}{rccc} w : & A & \to & \mathbb{N} \\ & (u,v) & \mapsto & |ov(u,v)| \end{array}$$

et où $ov(u, v)$ est le chevauchement maximal de u vers v.

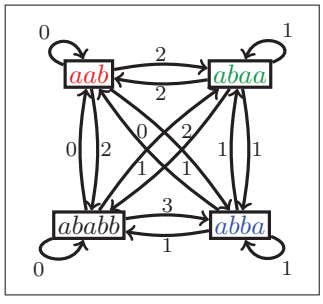


FIGURE 3.1 – Exemple de graphe des chevauchements pour $P = \{aab, abaa, ababb, abba\}$.

Comme le graphe des chevauchements d'un ensemble de mots P est orienté et complet, il a donc $|P|^2$ arcs, et donc la taille du graphe des chevauchements de P est linéaire en

$|P|^2$. Comme on ne peut pas espérer que la complexité de l'algorithme soit meilleure que le maximum de la taille de l'entrée plus la taille de la sortie, le meilleur algorithme de construction du graphe des chevauchements de P est en $O(||P|| + |P|^2)$: on peut retrouver un algorithme de cette complexité dans [34]. Enfin, comme le graphe des chevauchements a $|P|$ nœuds, la longueur d'un chemin hamiltonien est linéaire en $|P|$. Les longueurs d'un circuit hamiltonien et d'une couverture cyclique sont aussi linéaires en $|P|$.

Proposition 3.33 Soit P un ensemble de mots. Le graphe des chevauchements de P satisfait l'inégalité supérieure de Monge.

Preuve Soient u, v, x et y quatre mots tels que $|ov(u,v)| \geq |ov(u,x)|$ et $|ov(u,v)| \geq |ov(y,v)|$. On veut montrer que

$$|ov(u,v)| + |ov(y,x)| \geq |ov(u,x)| + |ov(y,v)|$$

On a alors deux cas :

— Si $|ov(u,v)| \geq |ov(u,x)| + |ov(y,v)|$, on a alors l'inégalité car $|ov(y,x)| \geq 0$.

— Si $|ov(u,v)| < |ov(u,x)| + |ov(y,v)|$ (voir Figure 3.2), on note alors α la sous-chaîne commune à $ov(u,v)$, $ov(u,x)$ et $ov(y,v)$. On a alors que α est un chevauchement de y vers x et donc $|\alpha| \leq |ov(y,x)|$. On a donc

$$\begin{aligned} |ov(u,v)| + |ov(y,x)| &\geq |ov(u,v)| + |\alpha| \\ &= |ov(u,v)| + |ov(u,x)| + |ov(y,v)| - |ov(u,v)| \\ &= |ov(u,x)| + |ov(y,v)| \end{aligned}$$

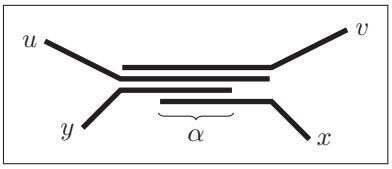


FIGURE 3.2 – Représentation de la preuve de la proposition 3.33.

Attention, comme dans le cas du graphe des préfixes, l'inégalité supérieure de Monge ne suffit pas pour définir le graphe des chevauchements. En effet, un graphe qui satisfait cette condition n'est pas forcément un graphe des chevauchements issu d'un ensemble de mots (voir Figure 3.3). Dans la littérature, ce problème de savoir si un graphe est un graphe des chevauchements est un sujet assez étudié : on peut se référer notamment à Braga et al. [13], Czuma et al. [19] et Gevezes et al. [28].

3.4.2 Liens entre les solutions optimales

Dans les sous-sections 2.5.4, 2.5.5 et 2.5.6, on a vu le lien entre les problèmes sur les graphes appliqués sur le graphe des préfixes et les problèmes des superchaînes SLS, SCS et SCCS. On peut faire de même le lien entre les problèmes sur le graphe des chevauchements et les problèmes de compression des superchaînes SLS_c, SCS_c et SCCS_c.

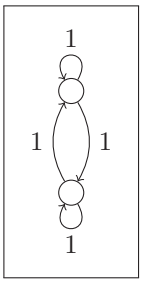


FIGURE 3.3 – Exemple de graphe qui satisfait l'inégalité supérieure de Monge mais qui n'est pas un graphe des chevauchements.

On va commencer par faire le lien entre les solutions optimales de Max-HDP sur le graphe des chevauchements et SLS_c. Ce lien est déjà connu, en effet on peut le retrouver notamment dans Tarhio et al. [87]. La proposition 3.34 formalise ce lien.

Proposition 3.34 ***Lien entre*** Max-HDP ***et*** SLS_c Soient $P := \{s_1, \ldots, s_n\}$ un ensemble de mots et G le graphe des chevauchements de P.

Soient $p := ((u_1, u_2), \ldots, (u_{n-1}, u_n)) \in OPT_{\mathsf{Max\text{-}HDP}}(G)$ et $w \in OPT_{\mathsf{SLS}_c}(P)$, on a alors :

- $|p| = comp(w)$,
- $sol_{HP \to LS}(p).u_n \in OPT_{\mathsf{SLS}_c}(P)$,
- $sol_{LS \to HP}(w) \in OPT_{\mathsf{Max\text{-}HDP}}(G)$.

On peut de même faire un lien entre les solutions optimales de Max-HDC sur le graphe des chevauchements et SCS_c.

Proposition 3.35 ***Lien entre*** Max-HDC ***et*** SCS_c Soient $P := \{s_1, \ldots, s_n\}$ un ensemble de mots et G le graphe des chevauchements de P.

Soient $c \in OPT_{\mathsf{Max\text{-}HDC}}(G)$ et $\langle w \rangle \in OPT_{\mathsf{SCS}_c}(P)$, on a alors :

- $|c| = comp(w)$,
- $sol_{HC \to CS}(c) \in OPT_{\mathsf{SCS}_c}(P)$,
- $sol_{CS \to HC}(\langle w \rangle) \in OPT_{\mathsf{Max\text{-}HDC}}(G)$.

On peut enfin faire un lien entre les solutions optimales de Max-HDCC sur le graphe des chevauchements et SCCS_c.

Proposition 3.36 ***Lien entre*** Max-HDCC ***et*** SCCS_c Soient $P := \{s_1, \ldots, s_n\}$ un ensemble de mots et G le graphe des chevauchements de P.

Soient $C \in OPT_{\mathsf{Max\text{-}HDCC}}(G)$ et $W \in OPT_{\mathsf{SCCS}_c}(P)$, on a alors :

- $||C|| = comp(W)$,
- $sol_{HCC \to CCS}(C) \in OPT_{\mathsf{SCCS}_c}(P)$,
- $sol_{CCS \to HCC}(W) \in OPT_{\mathsf{Max\text{-}HDCC}}(G)$.

Remarque 3.37 Grâce aux propositions 2.64, 3.7 et 3.34, pour un ensemble de mots P, G_p le graphe des préfixes de P et G_c le graphe des chevauchements de P, il existe alors une bijection entre $OPT_{\text{Max-HDP}}(G_c)$ et $OPT_{\text{Min-HDP}}(G_p)$ où p est le nœud puits du graphe des préfixes et f est cette bijection telle que

$$f : q = ((u_1, u_2), \ldots, (u_{n-1}, u_n)) \mapsto ((u_1, u_2), \ldots, (u_{n-1}, u_n), (u_n, p)).$$

On a de même l'inégalité pour Max-HDC : $OPT_{\text{Max-HDC}}(G_c) = OPT_{\text{Min-HDC}}(G_p^-)$ et l'inégalité pour Max-HDCC : $OPT_{\text{Max-HDCC}}(G_c) = OPT_{\text{Min-HDCC}}(G_p^-)$ où G_p^- est le graphe des préfixes réduit de P.

3.4.3 Liens entre les approximations

En plus du lien entre les solutions optimales du problème Max-HDP sur le graphe des chevauchements et les solutions optimales de SLS_c, on peut à partir d'un algorithme avec un ratio d'approximation optimal α sur Max-HDP, trouver un algorithme avec un ratio d'approximation optimal plus grand que α sur SLS_c. Ce résultat est déjà connu, dans l'article de Kaplan et al. [43], on a notamment une utilisation d'un algorithme sur Max-HDP pour trouver une solution sur SLS_c. On va formaliser ce résultat :

Proposition 3.38 Soient $P := \{s_1, \ldots, s_n\}$ un ensemble de mots et G le graphe des chevauchements de P. Soient $\mathcal{A}_\mathcal{P}$ un algorithme pour Max-HDP de ratio d'approximation optimal α_P et ${\mathcal{A}_\mathcal{P}}^*$ l'algorithme pour SLS_c qui donne la superchaîne $sol_{HP\to LS}(p).u_n$ de P pour tout chemin hamiltonien $p := ((u_1, u_2), \ldots, (u_{n-1}, u_n))$ sur le graphe des chevauchements de P donné par $\mathcal{A}_\mathcal{P}$. On a alors

$$comp({\mathcal{A}_\mathcal{P}}^*) \geq \alpha_P.$$

On peut faire de même entre Max-HDC et SCS_c.

Proposition 3.39 Soient $P := \{s_1, \ldots, s_n\}$ un ensemble de mots et G le graphe des chevauchements de P. Soient $\mathcal{A}_\mathcal{C}$ un algorithme pour Max-HDC$_d$e ratio d'approximation optimal α_C et ${\mathcal{A}_\mathcal{C}}^*$ l'algorithme pour SCS_c qui donne la superchaîne circulaire $sol_{HC\to CS}(c)$ de P pour tout circuit hamiltonien c sur le graphe des chevauchements de P donné par $\mathcal{A}_\mathcal{C}$. On a alors

$$comp({\mathcal{A}_\mathcal{C}}^*) \geq \alpha_C.$$

Enfin, on peut faire de même entre Max-HDCC et SCCS_c.

Proposition 3.40 Soient $P := \{s_1, \ldots, s_n\}$ un ensemble de mots et G le graphe des chevauchements de P. Soient $\mathcal{A}_{\mathcal{CC}}$ un algorithme pour Max-HDCC de ratio d'approximation optimal α_{CC} et ${\mathcal{A}_{\mathcal{CC}}}^*$ l'algorithme pour SCCS_c qui donne la superchaîne circulaire $sol_{HCC\to CCS}(C)$ de P pour toute couverture cyclique C sur le graphe des chevauchements de P donnée par $\mathcal{A}_{\mathcal{CC}}$. On a alors

$$comp({\mathcal{A}_{\mathcal{CC}}}^*) \geq \alpha_{CC}.$$

Conclusion de la section Dans cette sous-section, on a montré que l'on peut généraliser le lien connu entre le problème Max-HDP sur le graphe des chevauchements et le problème SLS_c (en terme de solutions optimales et de ratios d'approximation optimaux) aux liens entre les problèmes Max-HDC et Max-HDCC sur le graphe des chevauchements et les problèmes SCS_c et SCCS_c. Ce lien permet d'utiliser des algorithmes déjà connus sur les graphes pour résoudre les problèmes des superchaînes. On va utiliser dans la sous-section 4.3.2 ces propriétés pour appliquer les algorithmes gloutons sur les graphes aux problèmes des superchaînes.

Conclusion du chapitre On a mis en évidence dans ce chapitre des outils reliant les ratios d'approximation optimaux des version compression des problèmes aux ratios d'approximation optimaux des problèmes. Ce résultat est nouveau et permet en l'appliquant aux problèmes r-SLS et r-SCS d'améliorer leurs bornes des ratios d'approximation optimaux pour $r \leq 5$. L'intérêt de la méthode donnée est qu'elle peut être utilisée sur d'autres variantes de r-SLS et r-SCS comme par exemple pour les variantes avec la multiplicité, le complémentaire renversé ou la contrainte sur les chevauchements que l'on verra dans le chapitre 5.

Chapitre 4

L'algorithme glouton

Dans ce chapitre, on va s'intéresser à l'algorithme glouton pour les problèmes des superchaînes, c'est-à-dire les problèmes SLS, SCS et SCCS. Cet algorithme est un des algorithmes les plus simples et pourtant un des plus étudiés. On trouve les premières mentions de l'algorithme glouton pour le problème SLS dans la thèse de Galland [24] et on l'étudie encore : il existe une conjecture vieille de trente ans [11] qui n'est pas encore prouvée qui dit que le ratio d'approximation de l'algorithme glouton pour SLS est de 2. La plupart des algorithmes actuels pour SLS sont des versions améliorées de l'algorithme glouton, il y a donc un grand intérêt à comprendre l'algorithme glouton pour SLS pour comprendre ces autres algorithmes.

Dans une première partie, on va commencer par donner une définition des algorithmes gloutons classiques et énoncer quelques unes de leurs propriétés. Ensuite, on va définir les systèmes héréditaires qui vont nous permettre de définir l'algorithme glouton d'un système héréditaire. Enfin, on va montrer qu'on va pouvoir, pour chaque problème de superchaînes, définir un système héréditaire qui va nous permettre de montrer que les algorithmes gloutons classiques peuvent être définis comme un algorithme glouton d'un système héréditaire. En faisant cela, on démontrera le ratio d'approximation optimal des algorithmes gloutons pour la variante compression des problèmes des superchaînes, c'est-à-dire pour SLS_c, SCS_c et SCCS_c.

Sommaire

4.1 Les algorithmes gloutons

Dans cette section, on va s'intéresser à l'algorithme glouton classique. Cet algorithme est à la fois un des algorithmes les plus simples dans sa définition et un de ceux que l'on connaît le moins en fin de compte. Il a été défini pour donner une solution approchée au problème SLS, mais peut être facilement généralisé à d'autres problèmes comme par exemple SCS et SCCS. On peut retrouver dans Blum et al. [11] et dans Czuma et al. [19], une ébauche d'algorithme glouton pour SCCS.

Dans une première sous-section, on va donner pour chacun de ces algorithmes, une définition précise. Ensuite, on va voir que l'on peut redéfinir ces problèmes de manière récursive et donner quelques relations et propriétés sur les solutions de ces algorithmes. Enfin, on va présenter et s'intéresser à la conjecture des ratio d'approximation optimaux des algorithmes gloutons.

4.1.1 Définitions des algorithmes gloutons classiques

L'une des premières références à l'algorithme glouton peut être trouvée dans la thèse de Gallant [24]. Il est au départ défini pour résoudre le problème SLS mais ensuite il est regardé dans de nombreuses variantes de ce problème [50, 39, 22]. Au niveau du ratio d'approximation optimal de l'algorithme glouton pour SLS, une première preuve de Blum et al. [11] montre que ce ratio est entre 2 et 4. Une deuxième preuve de Kaplan et al. [44] a montré que ce ratio est inférieur à $\frac{7}{2} = 3,5$. Une conjecture qui n'a pas été encore prouvée est que le ratio d'approximation optimal de l'algorithme glouton pour SLS est de 2 [84, 87, 90, 11].

Contrairement à SLS, le ratio d'approximation optimal de l'algorithme glouton pour sa version compression SLS_c est connu, il est de $\frac{1}{2}$ [87, 90].

D'autres études ont été réalisées sur la complexité, on sait notamment que l'algorithme glouton pour SLS est en analyse lisse en $1 + o(1)$ [56, 76]. De plus on sait qu'asymptotiquement la moyenne de la somme des chevauchements de la solution optimale pour SLS est équivalente (asymptotiquement) à la moyenne de la somme des chevauchements de la solution de l'algorithme glouton pour SLS [23, 99, 2]. Enfin en pratique, le ratio d'approximation moyen de l'algorithme glouton pour SLS est de 1.014 [78, 100, 54].

Au niveau de l'implémentation de l'algorithme glouton pour SLS, Turner [90] propose une implémentation en $O(||P|| + |P| \times log(|P|))$ pour un alphabet petit et en $O(||P|| \times log(||P||))$ pour un alphabet quelconque. Ukkonen [91] améliore ce résultat en proposant une implémentation en $O(||P||)$ pour les petits alphabets et une implémentation en $O(||P|| \times min(log(|P|), log(|\Sigma|)))$ pour les alphabets de taille arbitraire (où Σ est l'alphabet de l'ensemble de mots P). Czuma [18] montre un peu plus tard que trouver les solutions de l'algorithme glouton pour SLS est dans **P-complet**, c'est-à-dire que c'est difficilement parallélisable et donc on ne trouvera sans doute pas mieux qu'une implémentation en $O(||P||)$.

L'idée de l'algorithme glouton est relativement simple, pour un ensemble de mots P, prendre deux mots qui se chevauchent le mieux, les enlever de P et ajouter à la place la fusion de ces deux mots dans P. Ensuite, on recommence avec ce nouveau P en continuant récursivement jusqu'à obtenir un singleton, c'est-à-dire un ensemble qu'avec un mot et ce

mot sera la superchaîne linéaire donnée par l'algortihme glouton. On entend par deux mots qui se chevauchent le mieux, les deux mots qui ont leur chevauchement maximal qui est plus grand que toutes les autres paires de mots, *i.e.* le mot de $\{ov(u,v) \mid u \text{ et } v \in P\}$ de longueur maximale.

On peut alors définir formellement cet algorithme pour le problème SLS que l'on va appeler $Greedy_{LS}$ (voir Algorithme 4.1).

Algorithme 4.1 : Définition de l'algorithme $Greedy_{LS}$ pour SLS.

Entrée : Un ensemble de mots P ;
Sortie : Une superchaîne linéaire w de P ;
1 **while** $|P| > 1$ **do**
2 Soient u et v deux mots différents de P qui ont le plus grand chevauchement maximal ;
3 $P \leftarrow P \cup \{u \oplus v\} \setminus \{u, v\}$;
4 **return** w tel que $P = \{w\}$;

On peut alors appliquer l'algorithme $Greedy_{LS}$ au cas du problème SCS, en modifiant la sortie : on ne veut plus une superchaîne linéaire mais une superchaîne circulaire. On pose alors $Greedy_{CS}$ l'algorithme glouton pour SCS (voir Algorithme 4.2).

Algorithme 4.2 : Définition de l'algorithme $Greedy_{CS}$ pour SCS.

Entrée : Un ensemble de mots P ;
Sortie : Une superchaîne circulaire $\langle pr(w,w)\rangle$ de P ;
1 **while** $|P| > 1$ **do**
2 Soient u et v deux mots différents de P qui ont le plus grand chevauchement maximal ;
3 $P \leftarrow P \cup \{u \oplus v\} \setminus \{u, v\}$;
4 **return** $\langle pr(w,w)\rangle$ tel que $P = \{w\}$;

Enfin, on peut définir l'algorithme glouton, noté $Greedy_{CCS}$, pour problème SCCS. Dans les algorithmes gloutons $Greedy_{LS}$ et $Greedy_{CS}$, on interdit de faire se chevaucher un mot avec lui-même, dans $Greedy_{CCS}$ on va enlever cette interdiction et on obtient l'algorithme 4.3.

4.1.2 Définitions récursives des algorithmes gloutons

Grâce à leurs définitions, on peut voir facilement que les algorithmes gloutons $Greedy_{LS}$, $Greedy_{CS}$ et $Greedy_{CCS}$ sont récursifs. On va alors redéfinir ces trois algorithmes en fonction des deux autres qui seront les étapes de ces algorithmes.

4.1.2.1 Algorithmes de base pour la récursivité

Pour commencer, on va définir les algorithmes qui vont simuler une étape de chaque algorithme glouton. Pour les algorithmes gloutons $Greedy_{LS}$ et $Greedy_{CS}$, on peut remarquer que les étapes sont les mêmes, c'est-à-dire de prendre un ensemble de mots linéaires et de retourner un nouvel ensemble de mots linéaires où l'on a fusionné les deux mots se chevauchant le mieux. On définit alors $Greedy\_step_L$ (voir Algorithme 4.4)

Algorithme 4.3 : Définition de l'algorithme $Greedy_{CCS}$ pour SCCS.

Entrée : Un ensemble de mots P ;
Sortie : Une couverture circulaire de mots F de P ;

```
1 F ← ∅
2 while |P| > 1 do
3     Soient u et v deux mots de P qui ont le plus grand chevauchement maximal ;
4     if u ≠ v then
5         P ← P ∪ {u ⊕ v} \ {u, v};
6     else
7         P ← P \ {u};
8         F ← F ∪ {⟨pr(u, u)⟩};
9 return F;
```

comme l'algorithme correspondant à cette étape. Pour l'algorithme glouton $Greedy_{CCS}$, on va définir l'algorithme $Greedy\_step_C$ (voir Algorithme 4.5) où on autorise en plus les chevauchements d'un mot avec lui-même.

Même si on n'utilisera ici l'algorithme $Greedy\_step_L$ que pour un ensemble de mots linéaires, on le définit comme prenant en entrée un ensemble de mots linéaires et de mots circulaires. En modifiant l'entrée, on ne modifie pas son comportement pour un ensemble de mots strictement linéaires mais on peut mieux le comparer avec l'algorithme $Greedy\_step_C$.

Algorithme 4.4 : Définition de l'algorithme $Greedy\_step_L$.

Entrée : Un ensemble de mots linéaires et de mots circulaires P ;
Sortie : Un ensemble de mots linéaires et de mots circulaires Q ;

```
1 if |{u mot linéaire de P}| ≥ 2 then
2     Soient u et v deux mots linéaires différents de P qui ont le plus grand chevauchement maximal ;
3     Q ← P ∪ {u ⊕ v} \ {u, v};
4     return Q ;
5 else
6     return ∅ ;
```

Algorithme 4.5 : Définition de l'algorithme $Greedy\_step_C$.

Entrée : Un ensemble de mots linéaires et de mots circulaires P ;
Sortie : Un ensemble de mots linéaires et de mots circulaires Q ;

```
1 if |{u mot linéaire de P}| ≥ 1 then
2     Soient u et v deux mots linéaires de P qui ont le plus grand chevauchement maximal ;
3     Q ← P ∪ {u ⊕ v} \ {u, v};
4     return Q ;
5 else
6     return ∅ ;
```

Remarque 4.1 La notation $u \oplus v$ correspond à la fusion de u vers v. On a alors que

$$u \oplus v = \begin{cases} pr(u,v).v & \text{Si } u \neq v \\ \langle pr(u,u) \rangle & \text{Si } u = v \end{cases}$$

On peut alors trouver quelques propriétés intéressantes sur ces deux algorithmes. On peut commencer par remarquer que les algorithmes $Greedy\_step_L$ et $Greedy\_step_C$ ne sont pas déterministes, en effet pour une même entrée, il n'existe pas forcément un unique couple de mots linéaires de l'instance qui ont le plus grand des chevauchements maximaux, et donc on peut avoir plusieurs solutions (voir Exemple 4.2).

Exemple 4.2 Soit $P = \{abb, bbb, bba\}$. On a alors que
- $SOL_{Greedy\_step_L}(P) = \{\{abbb, bba\}, \{abba, bbb\}, \{abb, bbba\}\}$,
- $SOL_{Greedy\_step_C}(P) = \{\{abbb, bba\}, \{abba, bbb\}, \{abb, bbba\}, \{abb, \langle b\rangle, bba\}\}$.

Même si les deux algorithmes $Greedy\_step_L$ et $Greedy\_step_C$ se ressemblent, ils ne donnent pas forcément les même solutions. On peut alors comparer les solutions de $Greedy\_step_L$ et les solutions de $Greedy\_step_C$ pour un ensemble de mots P.

Proposition 4.3 Soient P un ensemble de mots linéaires et de mots circulaires, $P_l \in SOL_{Greedy\_step_L}(P)$ et $P_c \in SOL_{Greedy\_step_C}(P)$. On a alors

$$||P_c|| \leq ||P_l|| \leq ||P||.$$

On obtient alors en soustrayant $||P||$ à l'équation de la proposition 4.3

$$||P|| - ||P_c|| \geq ||P|| - ||P_l||.$$

Cette équation nous dit que l'algorithme $Greedy\_step_C$ va prendre de meilleurs chevauchements que l'algorithme $Greedy\_step_L$, ce qui est normal car $Greedy\_step_C$ regarde tous les chevauchements que regardent $Greedy\_step_L$ plus les chevauchements d'un mot sur lui-même. On peut aussi comparer la différence entre les solutions successives de $Greedy\_step_L$.

Proposition 4.4 Soient P un ensemble de mots linéaires et de mots circulaires, $P_{l,1} \in SOL_{Greedy\_step_L}(P)$ et $P_{l,2} \in SOL_{Greedy\_step_L}(P_{l,1})$. On a alors

$$||P_{l,1}|| - ||P_{l,2}|| \leq ||P|| - ||P_{l,1}||.$$

Et donc plus on va utiliser récursivement l'algorithme $Greedy\_step_L$, moins on va réussir à améliorer la compression de l'ensemble de mots en sortie. On a le même résultat pour $Greedy\_step_C$.

Proposition 4.5 Soient P un ensemble de mots linéaires et de mots circulaires, $P_{c,1} \in SOL_{Greedy\_step_C}(P)$ et $P_{c,2} \in SOL_{Greedy\_step_C}(P_{c,1})$. On a alors

$$||P_{c,1}|| - ||P_{c,2}|| \leq ||P|| - ||P_{c,1}||.$$

Remarque 4.6 Les propositions 4.4 et 4.5 viennent du fait que les algorithmes gloutons choisissent à chaque étape le chevauchement maximum qu'ils peuvent prendre. Ils prennent alors des chevauchements de plus en plus petits.

4.1.2.2 Redéfinitions des algorithmes gloutons

On peut alors donner une autre définition de $Greedy_{LS}$ (voir Algorithme 4.6), $Greedy_{CS}$ (voir Algorithme 4.7) et $Greedy_{CCS}$ (voir Algorithme 4.8). Cette nouvelle définition sera équivalente à la première. De manière générale, la notation $Greedy\_step_L(P)$ correspond à une solution de $Greedy\_step_L$ pour l'ensemble de mots P. De même, $Greedy\_step_C(P)$ correspond à une solution de $Greedy\_step_C$ pour l'ensemble de mots P.

Algorithme 4.6 : Définition récursive de $Greedy_{LS}$.

```
1 while Greedy_step_L(P) ≠ ∅ do
2     P ← Greedy_step_L(P);
3 return w tel que P = {w};
```

Algorithme 4.7 : Définition récursive de $Greedy_{CS}$.

```
1 while Greedy_step_L(P) ≠ ∅ do
2     P ← Greedy_step_L(P);
3 return ⟨w⟩ tel que Greedy_step_C(P) = {⟨w⟩};
```

Algorithme 4.8 : Définition récursive de $Greedy_{CCS}$.

```
1 while Greedy_step_C(P) ≠ ∅ do
2     P ← Greedy_step_C(P);
3 return P;
```

Comme les algorithmes $Greedy\_step_L$ et $Greedy\_step_C$ sont non-déterministes, on a donc que les algorithmes $Greedy_{LS}$, $Greedy_{CS}$ et $Greedy_{CCS}$ sont eux aussi non déterministes.

On peut alors se demander comment comparer les solutions de $Greedy_{LS}$, $Greedy_{CS}$ et $Greedy_{CCS}$. La proposition 4.7 nous donne une comparaison entre les longueurs des solutions.

Proposition 4.7 Soient P un ensemble de mots linéaires, $w_l \in SOL_{Greedy_{LS}}(P)$, $\langle w_c \rangle \in SOL_{Greedy_{CS}}(P)$ et $C \in SOL_{Greedy_{CCS}}(P)$. On a alors

$$||C|| \leq |\langle w_c \rangle| \leq |w_l|.$$

Pour une meilleure compréhension des algorithmes gloutons $Greedy_{LS}$, $Greedy_{CS}$ et $Greedy_{CCS}$, on va construire le graphe de choix d'un ensemble de mots P.

Définition 4.8 ***Graphe de choix*** Soit P un ensemble de mots. Le *graphe de choix* de P, est le graphe orienté sans cycle $G := (V, A)$ tel que :

$$\begin{aligned} V &= V_L \cup V_C \cup V_{CC} \\ A &= \{(P', Q) \in V \times V \mid Q \in SOL_{Greedy\_step_L}(P') \text{ ou } Q \in SOL_{Greedy\_step_C}(P')\}, \end{aligned}$$

où

$$\begin{aligned} V_L &= \{Q \mid Q \neq \emptyset \text{ et } \exists P' \in V_L \text{ tel que } Q \in SOL_{Greedy\_step_L}(P')\} \\ V_C &= \{Q \mid \exists P' \in V_L \text{ tel que } |P'| = 1 \text{ et } Q \in SOL_{Greedy\_step_C}(P')\} \\ V_{CC} &= \{Q \mid Q \neq \emptyset \text{ et } \exists P' \in V_{CC} \text{ tel que } Q \in SOL_{Greedy\_step_C}(P')\}. \end{aligned}$$

On a alors la proposition suivante, illustrée dans l'exemple 4.10 et dans la figure 4.1.

Proposition 4.9 Soient P ensemble de mots, $w_l \in SOL_{Greedy_{LS}}(P)$, $\langle w_c \rangle \in SOL_{Greedy_{CS}}(P)$ et $C \in SOL_{Greedy_{CCS}}(P)$. On a que $\{w_l\}$, $\{\langle w_c \rangle\}$ et C sont des nœuds du graphe des choix de P.

Exemple 4.10 Soit $P = \{aab, abba, abaa, ababb\}$. La figure 4.1 représente le graphe de choix de P où l'on a représenté sur chaque figure les choix qui ont été faits par un des algorithmes gloutons. Pour réduire la taille du graphe, on a associé à chaque mot de P un numéro et on a utilisé une propriété que l'on verra dans la partie 4.3.2, qui est que toutes les solutions des algorithmes gloutons sont issues de permutations. On a représenté localement ces superchaînes par des rectangles pour les superchaînes circulaires et par des rectangles arrondis pour les superchaînes linéaires. Par exemple le nœud [31](42)

correspond à l'ensemble $\{\langle pr(s_3 \oplus s_1, s_3 \oplus s_1)\rangle, s_4 \oplus s_2\} = \{\langle aba \rangle, ababba\}$.

4.1.3 Conjecture sur le ratio d'approximation de l'algorithme glouton

Il existe une conjecture disant que le ratio d'approximation optimal de l'algorithme glouton pour SLS est de 2. Nous savons actuellement que ce ratio est entre 2 et $\frac{7}{2}$, mais aucune preuve n'a permis de trouver une borne supérieure plus petite ni un contre-exemple pour trouver une borne supérieure plus grande malgré les études qui ont été réalisées [51].

4.1.3.1 Difficulté de trouver une borne inférieure

Nous allons essayer de donner des arguments qui prouvent que la recherche d'un contre-exemple est plus difficile qu'il n'y parait et est souvent altérée par notre manière de la chercher. Le but ici n'est pas de donner des arguments pour montrer que la conjecture est fausse, mais de donner des arguments sur la difficulté du problème.

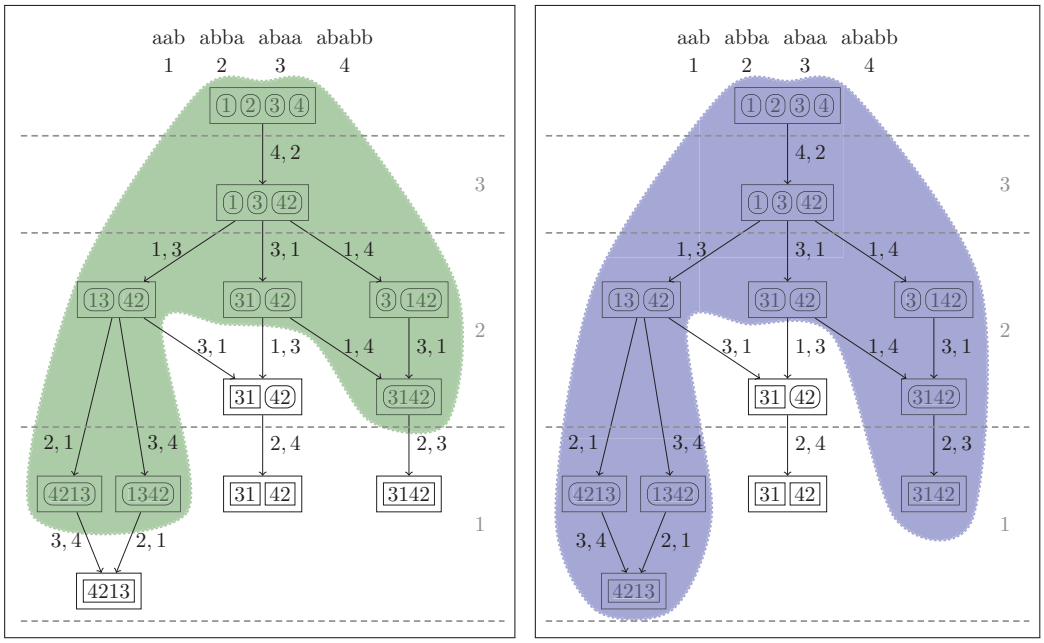
Pour cela, imaginons que la conjecture est fausse et que le ratio d'approximation optimal de l'algorithme glouton pour SLS est de $2 + k$ avec k un réel entre 0 et $\frac{3}{2}$. Plociennik [76] prouve qu'en cas d'existence d'instance ne vérifiant pas la conjecture gloutonne, le nombre de ces instances serait exponentiellement petit.

Prenons P un ensemble de mots tel que $super(P, Greedy_{LS}) > 2$. On peut alors construire le graphe des chevauchements G_P pour cet ensemble de mots P. On a vu dans la partie 3.2 le lien entre le problème Max-HDP sur le graphe des chevauchements et le problème SLS. Trouver directement cet ensemble de mots est souvent difficile, dans la plupart des cas, on essaye de trouver le graphe et ensuite on trouve un ensemble de mots qui vérifie ce graphe comme graphe de chevauchements.

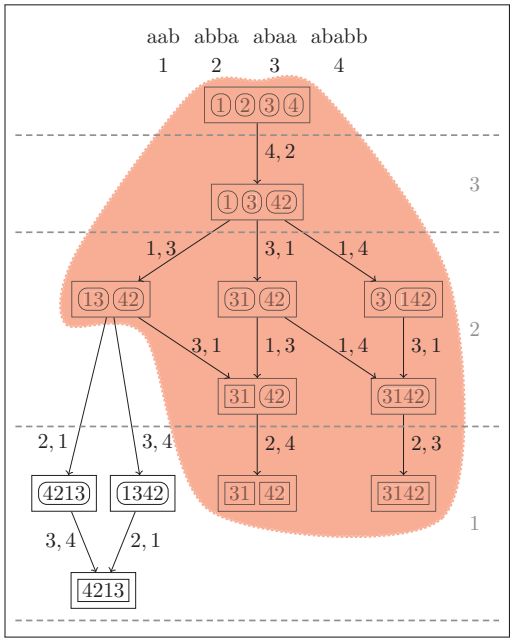
Si on utilise la technique de Gevezes et al. [28], à partir de G_P, on peut trouver un ensemble de mots P' qui satisfait ce graphe comme graphe des chevauchements. Cet ensemble de mots P' a comme propriété pour chaque mot de séparer les chevauchements à gauche des chevauchements à droite. On a alors que pour tout $w_{opt} \in OPT_{\mathsf{SLS}}(P')$, $||P'|| - w_{opt} \leq w_{opt}$ et donc $\frac{1}{2} \leq \frac{w_{opt}}{||P'||}$. D'après la proposition 3.9, comme $comp(Greedy_{LS}) = \frac{1}{2}$, on a que $super(P', Greedy_{LS}) \leq \frac{3}{2}$.

4.1.3.2 Ajustement de la conjecture

Actuellement la conjecture est que le ratio d'approximation optimal de l'algorithme glouton de SLS est de 2. On ne va ni diminuer ni augmenter ce ratio conjecturé, mais on



(a) $Greedy_{LS}$ (b) $Greedy_{CS}$



(c) $Greedy_{CCS}$

FIGURE 4.1 – Exemple de graphe de choix pour $P := \{aab, abba, abaa, ababb\}$. Sur la figure (a), on a mis en vert la partie du graphe de choix correspondant aux choix pris par l'algorithme $Greedy_{LS}$, sur la figure (b), on a mis en bleu la partie du graphe de choix correspondant aux choix pris par l'algorithme $Greedy_{CS}$ et sur la figure (c), on a mis en rouge la partie du graphe de choix correspondant aux choix pris par l'algorithme $Greedy_{CCS}$.

va pouvoir se demander si on ne peut pas trouver une borne supérieure un peu plus précise pour une solution de l'algorithme glouton de SLS.

Pour mieux comprendre cette nouvelle borne, on va commencer par définir une variante de SLS où on ne regarde que des ensembles de mots qui vérifient la propriété de l'ordre de glouton linéaire.

Définition 4.11 Soient P un ensemble de mots. L'ensemble P suit *l'ordre de glouton linéaire* si pour toute suite d'ensemble de mots $(P_1, \dots, P_{|P|})$ telle que $P_1 = P$, $P_{|P|} = \{w_l\}$ avec $w_l \in SOL_{Greedy_{LS}}(P)$ et pour tout i entre 1 et $|P| - 1$, $P_{i+1} \in SOL_{Greedy\_step_L}(P_i)$, on a $|P_{i+1} \setminus P_i| = 1$.

Dans l'article de Weinard et al. [95], on a la propriété suivante :

Proposition 4.12 *[95]* Soient P un ensemble de mots qui suit l'ordre de glouton linéaire, $w_{greedy} \in SOL_{Greedy_{LS}}(P)$, $w_{opt} \in OPT_{\mathsf{SLS}}(P)$ et $C_{opt} \in OPT_{\mathsf{SCCS}}(P)$. On a alors

$$|w_{greedy}| \leq |w_{opt}| + ||C_{opt}||$$

On peut alors se demander si cette propriété ne peut pas être généralisée pour tout ensemble P. Comme on le verra dans la section 4.3.2, pour tout ensemble de mots P, on a $SOL_{Greedy_{CCS}}(P) \subseteq OPT_{\mathsf{SCCS}}(P)$, on définit alors la conjecture suivante :

Conjecture Soient P un ensemble de mots, $w_{greedy} \in SOL_{Greedy_{LS}}(P)$, $w_{opt} \in OPT_{\mathsf{SLS}}(P)$ et $C_{greedy} \in SOL_{Greedy_{CCS}}(P)$. On a

$$|w_{greedy}| - ||C_{greedy}|| \leq |w_{opt}|$$

Remarque 4.13 Comme pour tout $w_{opt} \in OPT_{\mathsf{SLS}}(P)$ et $C_{opt} \in OPT_{\mathsf{SCCS}}(P)$, on a $|w_{opt}| \geq ||C_{opt}||$. On a alors que si $|w_{greedy}| - ||C_{greedy}|| \leq |w_{opt}|$, on a alors que $|w_{greedy}| \leq 2 \times |w_{opt}|$.

Conclusion de la sous-section Même si on trouve un graphe de chevauchements qui correspond à un contre-exemple, si on reconstruit un ensemble de mots qui vérifie ce graphe avec l'algorithme de Gevezes, ce nouvel ensemble de mots n'est plus un contre-exemple. De plus, on propose de préciser la conjecture historique en bornant la différence entre la taille d'une solution de l'algorithme glouton pour SLS et la taille d'une solution de l'algorithme glouton pour SCCS par la taille d'une solution optimale de SLS.

4.2 Système héréditaire

Dans cette section, on va résumer un travail de Mestre [59] sur les systèmes héréditaires pour pouvoir dans la section suivante (section 4.3), utiliser les résultats des systèmes héréditaires et les appliquer aux algorithmes gloutons sur les problèmes des superchaînes. Les systèmes héréditaires généralisent un autre concept : les matroïdes. Pour plus de détails sur les systèmes héréditaires (« *subset system* » en anglais) on pourra se référer à l'article de Mestre [59] et pour des détails sur les matroïdes, on pourra se référer à l'article de Whitney [97] ou au livre de Oxley [66].

Commençons par définir le concept de système héréditaire.

Définition 4.14 Un *système héréditaire* est un couple $(E, \mathcal{L})$ où E est un ensemble fini et $\mathcal{L}$ une famille de sous-ensembles de E (*i.e.* $\mathcal{L} \subseteq \mathcal{P}(E)$) satisfaisant les deux conditions suivantes :

— $\mathcal{L} \neq \emptyset$,

— si $A' \subseteq A$ et $A \in \mathcal{L}$, alors $A' \subseteq \mathcal{L}$.

Soient $(E, \mathcal{L})$ un système héréditaire et $A, B \in \mathcal{L}$. On dit que B est une *extension* de A si $A \subseteq B$. Soit un entier k, on dit alors qu'un système héréditaire $(E, \mathcal{L})$ est *k-extensible* si et seulement si pour tout $C \in \mathcal{L}$ et $x \notin C$ tels que $C \cup \{x\} \in \mathcal{L}$, et pour toute extension D de C, il existe un sous-ensemble $Y \subseteq D \setminus C$ avec $|Y| \leq k$ satisfaisant $(D \setminus Y) \cup \{x\} \in \mathcal{L}$. Un *matroïde* est un système héréditaire $(E, \mathcal{L})$ qui satisfait la condition suivante :

$$\forall A, B \in \mathcal{L}, \text{ et } |A| < |B|, \text{ on a alors qu'il existe } x \in B \setminus A \text{ tel que } A \cup \{x\} \in \mathcal{L}.$$

La proposition suivante relie les matroïdes et les systèmes héréditaires k-extensibles :

Proposition 4.15 *[59]* Le système héréditaire $(E, \mathcal{L})$ est un matroïde si et seulement s'il est 1-extensible.

Soient $(E, \mathcal{L}_1)$ et $(E, \mathcal{L}_2)$ deux matroïdes, le système héréditaire correspondant à l'intersection de $(E, \mathcal{L}_1)$ et $(E, \mathcal{L}_2)$ est $(E, \mathcal{L}_1 \cap \mathcal{L}_2)$. On peut alors généraliser cette intersection à k matroïdes et faire le lien entre intersection de matroïdes et systèmes héréditaires k-extensibles :

Proposition 4.16 *[59]* L'intersection de k matroïdes est un système héréditaire k-extensible.

Soit w une fonction de poids sur E. L'*algorithme glouton* associé au système héréditaire $(E, \mathcal{L})$ et à la fonction de poids w, noté $Greedy(E, \mathcal{L}, w)$, est l'algorithme 4.9.

Algorithme 4.9 : L'algorithme glouton $Greedy(E, \mathcal{L}, w)$ associé au système héréditaire $(E, \mathcal{L})$ et à la fonction de poids w.

```
1 Les éléments e_i de E sont triés par poids décroissants : w(e_1) ≥ w(e_2) ≥ ... ≥ w(e_n)
2 F ← ∅
3 for i = 1 to n do
4 |   if F ∪ {e_i} ∈ L then F ← F ∪ {e_i};
5 |   ;
6 return F
```

Mestre a établi un lien entre un système héréditaire k-extensible et une borne inférieure pour le ratio d'approximation de l'algorithme glouton associé à ce système héréditaire.

Théorème 4.17 *[59]* Soit $(E, \mathcal{L})$ un système héréditaire k-extensible, l'algorithme glouton associé à $(E, \mathcal{L})$ et à une fonction de poids quelconque w a un ratio d'approximation d'au moins $\frac{1}{k}$ pour le problème de maximisation défini par $(E, \mathcal{L})$.

Remarque 4.18 L'algorithme glouton $Greedy(E, \mathcal{L}, w)$ répond au problème de trouver l'élément $F \in \mathcal{L}$ qui maximise la somme $\sum_{e \in F} w(e)$. On a alors que l'algorithme glouton $Greedy(E, \mathcal{L}, w)$ répond à un problème de maximisation. Dans la suite, le théorème 4.17 ne pourra et ne sera appliqué qu'à des algorithmes gloutons d'un problème de maximisation.

4.3 Algorithmes gloutons et ratios d'approximation optimaux

Dans cette section, on va faire le lien entre les algorithmes gloutons classiques et les algorithmes gloutons associés à un système héréditaire. Pour cela, on va commencer par définir des systèmes héréditaires sur les graphes pour ensuite les définir sur les mots en les appliquant au graphe des chevauchements. Nous avons publié une partie de ces résultats dans (XI) et (VIII).

4.3.1 Sur les graphes

On a vu dans la section 3.4, le lien entre une superchaîne de P et un chemin hamiltonien sur le graphe des chevauchements de P, ainsi que le lien entre une superchaîne circulaire de P et un circuit hamiltonien sur le graphe des chevauchements de P, et enfin le lien entre une couverture circulaire de mots de P et un couverture circulaire sur le graphe des chevauchements de P. On va alors définir dans cette sous-section trois systèmes héréditaires : un pour modéliser la recherche d'un chemin hamiltonien, un pour modéliser la recherche d'un circuit hamiltonien et un pour modéliser la recherche d'une couverture circulaire.

Soit G un graphe orienté, on pose E_G l'ensemble des arcs de G. On va définir plusieurs familles $\mathcal{L}_i$ de sous-ensembles de E_G, de telle sorte que $(E_G, \mathcal{L}_i)$ forme un système héréditaire. Soient F un élément de la famille $\mathcal{L}_i$ et e un arc de E_G. L'ensemble $F \cup \{e\}$ n'est pas nécessairement un élément de $\mathcal{L}_i$. On va alors en plus définir une fonction Y_i de $\mathcal{L}_i \times E_G$ dans l'ensemble des sous-ensembles de E_G, *i.e.* $\mathcal{P}(E_G)$, telle que toute paire $(F, e) \in \mathcal{L}_i \times E_G$, l'ensemble d'arêtes $Y_i(F, e)$ est un ensemble d'arêtes que l'on doit enlever à F pour que $F \setminus Y_i(F, e) \cup \{e\}$ soit dans $\mathcal{L}_i$. Plus précisément, on va chercher $Y_i(F, e) \in \mathcal{P}(F)$, c'est-à-dire comme étant un sous-ensemble de F.

— Soit $\mathcal{L}_1(E_G)$ tel que $F \in \mathcal{L}_1(E_G)$ si et seulement si $F \subseteq E_G$ et pour tout $e \in F$, le cardinal de $\{e' \in F \mid e \text{ est suivi par } e'\}$ est d'au plus un. On pose alors l'application Y_1 (voir Figure 4.2) telle que :

$$\begin{array}{rcccl} Y_1 & : & \mathcal{L}_1 \times E_G & \to & \mathcal{P}(E_G) \\ & & (F, e) & \mapsto & \{f \in F \setminus \{e\} \mid e \text{ et } f \textbf{ ont le même début}\} \end{array}$$

— Soit $\mathcal{L}_2(E_G)$ tel que $F \in \mathcal{L}_2(E_G)$ si et seulement si $F \subseteq E_G$ et pour tout $e \in F$, le cardinal de $\{e' \in F \mid e' \text{ est suivi par } e\}$ est d'au plus un. On pose alors l'application Y_2 (voir Figure 4.3) telle que :

$$\begin{array}{rcccl} Y_2 & : & \mathcal{L}_2 \times E_G & \to & \mathcal{P}(E_G) \\ & & (F, e) & \mapsto & \{f \in F \setminus \{e\} \mid e \text{ et } f \textbf{ ont la même fin}\} \end{array}$$

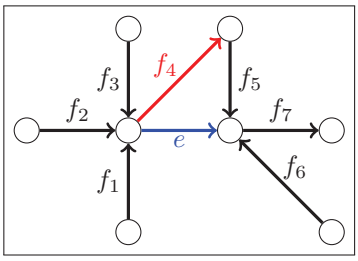


FIGURE 4.2 – Exemple de sous-graphe $F = \{f_1, \ldots, f_7\} \in (E_G, \mathcal{L}_1(E_G))$. On a alors que $Y_1(F, e) = \{f_4\}$.

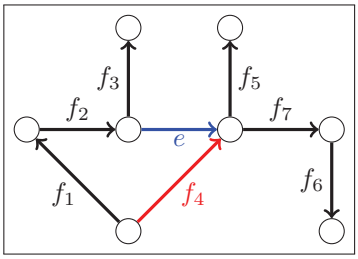


FIGURE 4.3 – Exemple de sous-graphe $F = \{f_1, \ldots, f_7\} \in (E_G, \mathcal{L}_2(E_G))$. On a alors que $Y_2(F, e) = \{f_4\}$.

— Soient r un entier et $\mathcal{L}_{3,r}(E_G)$ un ensemble tel que $F \in \mathcal{L}_{3,r}(E_G)$ si et seulement si $F \subseteq E_G$ et $\nexists C \subseteq F$ tel que C soit un sous-graphe circuit de G et $|C| \leq r$. On pose alors l'application Y_3 (voir Figure 4.4) telle que :

$$\begin{array}{rccl} Y_3 & : & \mathcal{L}_{3,r} \times E_G & \to & \mathcal{P}(E_G) \\ & & (F, e) & \mapsto & \{f \in F \setminus \{e\} \mid e \textbf{ est suivi par } f\} \end{array}$$

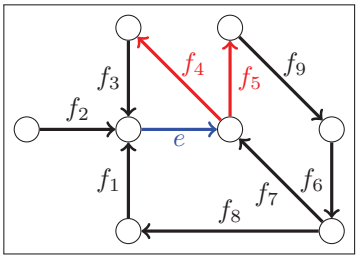


FIGURE 4.4 – Exemple de sous-graphe $F = \{f_1, \ldots, f_9\} \in (E_G, \mathcal{L}_{3,3}(E_G))$. On a alors que $Y_3(F, e) = \{f_4, f_5\}$.

Soient $F \in \mathcal{L}_1(E_G)$ et $e \in E_G \setminus F$; on a $|Y_1(F, e)| \leq 1$. En effet on sait que comme $F \in \mathcal{L}_1(E_G)$, $\forall f \in F$, $|Y_1(F, f)| = 0$. On a donc que $(E_G, \mathcal{L}_1(E_G))$ est 1-extensible et donc est un matroïde. De même, pour $F \in \mathcal{L}_2(E_G)$ et $e \in E_G \setminus F$, on a $|Y_2(F, e)| \leq 1$ et donc que $(E_G, \mathcal{L}_2(E_G))$ est un matroïde.

Malheureusement, pour certains (F, e) on ne peut pas borner $|Y_3(F, e)|$ par un nombre entier, on a donc que $(E_G, \mathcal{L}_{3,r}(E_G))$ n'est pas forcément un matroïde et cela pour tout

entier r (voir le contre exemple dans Figure 4.5).

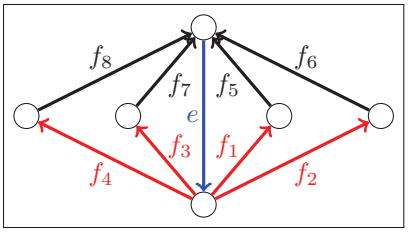


FIGURE 4.5 – Exemple de sous-graphe $F = \{f_1, \dots, f_8\} \in (E_G, \mathcal{L}_{3,3}(E_G))$. On a alors que $Y_3(F, e) = \{f_1, f_2, f_3, f_4\}$.

Même si on ne peut rien faire directement avec $(E_G, \mathcal{L}_{3,r}(E_G))$, on peut comparer l'intersection de ce système héréditaire avec les autres matroïdes. De manière à pouvoir appliquer la proposition 4.16, on va donner une extensibilité pour chacune de ces intersections.

Proposition 4.19 Soient G un graphe, n le nombre de sommets de G et E_G l'ensemble des arcs de G. On a alors

1. $(E_G, \mathcal{L}_1(E_G) \cap \mathcal{L}_2(E_G) \cap \mathcal{L}_{3,n}(E_G))$ est un système héréditaire 3-extensible.
2. $(E_G, \mathcal{L}_1(E_G) \cap \mathcal{L}_2(E_G) \cap \mathcal{L}_{3,n-1}(E_G))$ est un système héréditaire 3-extensible.
3. $(E_G, \mathcal{L}_1(E_G) \cap \mathcal{L}_2(E_G))$ est un système héréditaire 2-extensible.

Preuve On va montrer le point 1., c'est-à-dire que $(E_G, \mathcal{L}_1(E_G) \cap \mathcal{L}_2(E_G) \cap \mathcal{L}_{3,n}(E_G))$ est un système héréditaire 3-extensible. Les preuves des points 2. et 3. sont similaires et laissées au lecteur.

Soient $F \in \mathcal{L}_1(E_G) \cap \mathcal{L}_2(E_G) \cap \mathcal{L}_{3,n}(E_G)$ et $e \in E_G \setminus F$, on va prendre $Y^*(F, e) = Y_1(F, e) \cup Y_2(F, e) \cup Y_3(F, e)$. On sait que $|Y_1(F, e)| \leq 1$ et que $|Y_2(F, e)| \leq 1$. On va montrer que $|Y_3(F, e)| \leq 1$. Supposons que $|Y_3(F, e)| \geq 2$ et donc soient f_1 et f_2 deux éléments de $Y_3(F, e)$. Par la définition de $Y_3(F, e)$, e est suivi par f_1 et e est suivi par f_2, on a donc que f_1 et f_2 ont le même début. Ce qui est impossible car $F \in \mathcal{L}_1(E_G)$. D'où $|Y_3(F, e)| \leq 1$. On a alors

$$|Y^*(F, e)| \leq |Y_1(F, e)| + |Y_2(F, e)| + |Y_3(F, e)| \leq 3.$$

Comme $F \setminus Y^*(F, e) \cup \{e\} \in \mathcal{L}_1(E_G) \cap \mathcal{L}_2(E_G) \cap \mathcal{L}_{3,n}(E_G)$, on a que $(E_G, \mathcal{L}_1(E_G) \cap \mathcal{L}_2(E_G) \cap \mathcal{L}_{3,n}(E_G))$ est un système héréditaire 3-extensible. ■

Dans la proposition 4.19, on a défini trois systèmes héréditaires. On peut alors se demander quelle est la forme des éléments des systèmes héréditaires ainsi obtenus. On obtient alors la proposition suivante :

Proposition 4.20 Soient G un graphe, n le nombre de sommets de G et E_G l'ensemble des arcs de G. On a alors

1. $F \in \mathcal{L}_1(E_G) \cap \mathcal{L}_2(E_G) \cap \mathcal{L}_{3,n}(E_G)$ si et seulement si F est un sous-graphe d'un chemin hamiltonien de G.

2. $F \in \mathcal{L}_1(E_G) \cap \mathcal{L}_2(E_G) \cap \mathcal{L}_{3,n-1}(E_G)$ si et seulement si F est un sous-graphe d'un circuit hamiltonien de G.
3. $F \in \mathcal{L}_1(E_G) \cap \mathcal{L}_2(E_G)$ si et seulement si F est un sous-graphe d'une couverture cyclique de G.

Preuve Pour démontrer les points 1., 2. et 3., on ne va démontrer que la partie $\Rightarrow$ car l'autre sens ($\Leftarrow$) est évidente, il suffit de voir que le graphe vérifie les conditions et donc le sous-graphe aussi.

Montrons d'abord le point 3. Soit $F \in \mathcal{L}_1(E_G) \cap \mathcal{L}_2(E_G)$. Comme $F \in \mathcal{L}_1(E_G)$, il n'existe pas deux arêtes de F qui ont le même début. Comme $F \in \mathcal{L}_2(E_G)$, il n'existe pas deux arêtes de F qui partagent la même fin. On a donc que F est dans le sous-graphe d'une couverture cyclique de G.

Montrons le point 2. Soit $F \in \mathcal{L}_1(E_G) \cap \mathcal{L}_2(E_G) \cap \mathcal{L}_{3,n-1}(E_G)$. Comme $F \in \mathcal{L}_{3,n-1}(E_G)$, on a alors que F n'a pas de cycle plus petit ou égal à $n-1$. Comme en plus $F \in \mathcal{L}_1(E_G) \cap \mathcal{L}_2(E_G)$ et que le graphe G à n sommets, F est sous-graphe d'une couverture cyclique avec un cycle de longueur n, c'est-à-dire d'un circuit hamiltonien de G.

Enfin pour montrer le point 1, il suffit de voir que pour un $F \in \mathcal{L}_1(E_G) \cap \mathcal{L}_2(E_G) \cap \mathcal{L}_{3,n}(E_G)$, $F \in \mathcal{L}_1(E_G) \cap \mathcal{L}_2(E_G) \cap \mathcal{L}_{3,n-1}(E_G)$ et donc F est un sous-graphe d'un circuit hamiltonien de G. Comme $F \in \mathcal{L}_{3,n}(E_G)$, F n'admet aucun cycle de longueur plus petit ou égal à n, on a donc que F est un sous-graphe d'un circuit hamiltonien sans cycle de G, c'est-à-dire d'un chemin hamiltonien de G. ■

Le Théorème 4.17 nous donne un lien entre approximation de l'algorithme glouton d'un système héréditaire et extensibilité de ce même système héréditaire. On va maintenant faire le lien entre les problèmes sur les graphes et les algorithmes gloutons des systèmes héréditaires que l'on vient de définir.

Théorème 4.21 Soient G un graphe, n le nombre de sommets de G et E_G l'ensemble des arcs de G.

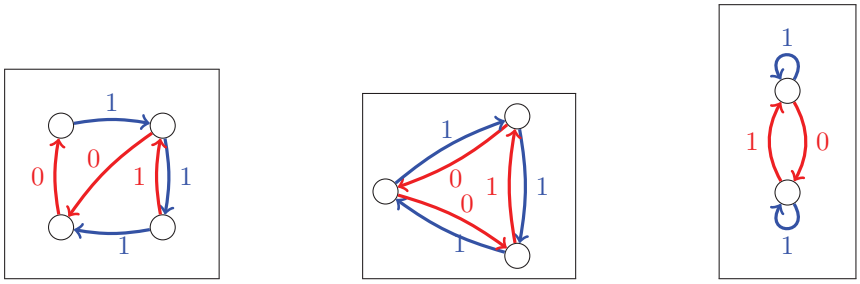
1. L'algorithme glouton pour $(E_G, \mathcal{L}_1(E_G) \cap \mathcal{L}_2(E_G) \cap \mathcal{L}_{3,n}(E_G))$ est un algorithme pour Max-HDP de ratio d'approximation optimal $\frac{1}{3}$.
2. L'algorithme glouton pour $(E_G, \mathcal{L}_1(E_G) \cap \mathcal{L}_2(E_G) \cap \mathcal{L}_{3,n-1}(E_G))$ est un algorithme pour Max-HDC de ratio d'approximation optimal $\frac{1}{3}$.
3. L'algorithme glouton pour $(E_G, \mathcal{L}_1(E_G) \cap \mathcal{L}_2(E_G))$ est un algorithme pour Max-HDCC de ratio d'approximation optimal $\frac{1}{2}$.

Preuve On va montrer le point 1., c'est-à-dire que l'algorithme glouton pour $(E_G, \mathcal{L}_1(E_G) \cap \mathcal{L}_2(E_G) \cap \mathcal{L}_{3,n}(E_G))$ est un algorithme de Max-HDP de ratio d'approximation optimal $\frac{1}{3}$.

Soit F une solution de l'algorithme $Greedy((E_G, \mathcal{L}_1(E_G) \cap \mathcal{L}_2(E_G) \cap \mathcal{L}_{3,n}(E_G)))$. On a alors, d'après la proposition 4.20, que F est un sous-graphe d'un chemin hamiltonien de G. Comme F est maximal pour l'inclusion par la définition de l'algorithme glouton, on a que F est un chemin hamiltonien de G. D'après le théorème 4.17, comme $(E_G, \mathcal{L}_1(E_G) \cap \mathcal{L}_2(E_G) \cap \mathcal{L}_{3,n}(E_G))$ est 3-extensible, l'algorithme glouton pour $(E_G, \mathcal{L}_1(E_G) \cap \mathcal{L}_2(E_G) \cap \mathcal{L}_{3,n}(E_G))$ est un algorithme de Max-HDP de ratio d'approximation optimal d'au moins

$\frac{1}{3}$. La figure 4.6a nous montre que la ratio d'approximation optimal est exactement de $\frac{1}{3}$.

Les points 2. et 3., ont une preuve similaire pour la borne supérieure du ratio d'approximation optimal des algorithmes $Greedy((E_G, \mathcal{L}_1(E_G) \cap \mathcal{L}_2(E_G) \cap \mathcal{L}_{3,n-1}(E_G)))$ et $Greedy((E_G, \mathcal{L}_1(E_G) \cap \mathcal{L}_2(E_G)))$. Pour montrer que ces ratios d'approximations sont optimaux, on utilise les contre-exemples des figures 4.6b et 4.6c.



(a) Max-HDP (b) Max-HDC (c) Max-HDCC

FIGURE 4.6 – Exemple d'instance pour déterminer la borne supérieure pour l'algorithme glouton pour Max-HDP (a), pour Max-HDC (b) et pour Max-HDCC (c). Pour chaque figure, le chemin (respectivement circuit) bleu est le circuit optimal et le chemin (respectivement circuit) rouge est le chemin (respectivement circuit) généré par l'algorithme glouton.

Conclusion de la sous-section Grâce aux systèmes héréditaires, on a montré dans cette partie que l'algorithme glouton pour Max-HDP et l'algorithme glouton pour Max-HDC ont un ratio d'approximation optimal de $\frac{1}{3}$, que l'algorithme glouton pour Max-HDCC a un ratio d'approximation optimal de $\frac{1}{2}$. Il est important de remarquer que le ratio d'approximation optimal de l'algorithme glouton pour Max-HDC a déjà été prouvé dans Jenkyns [38] comme égal à $\frac{1}{3}$.

4.3.2 Sur les superchaînes

D'après les propositions 3.38, 3.39 et 3.40, on déduit qu'à partir d'un algorithme d'approximation sur les graphes, on peut trouver un algorithme d'approximation pour les superchaînes. Or dans le Théorème 4.17, on a des ratios d'approximation de l'algorithme glouton pour les problèmes sur les graphes.

On va maintenant montrer que les algorithmes gloutons classiques $Greedy_{LS}$, $Greedy_{CS}$ et $Greedy_{CCS}$ peuvent être obtenus à partir des algorithmes gloutons des systèmes héréditaires $(E_G, \mathcal{L}_1(E_G) \cap \mathcal{L}_2(E_G) \cap \mathcal{L}_{3,n}(E_G))$, $(E_G, \mathcal{L}_1(E_G) \cap \mathcal{L}_2(E_G) \cap \mathcal{L}_{3,n-1}(E_G))$ et $(E_G, \mathcal{L}_1(E_G) \cap \mathcal{L}_2(E_G))$ où G est le graphe des chevauchements.

Proposition 4.22 ***voir Figure 4.7*** Soit P un ensemble de mots et G le graphe des chevauchements de P. On a alors que :

— $Greedy_{LS}$ est l'algorithme qui pour P, donne une superchaîne $sol_{HP \to LS}(p).u_n$ de P où $p := ((u_1, u_2), \ldots, (u_{n-1}, u_n))$ est un chemin hamiltonien de G donné par l'algorithme glouton pour le système héréditaire $(E_G, \mathcal{L}_1(E_G) \cap \mathcal{L}_2(E_G) \cap \mathcal{L}_{3,n}(E_G))$.

— $Greedy_{CS}$ est l'algorithme qui pour P, donne une superchaîne circulaire $sol_{HC\to CS}(c)$ de P où c est un circuit hamiltonien de G donné par l'algorithme glouton pour le système héréditaire $(E_G, \mathcal{L}_1(E_G) \cap \mathcal{L}_2(E_G) \cap \mathcal{L}_{3,n-1}(E_G))$.

— $Greedy_{CCS}$ est l'algorithme qui pour P, donne une couverture de mots circulaire $sol_{HCC\to CCS}(C)$ de P où C est une couverture cyclique de G donnée par l'algorithme glouton pour le système héréditaire $(E_G, \mathcal{L}_1(E_G) \cap \mathcal{L}_2(E_G))$.

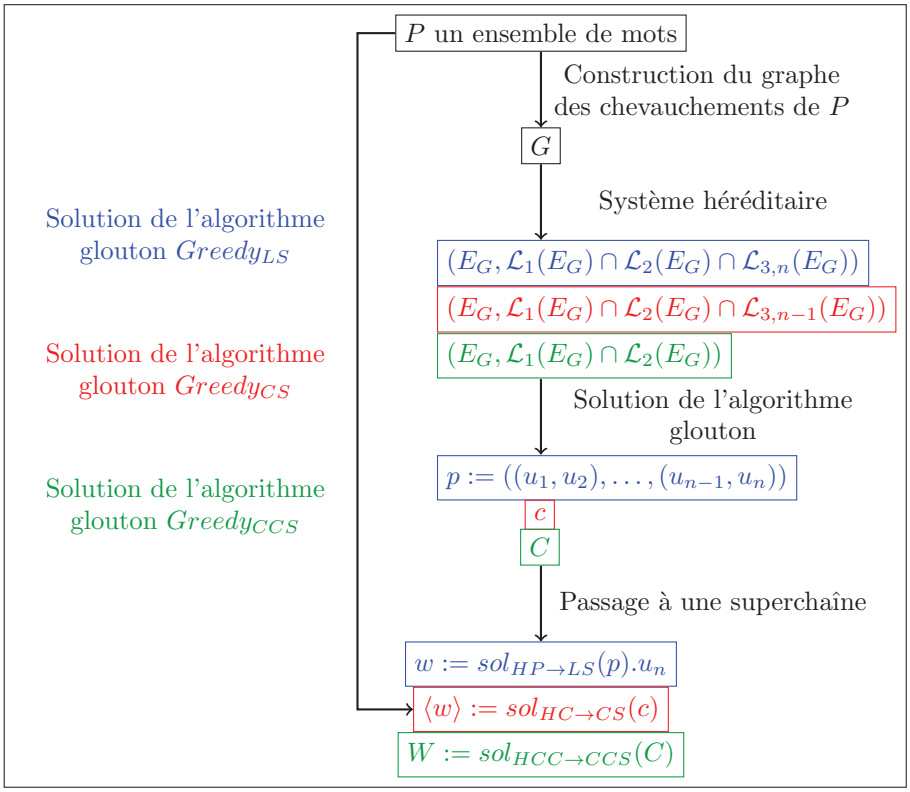


FIGURE 4.7 – Schéma résumant la proposition 4.22.

D'après les propositions 3.38, 3.39, 3.40 et 4.22, on a alors que l'algorithme glouton $Greedy_{LS}$ est un algorithme d'approximation de SLS_c de ratio $\frac{1}{3}$, $Greedy_{CS}$ est un algorithme d'approximation de SCS_c de ratio $\frac{1}{3}$ et $Greedy_{CCS}$ est un algorithme d'approximation de SCCS_c de ratio $\frac{1}{2}$.

Malheureusement, ces ratios d'approximation ne sont pas optimaux, car tous les graphes ne sont pas des graphes de chevauchements (voir Proposition 3.33). En reprenant la preuve de Mestre [59]), on peut alors améliorer ces ratios et trouver des ratios optimaux. Une preuve que $Greedy_{LS}$ est un algorithme d'approximation de SLS_c avec un ratio de $\frac{1}{2}$ est donnée dans [87, 90]. Ces preuves sont longues et examinent de nombreux cas. La preuve que l'on va donner ici utilise la puissance des systèmes héréditaires et de l'algorithme glouton pour simplifier au maximum la preuve.

Théorème 4.23

— L'algorithme glouton $Greedy_{LS}$ est un algorithme pour SLS_c de ratio d'approximation optimal $\frac{1}{2}$.

— L'algorithme glouton $Greedy_{CS}$ est un algorithme pour SCS_c de ratio d'approximation optimal $\frac{1}{2}$.

— L'algorithme glouton $Greedy_{CCS}$ est un algorithme pour SCCS_c de ratio d'approximation optimal 1.

Preuve Soit P un ensemble de mots. On va montrer que l'algorithme glouton $Greedy_{LS}$ est un algorithme d'approximation de SLS_c de ratio exact $\frac{1}{2}$. Les deux autres points du théorème ont une preuve similaire. La proposition 4.22 nous dit que $Greedy_{LS}$ est étroitement lié à l'algorithme glouton du système héréditaire $(E_G, \mathcal{L}_1(E_G) \cap \mathcal{L}_2(E_G) \cap \mathcal{L}_{3,n}(E_G))$ sur le graphe des chevauchements G de P. On va reprendre la preuve de Mestre [59] dans le cas où le graphe est un graphe des chevauchements.

Pour commencer cette preuve, on va poser $\mathcal{L}_p = \mathcal{L}_1(E_G) \cap \mathcal{L}_2(E_G) \cap \mathcal{L}_{3,n}(E_G)$, et pour tout $A \in \mathcal{L}_p$, on pose $OP(A)$ une extension de A de poids maximum. On a alors que $OP(\emptyset)$ est un élément de $\mathcal{L}_p$ de poids maximum, c'est-à-dire une solution de $OPT_{Max-HDP}(G)$.

Soient $F \in \mathcal{L}_p$ une solution de l'algorithme glouton du système héréditaire $(E_G, \mathcal{L}_p)$ et $x_1, x_2, \ldots, x_l$ l'ensemble des éléments pris dans l'ordre par l'algorithme glouton pour avoir à la fin F. Soient $F_0 := \emptyset, \ldots, F_{n-1}$ l'ensemble des valeurs successives de l'ensemble F pendant l'algorithme. En d'autres mots, on a $F_i := F_{i-1} \cup \{x_i\}$.

On va montrer dans un premier temps que pour tout élément x_i incorporé par l'algorithme glouton, on a l'inégalité $w(OP(F_{i-1})) \leq w(OP(F_i)) + w(x_i)$ et dans un second temps, on va raisonner par induction sur les ensembles F_i en commençant par F_0.

On sait que $OP(F_{i-1})$ est une extension de F_{i-1}. Par l'algorithme glouton et la définition de F_{i-1} et x_i, on a que $F_{i-1} \cup \{x_i\} \in \mathcal{L}_\mathcal{P}$. Comme $x_i \in E_G$, il existe s_p et s_o tels que $x_i = (s_p, s_o)$. En posant $Y_i = Y_1(OP(F_{i-1}) \setminus F_{i-1}, x_i) \cup Y_2(OP(F_{i-1}) \setminus F_{i-1}, x_i) \cup Y_3(OP(F_{i-1}) \setminus F_{i-1}, x_i)$, on a $OP(F_{i-1}) \setminus Y_i \cup \{x_i\} \in \mathcal{L}_\mathcal{P}$, et

$$\begin{aligned} w(OP(F_{i-1})) &= w(OP(F_{i-1}) \setminus Y_i \cup \{x_i\}) + w(Y_i) - w(x_i), \\ &\leq w(OP(F_i)) + w(Y_i) - w(x_i). \end{aligned}$$

En effet, $w(OP(F_{i-1}) \setminus Y_i \cup \{x_i\}) \leq w(OP(F_i))$ car $OP(F_{i-1}) \setminus Y_i \cup \{x_i\}$ est une extension de $F_{i-1} \cup \{x_i\}$ et car $OP(F_i)$ est une extension de poids maximum de $F_{i-1} \cup \{x_i\}$.

Maintenant montrons par contra-position que pour tout élément $y \in Y_i$, $w(y) \leq w(x_i)$. Supposons qu'il existe $y \in Y_i$ tel que $w(y) > w(x_i)$. Comme $y \notin F_{i-1}$, y a déjà été considéré par l'algorithme glouton mais pas ajouté à F. On a alors qu'il existe $j \leq i$ tel que $F_j \cup \{y\} \notin \mathcal{L}_\mathcal{P}$, mais $F_j \cup \{y\} \subseteq OP(F_{i-1}) \in \mathcal{L}_\mathcal{P}$, ce qui est absurde. On a donc que $w(y) \leq w(x_i)$ pour tout $y \in Y_i$.

On sait que $|Y_i| \leq 3$. Considérons deux cas :

1er cas : $|Y_i| \leq 2$.
Nous avons que $w(Y) \geq 2w(x_i)$, et donc $w(OP(F_{i-1})) \leq w(OP(F_i)) + w(x_i)$.

2nd cas : $|Y_i| = 3$.
Il existe s_k et $s_{k'}$ tels que $(s_p, s_{k'})$ et (s_k, s_o) sont dans Y_i. Comme $w(OP(F_i)) \leq$

$w(OP(F_{i-1}))$, on a que $w(x_i) + w((s_k, s_{k'})) \geq w((s_p, s_{k'})) + w((s_k, s_o))$. Comme $(s_p, s_{k'})$ et (s_k, s_o) sont dans $OP(F_{i-1})$, on a déduit que $(s_k, s_{k'}) \notin OP(F_{i-1})$.

Nous avons que $OP(F_{i-1}) \setminus Y_i \cup \{x_i, (s_k, s_{k'})\} \in \mathcal{L}_{\mathcal{P}}$. En effet, comme $Y_i \subseteq OP(F_{i-1})$, il n'y a ni de chevauchement à droite de s_k, ni de chevauchement à gauche de $s_{k'}$ dans $OP(F_{i-1})$. De plus, ajouter $(s_k, s_{k'})$ à $OP(F_{i-1}) \setminus Y_i \cup \{x_i\}$ ne peut pas créer de cycle, car sinon il existerait déjà un cycle dans $OPT(F_{i-1})$. Cette situation est illustrée par la figure 4.8.

Nous avons $w(OP(F_{i-1}) \setminus Y_i \cup \{x_i,\ (s_k, s_{k'})\}) \leq w(OP(F_{i-1} \cup \{x_i,\ (s_k, s_{k'})\}))$, car $OP(F_{i-1}) \setminus Y_i \cup \{x_i,\ (s_k, s_{k'})\}$ est une extension de $F_{i-1} \cup \{x_i,\ (s_k, s_{k'})\}$ et $OP(F_{i-1} \cup \{x_i,\ (s_k, s_{k'})\})$ est une extension de poids maximum $F_{i-1} \cup \{x_i,\ (s_k, s_{k'})\}$.

Comme $w(OP(F_{i-1} \cup \{x_i,\ (s_k, s_{k'})\})) \leq w(OP(F_{i-1} \cup \{x_i\}))$, on a :

$$\begin{array}{rcl} w(OP(F_{i-1})) & = & w(OP(F_{i-1}) \setminus Y_i \cup \{x_i,\ (s_k, s_{k'})\}) + w(Y_i) - w(x_i) \\ & & -w((s_k, s_{k'})), \\ & \leq & w(OP(F_{i-1} \cup \{x_i,\ (s_k, s_{k'})\})) + w(Y_i) - w(x_i) \\ & & -w((s_k, s_{k'})), \\ & \leq & w(OP(F_i)) + w(Y_i) - w(x_i) - w((s_k, s_{k'})). \end{array}$$

Comme $Y_i = \{(s_p, s_{k'}), (s_k, s_o), (s_{k''}, s_p)\}$, on obtient

$$\begin{array}{rcl} w(OP(F_{i-1})) & \leq & w(OP(F_i)) - w((s_k, s_{k'})) + w(Y_i) - w(x_i), \\ & \leq & w(OP(F_i)) - w((s_k, s_{k'})) + w((s_p, s_{k'})) + w((s_k, s_o)) \\ & & +w((s_{k''}, s_p)) - w(x_i). \\ & \leq & w(OP(F_i)) + w((s_{k''}, s_p)) \\ & \leq & w(OP(F_i)) + w(x_i). \end{array}$$

En se souvenant que $OP(\emptyset)$ est une solution de $OPT_{Max-HDP}(G)$, par induction, on a :

$$\begin{array}{rcl} w(OP(F_0)) & \leq & w(OP(F_{n-1})) + \sum_{i=1}^{n-1} w(x_i) \\ & \leq & w(F) + w(F) \\ & \leq & 2w(F). \end{array}$$

On peut substituer $w(OP(F_{n-1}))$ par $w(F)$ car F_{n-1} est maximal par inclusion et $F_{n-1} = F$. On a alors que l'algorithme glouton de $(E_G, \mathcal{L}_p)$ a un ratio d'approximation de $\frac{1}{2}$ sur Max-HDP pour le graphe des chevauchements.

On obtient alors que l'algorithme glouton $Greedy_{LS}$ a un ratio d'approximation de $\frac{1}{2}$ pour le problème SLS_c.

Si on prend $P_1 = \{ab, bb, bc\}$, on a alors que $abcbb$ est une solution de $Greedy_{LS}$ et $abbc \in OPT_{\mathsf{SLS}_c}(P_1)$. On a alors $\frac{||P_1||-|abcbb|}{||P_1||-|abbc|} = \frac{1}{2}$

On a alors que le ratio d'approximation de $Greedy_{LS}$ est exactement de $\frac{1}{2}$.

On a de plus que $< abcbb >$ est une solution de $Greedy_{CS}$ et $< abbc > \in OPT_{\mathsf{SLS}_c}(P_1)$. On a alors $\frac{||P_1||-|<abcbb>|}{||P_1||-|<abbc>|} = \frac{1}{2}$. ■

On obtient alors une preuve d'un fait annoncé pour SCCS mais pas démontré par Blum et al. [11] et Czuma et al. [19].

Corollaire 4.24 L'algorithme glouton $Greedy_{CCS}$ est un algorithme optimal de SCCS.

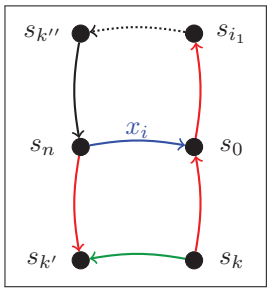


FIGURE 4.8 – Impossibilité de créer un cycle en ajoutant $(s_k, s_{k'})$ à $OP(F_{i-1}) \setminus Y_i \cup \{x_i\}$, sans créer un cycle dans $OP(F_{i-1})$. Comme on ajoute x_i à $OP(F_{i-1})$, on doit supprimer les trois éléments en rouge : $(s_{k''}, s_p), (s_p, s_{k'}), (s_k, s_o)$.

Preuve D'après le théorème 4.23, on a que le ratio d'approximation de l'algorithme glouton $Greedy_{CCS}$ sur SCCS_c est de 1. On a alors que l'algorithme glouton $Greedy_{CCS}$ est optimal sur SCCS_c. D'après la proposition 3.7, on a alors que l'algorithme glouton $Greedy_{CCS}$ est optimal sur SCCS. ■

Conclusion de la sous-section A l'aide des systèmes héréditaires, on a montré dans cette partie que l'algorithme glouton pour SLS_c et l'algorithme glouton pour SCS_c ont un ratio d'approximation optimal de $\frac{1}{2}$ et que l'algorithme glouton pour SCCS (et pour SCCS_c) est optimal.

Conclusion du chapitre Plus que les résultats, ce qui est important dans ce chapitre est la méthode utilisée. En effet, cette méthode peut être appliquée facilement à d'autres variantes de SLS_c et permet de trouver l'algorithme glouton correspondant et prouver relativement facilement des bornes de cet algorithme pour les ratios d'approximation optimaux de ces problèmes (voir Chapitre 5). Passer par les systèmes héréditaires des graphes permet de généraliser les preuves pour les versions compressions des problèmes de superchaînes sans toutefois en augmenter la difficulté de compréhension.

Chapitre 5

Autres problèmes liés aux superchaînes

Dans le chapitre 4, on a étudié des algorithmes gloutons classiques, c'est-à-dire des algorithmes gloutons pour SLS, SCS et SCCS, en utilisant des résultats sur les systèmes héréditaires. Dans ce chapitre, on va montrer que l'on peut étendre facilement cette étude à des variantes de ces problèmes des superchaînes. On va s'intéresser à des variantes qui nous sont imposées par les problèmes pratiques de l'assemblage, c'est-à-dire le cas du complémentaire renversé, le cas avec une multiplicité et le cas avec des chevauchements interdits.

Sommaire

5.1 Partition de l'instance

Avant de poser le problème du cas du complémentaire renversé, on commencera par généraliser le problème recherché. Ici, on va s'intéresser au problème sur les graphes où on veut un chemin qui passe une et une seule fois par chaque sommet d'une partition des sommets. On va ensuite appliquer ce problème à deux problèmes sur les mots. La première application va être du problème sur les superchaînes où on a en entrée un ensemble de chaînes circulaires et où on veut trouver le plus petit mot qui contient au moins un décalé circulaire de chaque chaîne circulaire. La deuxième application va être le problème sur les superchaînes où en entrée on a un ensemble de mots sur l'alphabet $\{A, T, G, C\}$ et on veut trouver une superchaîne qui contient pour chaque mot de l'instance, soit ce mot, soit le complémentaire-renversé de ce mot. Nous avons publié ces résultats dans (I) et (VI).

5.1.1 Chemin Maximum Partitionné Hamiltonien

On va commencer par définir le problème Chemin Maximum Partitionné Hamiltonien (Max-PHP). Soit (V, A) un graphe orienté complet tel que V est partitionné en p classes de nœuds $\{V_1, \ldots, V_p\}$.

Soit W un sous-ensemble de V et F un sous-ensemble de A. On note $n_W(F)$ le nombre de nœuds de W étant une extrémité d'un arc de F, *i.e.*

$$n_W(F) = |\{x \in W \mid \exists z \text{ tel que } (z, x) \in F \text{ ou } (x, z) \in F\}|.$$

On peut voir un chemin simple q de (V, A) comme un sous-ensemble A_q de A. Par simplicité, on note $n_W(A_q)$ par $n_W(q)$.

Un chemin q de (V, A) est dit *partitionné hamiltonien* sur $\{V_1, \ldots, V_p\}$ si q est un chemin simple et pour chaque i entre 1 et p, $n_{V_i}(q) = 1$. En d'autres termes, si pour chaque classe, il ne traverse qu'un seul élément de la classe. Le problème du *Chemin Partitionné Hamiltonien* (PHP, « Partitioned Hamiltonian Path problem » en anglais) cherche s'il existe un chemin partitionné hamiltonien sur $\{V_1, \ldots, V_p\}$.

Définissons maintenant le problème d'optimisation relié au problème du Chemin Maximum Partitionné Hamiltonien.

Définition 5.1 Max-PHP Soient (V, A, w) un graphe orienté complet avec w une fonction de poids non négative sur A, et $\{V_1, \ldots, V_p\}$ une partition de V. Le *Maximum Chemin Partitionné Hamiltonien* (ou Max-PHP) cherche un chemin partitionné hamiltonien p sur $\{V_1, \ldots, V_p\}$ qui maximise la somme des poids de ses arcs, *i.e.* $\sum_{e \in p} w(e)$.

Remarque 5.2 L'entrée du problème Max-PHP est un graphe $G = (V, A, w)$ orienté complet et pondéré, ainsi qu'une partition $\{V_1, \ldots, V_p\}$ de V. Comme le graphe est complet, l'ensemble A est forcément égal à $V \times V$. De plus, comme $\{V_1, \ldots, V_p\}$ est une partition de V, on a que $\bigcup_{i=1}^{p} V_i = V$. On peut donc prendre comme entrée du problème Max-PHP, un couple $(\{V_1, \ldots, V_p\}, w)$ où $\{V_1, \ldots, V_p\}$ est une partition de $\bigcup_{i=1}^{p} V_i$ et w une application de $\left(\bigcup_{i=1}^{p} V_i\right) \times \left(\bigcup_{i=1}^{p} V_i\right)$ vers $\mathbb{N}$.

Il est clair que Max-HDP est un cas spécial de Max-PHP où chaque classe est un singleton, *i.e.* pour un graphe $G = (V, A, w)$ orienté complet et pondéré, on a que

$$SOL_{Max-HDP}((V, A, w)) = SOL_{Max-PHP}((\bigcup_{v \in V} \{\{v\}\}, w)).$$

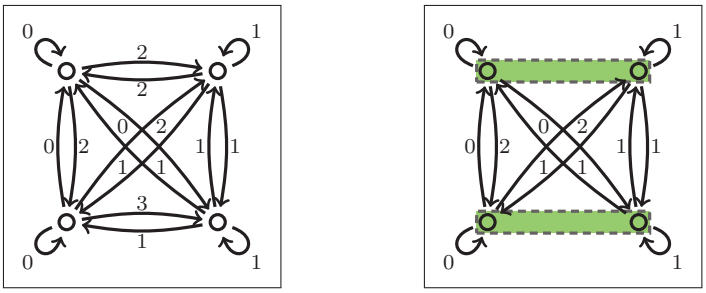
On en déduit que Max-PHP est **NP-Difficile** et est difficile à approximer. On va montrer que l'algorithme glouton sur Max-PHP admet un ratio d'approximation constant, ce qui veut dire que Max-PHP appartient à la classe **APX**.

On peut définir de même Max-PHC et Max-PHCC, les pendants de Max-PHP où on ne cherche plus un chemin, mais un circuit et une couverture cyclique respectivement.

On note $Graphe\_Part(\{V_1, \ldots, V_p\}, w) = (V = \bigcup_{i=1}^{p} V_i, A_C, w)$ le sous-graphe de (V, A, w) tel que

$$A_C = \{(x, y) \in A \mid x \in V_i, y \in V_j \text{ et } i \neq j\} \cup \{(x, x) \in A \mid x \in V\}.$$

Si on compare A_C à A, il manque à A_C tous les arcs reliant deux membres différents d'une même classe (voir Exemple 5.1).



(a) Graphe (V, A, w) (b) Graphe $Graphe\_Part(\{V_1, V_2\}, w)$

FIGURE 5.1 – Exemple de Graphe (V, A) et du sous-graphe $Graphe\_Part(\{V_1, V_2\}, w)$ pour la partition des nœuds en vert.

Maintenant, définissons un système héréditaire sur A_C l'ensemble des arcs du graphe $Graphe\_Part(\{V_1, V_2\}, w)$ dont on montrera après qu'il modélise le problème Max-PHP pour $(\{V_1, \dots, V_p\}, w)$.

Définition 5.3 Soit r un entier positif. On définit le système héréditaire $(A_C, \mathcal{M}_1 \cap \mathcal{M}_2 \cap \mathcal{M}_{3,r} \cap \mathcal{M}_4)$ tel que

- $F \in \mathcal{M}_1$ si et seulement si
  $\forall x, y \in V$ et $z, z' \in V_i$, (x, z) et $(y, z') \in F$ implique $x = y$ et $z = z'$,
- $F \in \mathcal{M}_2$ si et seulement si
  $\forall x, y \in V$ et $z, z' \in V_i$, (z, x) et $(z', y) \in F$ implique $x = y$ et $z = z'$,
- $F \in \mathcal{M}_{3,r}$ si et seulement si
  $\nexists C \subseteq F$ tel que C est un sous-graphe circuit de (V, A_C) et $|C| \leq r$,
- $F \in \mathcal{M}_4$ si et seulement si
  $\forall x, y \in V$ et $z, z' \in V_i$, (x, z) et $(z', y) \in F$ implique $z = z'$.

On a alors que $\mathcal{M}_1$ et $\mathcal{M}_2$ préviennent que deux arcs distincts ne rentrent ou ne sortent que par deux nœuds de la même classe. L'ensemble $\mathcal{M}_{3,r}$ prévient l'existence de circuit plus petit ou égal en taille à r et $\mathcal{M}_4$ interdit qu'un arc arrive par un nœud d'une classe et qu'un autre arc sorte avec un autre nœud de la même classe. Ces contraintes sont illustrées dans la Figure 5.2. On va de plus définir les systèmes héréditaires $(A_C, \mathcal{M}_1 \cap \mathcal{M}_2 \cap \mathcal{M}_{3,p-1} \cap \mathcal{M}_4)$ et $(A_C, \mathcal{M}_1 \cap \mathcal{M}_2 \cap \mathcal{M}_4)$ qui modéliseront les problèmes Max-PHC et de Max-PHCC. Pour chacun de ces trois systèmes héréditaires, on peut alors donner leur extensibilité.

Proposition 5.4 Les systèmes héréditaires $(A_C, \mathcal{M}_1 \cap \mathcal{M}_2 \cap \mathcal{M}_{3,p} \cap \mathcal{M}_4)$, $(A_C, \mathcal{M}_1 \cap \mathcal{M}_2 \cap \mathcal{M}_{3,p-1} \cap \mathcal{M}_4)$ et $(A_C, \mathcal{M}_1 \cap \mathcal{M}_2 \cap \mathcal{M}_4)$ sont 4-extensibles.

Preuve Montrons que le système héréditaire $(A_C, \mathcal{M}_1 \cap \mathcal{M}_2 \cap \mathcal{M}_{3,p} \cap \mathcal{M}_4)$ est 4-extensible. Les preuves des deux autres systèmes héréditaires sont similaires et laissées au lecteur.

Soient $C \in \mathcal{M}_1 \cap \mathcal{M}_2 \cap \mathcal{M}_{3,p} \cap \mathcal{M}_4$ et $a \in A_C \setminus C$ tel que $C \cup \{a\}$ est une extension de C. Soit D une autre extension de C. Comme $a \in A_C$, il existe $x \in V_i$ et $y \in V_j$ tels

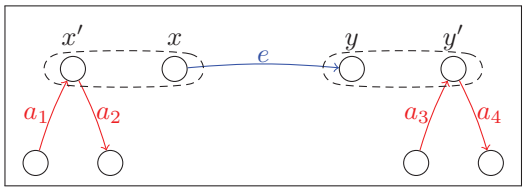


FIGURE 5.2 – Illustration pour la preuve de la proposition 5.4 des conditions définissant le système héréditaire pour Max-PHP. Dans la preuve, on souhaite étendre la solution courante D avec un arc a reliant x à y. Comme e n'est pas compatible avec les arcs a_1, a_2, a_3, a_4 (en rouge) déjà dans D, car x et x' sont dans la même classe ainsi que y et y' (les classes sont représentées part les lignes à tirets), on ne peut ajouter e tout en gardant $\{a_1, a_2, a_3, a_4\}$ dans la solution.

que $i \neq j$ et $a = (x, y)$. On peut alors définir Y tel que

$$\begin{aligned} Y &= \{(x', y') \in D \setminus C \mid x' \in V \text{ et } y' \in V_j\} \cup \\ &\quad \{(x', y') \in D \setminus C \mid x' \in V_i \text{ et } y' \in V\} \cup \\ &\quad \{(x', y') \in D \setminus C \mid x' \in V \text{ et } y' \in V_i\} \cup \\ &\quad \{(x', y') \in D \setminus C \mid x' \in V_j \text{ et } y' \in V\} \\ &= (D \setminus C) \cap \Big((V \times V_j) \cup (V_i \times V) \cup (V \times V_i) \cup (V_j \times V)\Big). \end{aligned}$$

Comme D est dans $\mathcal{M}_1 \cap \mathcal{M}_2 \cap \mathcal{M}_4$, $|D \cap (V \times V_j)| \leq 1$, $|D \cap (V_i \times V)| \leq 1$, $|D \cap (V \times V_i)| \leq 1$ et $|D \cap (V_j \times V)| \leq 1$. On a alors que

$$\begin{aligned} |Y| &= |(D \setminus C) \cap \Big((V \times V_j) \cup (V_i \times V) \cup (V \times V_i) \cup (V_j \times V)\Big)| \\ &\leq |D \cap \Big((V \times V_j) \cup (V_i \times V) \cup (V \times V_i) \cup (V_j \times V)\Big)| \\ &\leq |D \cap (V \times V_j)| + |D \cap (V_i \times V)| + |D \cap (V \times V_i)| + |D \cap (V_j \times V)| \\ &\leq 4 \end{aligned}$$

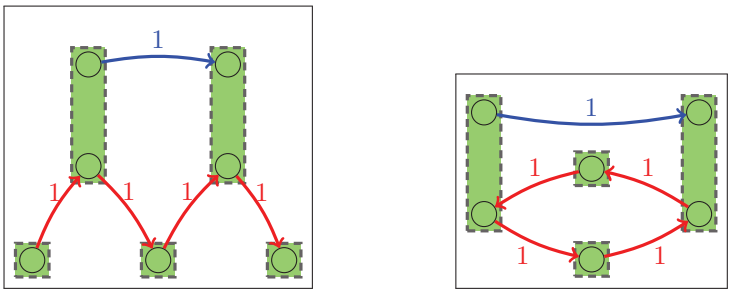
Par définition de Y, $D \setminus Y \cup \{a\} \in \mathcal{M}_1 \cap \mathcal{M}_2 \cap \mathcal{M}_{3,p} \cap \mathcal{M}_4$. En effet, a ne peut pas appartenir à une cycle de $D \setminus Y \cup \{a\}$, car Y contient les arcs entrants dans x et sortants de y. On a alors, $D \setminus Y \cup \{a\}$ est aussi dans $\mathcal{M}_{3,p}$. L'ensemble $D \setminus Y \cup \{a\}$ appartient à $\mathcal{M}_1 \cap \mathcal{M}_2 \cap \mathcal{M}_{3,p} \cap \mathcal{M}_4$, ce qui termine la preuve de l'extensibilité de $(A_C, \mathcal{M}_1 \cap \mathcal{M}_2 \cap \mathcal{M}_{3,p} \cap \mathcal{M}_4)$. ■

Nous obtenons donc les ratios d'approximation optimaux des algorithmes gloutons pour les problèmes Max-PHP, Max-PHC et Max-PHCC.

Théorème 5.5

1. L'algorithme glouton de $(A_C, \mathcal{M}_1 \cap \mathcal{M}_2 \cap \mathcal{M}_{3,p} \cap \mathcal{M}_4)$ a un ratio d'approximation optimal de $\frac{1}{4}$ pour Max-PHP.
2. L'algorithme glouton de $(A_C, \mathcal{M}_1 \cap \mathcal{M}_2 \cap \mathcal{M}_{3,p-1} \cap \mathcal{M}_4)$ a un ratio d'approximation optimal de $\frac{1}{4}$ pour Max-PHC.
3. L'algorithme glouton de $(A_C, \mathcal{M}_1 \cap \mathcal{M}_2 \cap \mathcal{M}_4)$ a un ratio d'approximation optimal de $\frac{1}{4}$ pour Max-PHCC.

Preuve En utilisant le théorème 4.17 et la proposition 5.4, on obtient la borne inférieure du ratio d'approximation optimal de chacun des trois problèmes. La figure 5.3 montre des exemples tels que la solution gloutonne est bien quatre fois plus petite que la solution optimale, montrant ainsi que ces trois ratios sont optimaux. ■



(a) Max-PHP (b) Max-PHC et Max-PHCC

FIGURE 5.3 – Exemple d'instance pour déterminer la borne supérieure pour l'algorithme glouton pour Max-PHP (a), pour Max-PHC et pour Max-PHCC (b). Pour chaque figure, le circuit bleu est le circuit optimal et le circuit rouge est le circuit généré par l'algorithme glouton.

Conclusion de la sous-section Dans cette sous-section, on a montré que les algorithmes gloutons pour les problèmes Max-PHP, Max-PHC et Max-PHCC ont chacun un ratio d'approximation optimal plus grand ou égal à $\frac{1}{4}$. Les problèmes Max-PHP, Max-PHC et Max-PHCC sont donc **APX**, c'est-à-dire qu'il existe des algorithmes avec un ratio d'approximation optimal constant.

5.1.2 Shortest Multi-Cyclic To Linear Superstring

On va maintenant appliquer le problème Max-PHP aux problèmes sur les mots. On va commencer par prendre des mots circulaires en entrée. En décomposant chaque mot circulaire en un ensemble de mots linéaires représentant ce mot circulaire et en prenant comme partition cet ensemble, on retrouve le problème Max-PHP. On va définir tout cela formellement.

Soit $P := \{\langle c_1\rangle, \dots, \langle c_p\rangle\}$ un ensemble de p mots circulaires. P est dit *cyclic-factor free* si pour tout i, j distincts entre 1 et p, il n'existe pas $x \in Lin(\langle c_i\rangle)$ et $y \in Lin(\langle c_j\rangle)$ tels que x est une sous-chaîne de y.

On peut alors définir le pendant du problème SLS qui prend en entrée un ensemble de mots circulaires.

Définition 5.6 ***Shortest Multi-Cyclic To Linear Superstring*** *(*SMC2LS*)* Soit $P := \{\langle c_1\rangle, \dots, \langle c_p\rangle\}$ un ensemble de mots circulaires cyclic-factor free. Le problème *Shortest Multi-Cyclic To Linear Superstring* (SMC2LS) cherche un mot linéaire w qui minimise la taille de w et tel que pour chaque $i \in \{1, \dots, p\}$, au moins un décalage circulaire de $\langle c_i\rangle$ soit une sous-chaîne de w.

Exemple 5.7 Soit $P = \{\langle abba\rangle, \langle bbab\rangle\}$. On a alors que *abbab* et *bbbaa* sont deux solutions optimales du problème SMC2LS pour P.

On peut définir de même SMC2CS et SMC2CCS les pendants de SMC2LS où on ne cherche plus à trouver le plus petit mot linéaire, mais le plus petit mot circulaire et la plus petite couverture de mots circulaires.

On a clairement qu'une solution de SMC2LS (respectivement SMC2CS et SMC2CCS) peut être calculée en appliquant l'algorithme glouton sur Max-PHP (respectivement Max-PHC et Max-PHCC) pour une variante du graphe de chevauchement qui serait adapté au mots circulaires en entrée.

Définition 5.8 ***Graphe des chevauchements partitionné*** Soient P un ensemble de mots circulaires cyclic-factor free. Le *graphe des chevauchements partitionné* (V_P, E_P) de P est tel que :

$$\begin{aligned} V_P &= \bigcup_{\langle c\rangle \in P} Lin(\langle c\rangle) \\ E_P &= \{(u,v) \mid \exists \langle c\rangle \text{ et } \langle c'\rangle \in P \text{ tels que } (u,v) \in Lin(\langle c\rangle) \times Lin(\langle c'\rangle) \\ &\quad \text{et } \langle c\rangle \neq \langle c'\rangle\} \cup \{(u,u) \mid u \in V_P\}. \end{aligned}$$

Remarque 5.9 L'ensemble d'arcs $\{(u,u) \mid u \in V_P\}$ est utile pour le problème SMC2CCS.

Un exemple de graphe des chevauchements partitionné est donné dans la figure 5.4.

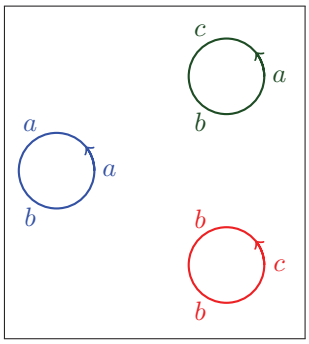
Remarque 5.10 Comme P est cyclic-factor free, pour tous $\langle c_i\rangle$ et $\langle c_j\rangle \in P$ tels que $\langle c_i\rangle \neq \langle c_j\rangle$, on a alors $Lin(c_i) \cap Lin(c_j) = \emptyset$. On a alors que $\bigcup_{\langle c\rangle \in P}\{Lin(\langle c\rangle)\}$ est bien une partition de $\bigcup_{\langle c\rangle \in P} Lin(\langle c\rangle)$.

Pour un ensemble de mots circulaires P, on peut alors faire le lien entre le graphe des chevauchements de $\bigcup_{\langle c\rangle \in P} Lin(\langle c\rangle)$ et le graphe des chevauchements partitionné de P que l'on peut écrire comme le graphe $Graphe\_Part(\bigcup_{\langle c\rangle \in P}\{Lin(\langle c\rangle)\}, ov)$ où ov est l'application de $\Sigma^* \times \Sigma^*$ dans $\mathbb{N}$ qui correspond à la taille du chevauchement maximal.

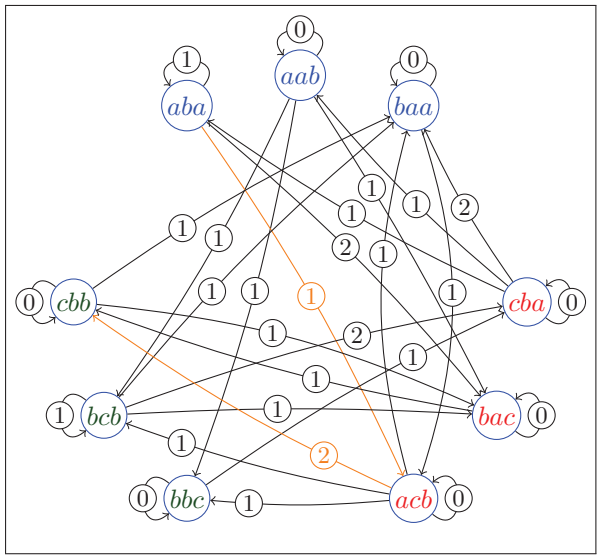
En prenant l'algorithme glouton sur Max-PHP et en l'appliquant sur le graphe des chevauchements partitionné, on obtient alors un algorithme sur SMC2LS. De même, on peut obtenir un algorithme sur SMC2CS et SMC2CCS. Grâce aux ratios d'approximation optimaux des algorithmes gloutons pour Max-PHP, Max-PHC et Max-PHCC obtenus par le théorème 5.5, on peut définir une borne pour le ratio d'approximation optimal pour SMC2LS_c, SMC2CS_c et SMC2CCS_c (les versions compression de SMC2LS, SMC2CS et SMC2CCS).

Théorème 5.11 Soient $P := \{\langle c_1\rangle, \ldots, \langle c_p\rangle\}$ un ensemble de mots circulaires et G le graphe des chevauchements partionné de P.

- L'algorithme glouton défini pour Max-PHP et appliqué sur G a un ratio d'approximation optimal supérieur ou égal à $\frac{1}{4}$ pour SMC2LS_c,
- L'algorithme glouton défini pour Max-PHC et appliqué sur G a un ratio d'approximation optimal supérieur ou égal à $\frac{1}{4}$ pour SMC2CS_c,
- L'algorithme glouton défini pour Max-PHCC et appliqué sur G a un ratio d'approximation optimal supérieur ou égal à $\frac{1}{4}$ pour SMC2CCS_c.



(a) Un ensemble P de 3 mots circulaires



(b) Le graphe des chevauchements partionné de P avec un chemin partitionné hamiltonien

FIGURE 5.4 – Exemple de graphe des chevauchements partitionné de $P := \{\langle aba\rangle, \langle bbc\rangle, \langle acb\rangle\}$. Tous les décalages circulaires d'un même mot apparaissent de la même couleur (bleu pour $\langle aba\rangle$, vert pour $\langle bbc\rangle$, rouge pour $\langle acb\rangle$). On n'a représenté que les arcs avec un poids strictement positif. Une solution non optimale de Max-PHP pour P a été représentée en orange et donne le mot linéaire *abacbb*.

Remarque 5.12 Les bornes données par le théorème 5.11 ne sont pas strictes.

Q : Les problèmes SMC2LS$_c$, SMC2CS$_c$ et SMC2CCS$_c$ sont-ils NP-difficile ?

Q : Quels sont les ratios d'approximation optimaux des algorithmes gloutons pour SMC2LS, SMC2LS$_c$, SMC2CS, SMC2CS$_c$, SMC2CCS et SMC2CCS$_c$?

Conclusion de la sous-section On a montré dans cette partie que pour chacun des problèmes SMC2LS$_c$, SMC2CS$_c$ et SMC2CCS$_c$, il existe un algorithme avec un ratio d'approximation optimal plus grand au égal à $\frac{1}{4}$. On a donc que les problèmes SMC2LS$_c$, SMC2CS$_c$ et SMC2CCS$_c$ sont **APX**.

5.1.3 Complémentaire-renversé

On va voir maintenant une autre application du problème Max-PHP aux problèmes sur les mots. Une séquence d'ADN peut être vue comme un mot sur l'alphabet $\Sigma = \{A, T, C, G\}$. Comme l'ADN est une molécule à double brin, chaque séquence sur un brin a son reverse complement qui est sur l'autre brin, *i.e.* en lisant de droite à gauche en complémentant ($A \leftrightarrow T$, $C \leftrightarrow G$). On note le *complémentaire-renversé* d'un mot w par $\overleftarrow{\overline{w}}$. Cette notion et la notation sont étendues à un ensemble fini de mots. L'opération de complémentaire-renversé d'un mot est une composition d'une inversion avec une permutation d'ordre 2 sur Σ (voir remarque 5.14).

Exemple 5.13 Soient $w = ATAGGA$ un mot linéaire, $\langle w \rangle = \langle CGTAG \rangle$ un mot circulaire et $P = \{ATTC, TTA, CGGC\}$ un ensemble de mots sur $\Sigma = \{A, T, C, G\}$. On a alors que $\overleftarrow{w} = TCCTAT$, que $\overleftarrow{\langle w \rangle} = \langle \overleftarrow{CTACG} \rangle$ et que $\overleftarrow{P} = \{\overleftarrow{ATTC}, \overleftarrow{TTA}, \overleftarrow{CGCG}\} = \{GAAT, TAA, CGCG\}$.

Remarque 5.14 Dans (VI), on a montré que les résultats que l'on a pour le cas du complémentaire-renversé pouvait être généralisé à toutes compositions d'une inversion avec une permutation d'ordre 2 sur Σ.

La proposition suivante nous donne un lien entre le chevauchement maximal et le complémentaire-renversé.

Proposition 5.15 Soient u et v deux mots linéaires. On a alors $ov(\overleftarrow{u}, \overleftarrow{v}) = \overleftarrow{ov(v, u)}$.

Soit P un ensemble de mots sur $\{A, T, C, G\}$. On dit alors que P est *ADN-factor-free* si $P \cup \overleftarrow{P}$ est factor-free.

Exemple 5.16 Soient $P_1 = \{ATCCA, GGA\}$ et $P_2 = \{ATCCA, GGT\}$. On a alors que P_2 est ADN-factor-free mais que P_1 ne l'est pas. En effet $\overleftarrow{GGA} = TCC$ est une sous-chaîne de $ATCCA$.

On va introduire une variante pour le cas de l'ADN de la superchaîne, l'*ADN-superchaîne* de P : c'est un mot w tel que pour tout mot $s_i \in P$, s_i ou $\overleftarrow{s_i}$ est une sous-chaîne de w.

On peut définir le problème de trouver la plus petite ADN-superchaîne d'un ensemble de mots DNA-factor-free.

Problèmme 5.17 ***Shortest DNA Linear Superstring*** *(*SDLS*)* Soit P un ensemble de mots ADN-factor-free sur $\{A, T, C, G\}$. Le problème Shortest DNA Linear Superstring cherche une ADN-superchaîne w de P qui est de longueur minimale.

Exemple 5.18 Soit $P = \{ATTCA, GGTG, CCAG\}$ un ensemble de mots.

Le mot $ATTCACCAG$ est une ADN-superchaîne de P, car $ATTCA$, $\overleftarrow{GGTG} = CACC$ et $CCAG$ sont des sous-chaînes de $ATTCACCAG$.

On définit de même les problèmes SDCS et SDCCS qui cherchent la plus petite *ADN-superchaîne circulaire* et la plus petite *ADN-couverture de mots circulaires* (on ne cherche plus un mot de P comme sous-chaîne mais un mot de P ou son complémentaire-renversé).

On peut alors s'intéresser à la complexité des problèmes SDLS et SDCS. En effet on ne regarde pas ici SDCCS, d'après le théorème 5.22, on a que SDCCS_d est dans **P**.

Théorème 5.19 Les problèmes SDLS_d et SDCS_d sont **NP-complets** même si l'entrée est écrite sur un alphabet de cardinalité 2.

Preuve La preuve pour SDLS_d peut être trouvée dans (VI). ■

Algorithme 5.1 : L'algorithme glouton pour SDLS.

Entrée : Soit P un ensemble de mots.

1 **while** $|P| > 1$ **do**

2 Soit u et v les deux mots de $P \cup \overleftarrow{P}$ dont le chevauchement maximal est le plus grand;

3 $P \leftarrow P \cup \{u \oplus v\} \setminus \{u, v, \overleftarrow{u}, \overleftarrow{v}\}$;

4 **return** w tel que $P = \{w\}$.

On peut alors définir l'algorithme glouton sur SDLS (voir Algorithme 5.1). On peut définir de même l'algorithme glouton sur SDCS et SDCCS.

Comme on l'a fait dans sous-section 5.1.2, on va définir un graphe des chevauchements qui modélise notre problème.

Définition 5.20 ***Graphe des chevauchements reverse-complement (voir Figure 5.5)*** Soit P un ensemble de mots ADN-factor-free. Le *graphe des chevauchements reverse-complement* (V_P, E_P) de P est tel que :

$$\begin{aligned} V_P &= \bigcup_{w \in P} \{w, \overleftarrow{w}\} \\ E_P &= \{(u,v) \in V_P \times V_P \mid u \neq v \text{ et } u \neq \overleftarrow{v}\} \cup \{(u,u) \mid u \in V_P\}. \end{aligned}$$

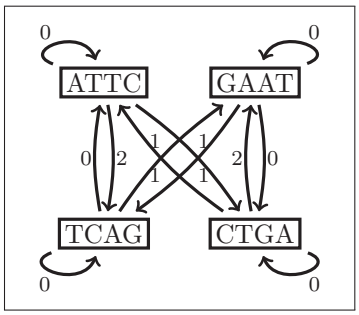


FIGURE 5.5 – Exemple de graphe des chevauchements reverse-complement pour $P = \{ATTC, CTGA\}$.

Remarque 5.21 Comme P est ADN-factor-free, pour tous w_i et $w_j \in P$ tels que $w_i \neq w_j$ et $w_i \neq \overleftarrow{w_j}$, on a alors $\{w_i, \overleftarrow{w_i}\} \cap \{w_j, \overleftarrow{w_j}\} = \emptyset$. On a alors que $\bigcup_{w \in P}\{\{w, \overleftarrow{w}\}\}$ est bien une partition de $\bigcup_{w \in P}\{w, \overleftarrow{w}\}$.

Pour un ensemble de mots linéaires P, on peut, comme dans la sous-section précédente, établir le lien entre le graphe des chevauchements de $\bigcup_{w \in P}\{w, \overleftarrow{w}\}$ et le graphe des chevauchements reverse-complement de P que l'on peut écrire comme le graphe

$$Graphe\_Part(\bigcup_{w \in P} \{\{w, \overleftarrow{w}\}\}, ov)$$

où ov est l'application de $\Sigma^* \times \Sigma^*$ dans $\mathbb{N}$ qui correspond à la taille du chevauchement maximal.

En prenant l'algorithme glouton sur Max-PHP (respectivement Max-PHC et Max-PHCC) et en l'appliquant sur le graphe des chevauchements reverse-complement, on obtient alors un algorithme sur SDLS (respectivement SDCS et SDCCS). À l'aide des ratios d'approximation optimaux des algorithmes gloutons pour Max-PHP, Max-PHC et Max-PHCC obtenus par le théorème 5.5, on peut borner le ratio d'approximation optimal pour SDLS_c, SDCS_c et SDCCS_c par $\frac{1}{4}$.

En utilisant la proposition 5.15 et le fait que le graphe des chevauchements reverse-complement satisfait l'inégalité supérieure de Monge, on peut améliorer facilement les bornes du ratio d'approximation optimal de l'algorithme glouton défini pour Max-PHP, (resp. Max-PHC et Max-PHCC) et appliqué sur le graphe des chevauchements reverse-complement de P.

Théorème 5.22 Soient $P := \{s_1, \ldots, s_n\}$ un ensemble de mots et G le graphe des chevauchements reverse-complement de P.

1. L'algorithme glouton défini pour Max-PHP et appliqué sur G a un ratio d'approximation optimal de $\frac{1}{2}$ pour SDLS_c,
2. L'algorithme glouton défini pour Max-PHC et appliqué sur G a un ratio d'approximation optimal de $\frac{1}{2}$ pour SDCS_c,
3. L'algorithme glouton défini pour Max-PHCC et appliqué sur G a un ratio d'approximation optimal de 1 pour SDCCS_c.

Preuve Une preuve de ce théorème pourra être trouvée dans (VI) et est similaire à une des preuves que l'on a réalisé dans (VIII). ■

Grâce au 3. du théorème 5.22, on a alors le résultat suivant :

Corollaire 5.23 L'algorithme glouton défini pour Max-PHCC et appliqué sur le graphe des chevauchements reverse-complement de P est un algorithme optimal de SDCCS.

Conclusion de la sous-section Dans cette partie, on a montré que les problèmes SDLS_c et SDCS_c sont dans **NP-complets** et **APX**. En effet, il existe un algorithme pour SDLS_c et SDCS_c avec un ratio d'approximation optimal de $\frac{1}{2}$. Le problème SDCCS est quant à lui dans **P**, car on a montré que l'algorithme glouton est optimal pour SDCCS.

5.2 Multiplicité

On va maintenant s'intéresser aux problèmes des superchaînes avec multiplicités. La multiplicité d'une chaîne en entrée indique le nombre d'occurrences voulu dans la superchaîne à calculer. On sait que ce problème a déjà un algorithme exact polynomial dans le cas où le nombre de mots de l'instance est borné et un algorithme exact quadratique en la norme de l'instance dans le cas où les mots de l'instance sont de longueur 2 : on peut retrouver ce résultat dans Crochemore et al. [16]. On va regarder comment réagit l'algorithme glouton pour ce problème.

Commençons par définir le problème formellement :

Problèmme 5.24 ***Multi Shortest Linear Superstring*** *(*Multi-SLS*)* Soient $P := \{s_1, \dots, s_n\}$, un ensemble de mots et m une application de P vers $\mathbb{N}^*$ qui donnera une multiplicité pour chaque mot de P. On cherche un mot linéaire w qui soit de longueur minimale et tel que pour tout $s_i \in P$, $|Occ(s_i, w)| \geq m(s_i)$.

On peut de même définir les problèmes Multi-SCS et Multi-SCCS qui cherchent non pas un des plus petits mots w, mais un des plus petits mots circulaires c et une des plus petites couvertures de mots circulaires W où on peut voir chaque mot s_i de P au moins $m(s_i)$ fois.

Soient $P := \{s_1, \dots, s_n\}$ un ensemble de mots et $m : P \to \mathbb{N}^*$. Ici l'ensemble P n'est pas forcément factor-free. On va définir une propriété qui va généraliser le principe en définissant une nouvelle application $reduc_m$ telle que :

$$\begin{array}{rccl} reduc_m & : & P & \to & \mathbb{N}^* \\ & & s_i & \mapsto & max(0, m(s_i) - \sum_{\substack{s_i \subset s_j \\ \text{sub}}} reduc_m(s_j)) \end{array}$$

et un ensemble de mots $Reduc(P, m) = \{v \in P \mid m(v) > 0\}$.

Cette application prend en compte le fait qu'un mot peut être sous-chaîne d'un autre et qu'il faut alors réduire sa multiplicité du nombre d'occurrences demandé par le mot qui le contient.

Exemple 5.25 Soient $P = \{abb, ba, bb, b\}$ et m tel que

$$\begin{array}{rccc} m & : & abb & \mapsto & 2 \\ & & bb & \mapsto & 3 \\ & & ba & \mapsto & 2 \\ & & b & \mapsto & 6 \end{array}$$

On a alors que

$$\begin{array}{rccc} reduc_m & : & abb & \mapsto & 2 \\ & & bb & \mapsto & 1 \\ & & ba & \mapsto & 2 \\ & & b & \mapsto & 0 \end{array}$$

et que $Reduc(P, reduc_m) = \{abb, ba, bb\}$. De plus *babbbabb* est une solution optimale de Multi-SLS pour (P, m).

On pose $Set(P, m) = \{(s_1, 1), \dots, (s_1, m(s_1)), (s_2, 1), \dots, (s_n, m(s_n))\}$. Soit τ une permutation sur $Set(P, m)$. On note alors la superchaîne construite en fusionnant les mots dans l'ordre de la permutation avec leurs chevauchements maximaux par

$$linear(P, m, \tau) = \tau((s_1, 1))[1] \oplus \tau((s_1, 2))[1] \oplus \dots \oplus \tau((s_1, m(s_1)))[1] \oplus \dots \oplus \tau((s_n, m(s_n)))[1]$$

où pour $\tau((s_i, j)) = (s_k, l)$, on pose $\tau((s_i, j))[1] = s_k$.

Exemple 5.26 Soient $P = \{abb, ba, b\}$ et l'application m telle que $m(abb) = 2$, $m(bb) = 1$ et $m(ba) = 2$. On a alors que $Set(P, m) = \{(abb, 1), (abb, 2), (bb, 1), (ba, 1), (ba, 1)\}$. Soit τ la permutation de $Set(P, m)$ telle que

$$\begin{array}{rccc} \tau & : & (abb, 1) & \mapsto & (ba, 2) \\ & & (abb, 2) & \mapsto & (abb, 1) \\ & & (bb, 1) & \mapsto & (bb, 3) \\ & & (ba, 1) & \mapsto & (ba, 1) \\ & & (ba, 2) & \mapsto & (abb, 2) \end{array}$$

On a alors que $linear(P, m, \tau) = ba \oplus abb \oplus bb \oplus ba \oplus abb = babbbabb$.

Il est facile de voir que le problème SLS est un cas particulier de Multi-SLS dans le cas où l'application qui donne la multiplicité est l'application de P vers $\{1\}$ (pour tout $x \in P$, $m(x) = 1$). Dans ce cas, en reprenant la notation de la partie 2.1.1, on constate que l'ensemble $Factor-free(P)$ est égal à $Reduc(P, reduc_m)$. De plus pour σ une permutation de $\{1, \dots, n\}$, en posant τ la permutation de $Set(Reduc(P, reduc_{m_1}), reduc_{m_1})$ telle que $\tau((s_i, 1)) = (s_{\sigma(i)}, 1)$ et $P' = Reduc(P, reduc_{m_1})$, on a que

$$linear(P', reduc_{m_1}, \tau) = P'_l(\sigma)$$

On va alors généraliser les résultats de SLS et notamment on a que pour toute solution optimale w de Multi-SLS de (P, m), il existe une permutation τ de l'ensemble $Set(Reduc(P, reduc_m), reduc_m)$ telle que $w = linear(Reduc(P, reduc_m), reduc_m, \tau)$.

On peut alors redéfinir Multi-SLS en utilisant des permutations.

Problèmme 5.27 ***Multi Shortest Linear Superstring (*Multi-SLS*) version 2*** Soient P un ensemble de mots et m une application de P vers $\mathbb{N}^*$ qui donnera une multiplicité pour chaque mot de P. On cherche un mot linéaire w tel qu'il existe une permutation τ de $Set(Reduc(P, reduc_m), reduc_m)$ telle que $w = linear(Reduc(P, reduc_m), reduc_m, \tau)$, qui soit de longueur minimale.

On peut alors se demander quels sont les liens entre les problèmes sur les graphes et les problèmes Multi-SLS, Multi-SCS et Multi-SCCS. On va pour cela définir une nouvelle version du graphe des chevauchements.

Définition 5.28 ***Graphe des chevauchements multiple (voir Figure 5.6)*** Le *graphe des chevauchements multiple* de (P, m) est le graphe orienté complet $G = (V, A, w)$ où

$$V := Set(Reduc(P, reduc_m), reduc_m)$$

et

$$\begin{array}{rccc} w : & A & \to & \mathbb{N} \\ & ((s_i, j), (s_k, l)) & \mapsto & |ov(s_i, s_k)| \end{array}$$

Dés lors, on déduit que chercher un chemin hamiltonien de poids maximum sur le graphe des chevauchements multiple de (P, m) est alors équivalent à trouver une solution optimale de Multi-SLS.

En reprenant les résultats du théorème 4.23 (page 100), on a alors le théorème suivant :

Théorème 5.29 Soient P un ensemble de mots, m une application de P vers $\mathbb{N}$ et G le graphe des chevauchements multiple de (P, m).

1. L'algorithme glouton défini pour Max-HDP et appliqué sur G a un ratio d'approximation optimal de $\frac{1}{2}$ pour Multi-SLS$_c$,
2. L'algorithme glouton défini pour Max-HDC et appliqué sur G a un ratio d'approximation optimal de $\frac{1}{2}$ pour Multi-SCS$_c$,
3. L'algorithme glouton défini pour Max-HDCC et appliqué sur G a un ratio d'approximation optimal de 1 pour Multi-SCCS$_c$,

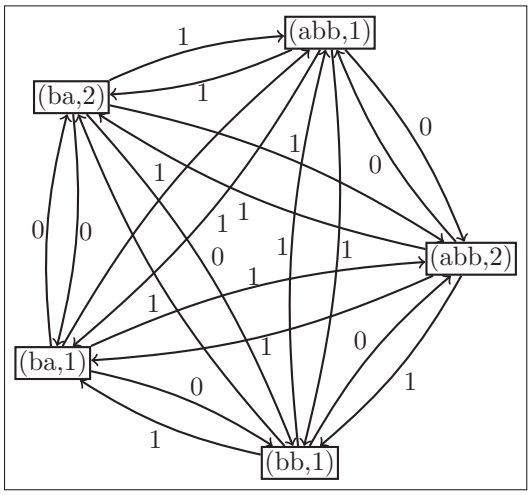


FIGURE 5.6 – Exemple de graphe des chevauchements multiple pour $P = \{abb, ba, b\}$ et m telle que $m(abb) = 2$, $m(bb) = 3$ et $m(ba) = 2$.

Grâce au théorème 5.29, on a le corollaire suivant :

Corollaire 5.30 Pour un ensemble de mots P et une application m de P vers $\mathbb{N}^*$, l'algorithme glouton défini pour Max-HDCC et appliqué sur le graphe des chevauchements multiple de (P, m) est optimal pour Multi-SCCS.

Conclusion de la section On a montré dans cette section, que les problèmes Multi-SLS$_c$ et Multi-SCS$_c$ qui sont **NP-difficiles** sont aussi **APX**. En effet, on a trouvé pour chacun de ces problèmes un algorithme avec un ratio d'approximation optimal de $\frac{1}{2}$. Enfin, on a que l'algorithme glouton pour le problème Multi-SCCS est optimal et donc le problème est dans **P**.

5.3 Chevauchements interdits

Dans cette partie, on va s'intéresser aux problèmes de superchaînes où on va contraindre la solution en empêchant certains chevauchements. En effet, par exemple pour SLS, on sait que l'on peut se restreindre à regarder les superchaînes générées par une permutation et donc on a un ordre sur les mots de l'instance et on peut vouloir éviter certains chevauchements entre deux mots successifs dans la solution.

Soit $P := \{s_1, \ldots, s_n\}$ un ensemble de mots.

Soit w un élément de $PSLS(P)$, on a alors qu'il existe σ une permutation de $\{1, \ldots, n\}$ telle que $w = P_l(\sigma)$. On pose alors $Overlap(w) := \{ov(s_{\sigma(i)}, s_{\sigma(i+1)}) \mid i \in \{1, \ldots, n-1\}\}$ l'ensemble des chevauchements vu par deux mots de l'instance consécutifs dans w. La restriction aux solutions issues de permutations permet de connaître l'ensemble des chevauchements utilisés par une solution. On peut donc maintenant contraindre cet ensemble en introduisant un ensemble F donnant les chevauchements qui sont interdits.

On va pouvoir alors définir le problème de superchaîne linéaire contraint.

Problèmme 5.31 Contraint-SLS Soient P et F deux ensembles de mots, on cherche w une superchaîne linéaire de $PSLS(P)$ qui soit de longueur minimale et telle que $Overlap(w) \cap Fact(F) = \emptyset$.

On peut alors définir une version de Contraint-SLS où on cherche une superchaîne circulaire et une version où on recherche une couverture circulaire de mots. Pour cela, pour un ensemble de mots P, on va définir l'application $Overlap(.)$ sur $PSCS(P)$ et sur $PSCCS(P)$.

Soit $\langle w \rangle$ un élément de $PSCS(P)$. On a alors qu'il existe σ une permutation circulaire de $\{1, \ldots, n\}$ telle que $\langle w \rangle = Circular(P, \sigma)$. On pose alors $Overlap(\langle w \rangle) := \{ov(s_i, s_{\sigma(i)}) \mid i \in \{1, \ldots, n\}\}$. Soit C un élément de $PSCCS(P)$. On a alors qu'il existe σ une permutation de $\{1, \ldots, n\}$ telle que $C = CC(P, \sigma)$. On pose alors $Overlap(C) := \{ov(s_i, s_{\sigma(i)}) \mid i \in \{1, \ldots, n\}\}$.

On peut alors définir le problème Contraint-SCS (respectivement Contraint-SCCS) qui pour deux ensembles de mots P et F, cherche une des plus petites superchaînes circulaires $\langle w \rangle$ (respectivement couvertures circulaires de mots C) de P telles que $Overlap(\langle w \rangle) \cap Fact(F) = \emptyset$ (respectivement $Overlap(C) \cap Fact(F) = \emptyset$).

Notons que si on prend deux éléments x et y de F tels que y est une sous-chaîne de x, pour toute superchaîne linéaire, superchaîne circulaire ou couverture circulaire de mots z, le fait d'avoir $Overlap(z) \cap Fact(\{x\}) = \emptyset$ implique que $Overlap(z) \cap Fact(\{y\}) = \emptyset$. On peut donc à partir de maintenant supposer que F est factor-free. En effet, pour P et F deux ensembles de mots, on a alors que $OPT_{Contraint-SCS}(P, F) = OPT_{Contraint-SCS}(P, Factor - free(F))$. De plus, pour une superchaîne linéaire, superchaîne circulaire ou couverture circulaire de mots z de P, on a que si $P \cap F \neq \emptyset$, on a $Overlap(z) \cap Fact(F) \neq \emptyset$ et donc z n'est pas une solution du problème. Pour la suite, on prendra F un sous-ensemble de $Fact(P) \setminus P$.

Malheureusement, pour certaines instances de P et de F, Contraint-SLS, Contraint-SCS ou même Contraint-SCCS n'ont aucune solution (voir Exemple 5.34).

On va alors définir deux versions de Contraint-SLS où dans la première (Contraint-SMCS) on cherche une couverture mixte de mots et dans la seconde (Contraint-SLCS), on cherche une couverture linéaire de mots. Pour cela on va généraliser l'application $Overlap(.)$ à ces deux cas en reprenant les notations de $Presque - Coupe_\sigma$ et de $Coupe_\sigma$ que l'on a vu dans les sous-sections 2.4.2 (page 51) et 2.4.1 (page 49).

Soit M un élément de $PSMCS(P)$, on a alors qu'il existe σ une permutation de $\{1, \ldots, n\}$ et $E \in Presque - Coupe_\sigma$ tels que $M = MC(P, \sigma, E)$. On pose alors $Overlap(M) := \{ov(s_i, s_{\sigma(i)}) \mid i \in \{1, \ldots, n\} \setminus E\}$. De même pour $L \in PSLCS(P)$, il existe σ une permutation de $\{1, \ldots, n\}$ et $E \in Coupe_\sigma$ tels que $L = LC(P, \sigma, E)$. On pose alors $Overlap(L) := \{ov(s_i, s_{\sigma(i)}) \mid i \in \{1, \ldots, n\} \setminus E\}$.

On peut alors définir les problèmes Contraint-SMCS et Contraint-SLCS.

Problèmme 5.32 Contraint-SMCS Soient P et F deux ensembles de mots, on cherche une couverture mixte de mots M de $PSMCS(P)$ qui soit de norme minimale et telle que $Overlap(M) \cap Fact(F) = \emptyset$.

Problèmme 5.33 Contraint-SLCS Soient P et F deux ensembles de mots, on cherche une couverture linéaire de mots L de $PSLCS(P)$ qui soit de norme minimale et telle que $Overlap(L) \cap Fact(F) = \emptyset$.

On a alors que si $F \subseteq Fact(P) \setminus P$, l'ensemble P lui-même est une couverture mixte de mots de P et une couverture linéaire de mots de P, on a donc $OPT_{Contraint-SMCS}(P) \neq \emptyset$ et $OPT_{Contraint-SLCS}(P) \neq \emptyset$.

Exemple 5.34 Soient $P = \{abba, baab, bcb\}$ et $F = \{b\}$. On a alors que
- $OPT_{Contraint-SLS}(P, F) = \emptyset$,
- $OPT_{Contraint-SCS}(P, F) = \emptyset$,
- $OPT_{Contraint-SCCS}(P, F) = \emptyset$,
- $OPT_{Contraint-SLCS}(P, F) = \{\{abbaab, bcb\}, \{baabba, bcb\}\}$,
- $OPT_{Contraint-SMCS}(P, F) = \{< abba >, bcb\}$.

On sait que pour toute permutation σ de $\{1, \dots, n\}$, on peut obtenir la couverture circulaire de mots issue de σ. Si $Overlap(CC(P, \sigma)) \cap F = \emptyset$, alors $CC(P, \sigma)$ est une couverture circulaire de mots de P qui satisfait F et donc une couverture mixte de mots de P qui satisfait F. Mais si $Overlap(CC(P, \sigma)) \cap F \neq \emptyset$, on ne peut pas utiliser directement $CC(P, \sigma)$, on va devoir enlever les chevauchements $Overlap(CC(P, \sigma)) \cap F$ de chaque composante. L'idée ici, est de définir des couvertures mixtes de mots et des couvertures linéaires de mots issues d'une permutation et d'un ensemble P qui seront générées par la permutation et l'ensemble des chevauchements interdits F.

Soit σ_c une permutation circulaire de $\{1, \dots, n\}$. On pose

$$Violation(P, F, \sigma_c) := \{i \in \{1, \dots, n\} \mid \exists f \in F \text{ tel que } ov(s_i, s_{\sigma_c(i)}) \underset{\text{sub}}{\subset} f\}.$$

Soit σ_c une permutation circulaire de $\{1, \dots, n\}$. La permutation circulaire σ_c est dite *cohérente* si $|Violation(P, F, \sigma_c)| \leq 1$. On pose

$$CMC(P, F, \sigma_c) := \begin{cases} Circular(P, \sigma_c) & \text{si } |Violations(P, F, \sigma_c)| = 0, \\ Linear(P, \sigma_c, i) & \text{si } Violations(P, F, \sigma_c) = \{i\}. \end{cases}$$

et

$$CLC(P, F, \sigma_c) := \begin{cases} Linear(P, \sigma_c, j) & \text{si } |Violations(P, F, \sigma_c)| = 0 \text{ et} \\ & ov(s_j, s_{\sigma_c(j)}) \text{ est le plus petit} \\ & \text{des chevauchements utilisés par } \sigma_c, \\ Linear(P, \sigma_c, i) & \text{si } Violations(P, F, \sigma_c) = \{i\}. \end{cases}$$

Soit σ une permutation cohérente de $\{1, \dots, n\}$, c'est-à-dire telle que chaque permutation circulaire σ_i de sa décomposition en permutation circulaire $\sigma = \sigma_1 \dots \sigma_m$ est cohérente. On pose alors

$$CMC(P, F, \sigma) := \{CMC(P_1, F, \sigma_1), \dots, CMC(P_m, F, \sigma_m)\}$$

et

$$CLC(P, F, \sigma) := \{CLC(P_1, F, \sigma_1), \dots, CLC(P_m, F, \sigma_m)\}$$

où $Part_\sigma = \{P_1, \dots, P_m\}$ est tel que pour tout $i \in \{1, \dots, m\}$, P_i est l'élément de $Part_\sigma$ correspondant à σ_i.

Soient $Ov(P)$ l'ensemble de tous les chevauchements maximaux de P ($Ov(P) := \{ov(s_i, s_j) \mid i, j \in \{1, \dots, n\}\}$) et $Ov^*(P, F) = Ov(P) \setminus Fact(F)$. On peut alors définir

Algorithme 5.2 : L'algorithme glouton pour Contraint-SMCS.

Entrée : Soient P un ensemble de mots et $F \subseteq Fact(P)$.

$C \leftarrow \emptyset$;
while $Ov^*(P, F) \neq \emptyset$ **do**
  Soit u et v deux mots de P dont le chevauchement maximal est le plus grand et tels que $ov(u, v) \notin Fact(F)$;
  $P \leftarrow P \setminus \{u, v\}$;
  if $u = v$ **then** $C \leftarrow C \cup \{u \oplus v\}$;
  else $P \leftarrow P \cup \{u \oplus v\}$;
return $C \cup P$.

Algorithme 5.3 : L'algorithme glouton pour Contraint-SLCS.

Entrée : Soient P un ensemble de mots et $F \subseteq Fact(P)$.

while $Ov^*(P, F) \neq \emptyset$ **do**
  Soit u et v deux mots **différents** de P dont le chevauchement maximal est le plus grand et tels que $ov(u, v) \notin Fact(F)$;
  $P \leftarrow P \cup \{u \oplus v\} \setminus \{u, v\}$;
return P.

un algorithme glouton pour Contraint-SMCS (voir Algorithme 5.2) et pour Contraint-SLCS (voir Algorithme 5.3).

On va maintenant définir une nouvelle version du graphe des chevauchements, le graphe des chevauchements contraint.

Définition 5.35 ***Graphe des chevauchements contraint (voir figure 5.7)*** Le *graphe des chevauchements contraint* de (P, F) qui est le graphe orienté pondéré $G := (V, A, w)$ tel que

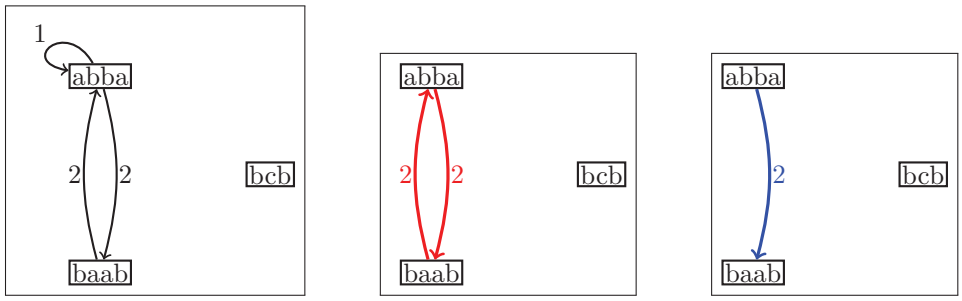
$$\begin{array}{rcl} V & := & P \\ A & := & \{(u, v) \in P \times P \mid ov(u, v) \in Ov^*(P, F)\} \end{array}$$

$$\begin{array}{rccc} w : & A & \to & \mathbb{N} \\ & (u, v) & \mapsto & |ov(u, v)| \end{array}$$

On obtient alors que les algorithmes 5.3 et 5.2 sont équivalents aux algorithmes gloutons pour SLS et SCCS où on aurait arrêté l'algorithme avant la fin (voir section 4.1). On obtient alors le théorème suivant car on peut reprendre les systèmes héréditaires de Max-HDP et Max-HDCC et on a la même extensibilité pour chacun (pour Contraint-SLCS$_c$ et pour Max-HDP et pour Contraint-SMCS$_c$ et pour Max-HDCC).

Théorème 5.36 Soient P un ensemble de mots, $F \subseteq Fact(P) \setminus P$ et G le graphe des chevauchements contraint de (P, F).

1. L'algorithme glouton défini pour Max-HDP et appliqué sur G a un ratio d'approximation optimal de $\frac{1}{3}$ pour Contraint-SLCS$_c$,
2. L'algorithme glouton défini pour Max-HDCC et appliqué sur G a un ratio d'approximation optimal de $\frac{1}{2}$ pour Contraint-SMCS$_c$.



(a) Graphe des chevauchements contraint pour P

(b) Solution optimale de Contraint-SMCS sur G

(c) Solution optimale de Contraint-SLCS sur G

FIGURE 5.7 – Exemple de graphes des chevauchements contraints G pour $P = \{abba, baab, bcb\}$ et $F = \{b\}$.

Conclusion de la section Dans cette section, on s'est intéressé aux problèmes Contraint-SLCS$_c$ et Contraint-SMCS$_c$ et on a montré que l'algorithme glouton avait des ratios d'approximation optimaux de $\frac{1}{3}$ et $\frac{1}{2}$ pour ces problèmes.

5.4 Problèmes avec fusion des contraintes

On s'est intéressé aux problèmes des superchaînes où on regardait le cas du complémentaire-renversé (voir sous-section 5.1.3), où on ajoutait une multiplicité sur les mots en entrée (voir section 5.2) et où on interdisait des chevauchements (voir section 5.3). On peut alors se demander ce qu'il se passe si on fusionne les problèmes. Par exemple, la variante SLS où on regarde le reverse-complement et la multiplicité est alors le problème suivant :

Problèmme 5.37 Multi-SDLS Soient $P := \{s_1, \dots, s_n\}$ un ensemble de mots ADN-factor-free sur $\{A, T, C, G\}$ et m une application de P vers $\mathbb{N}^*$ qui donnera une multiplicité pour chaque mot de P. Le problème Multi-SDLS cherche un mot linéaire w qui soit de longueur minimale et tel que pour tout $s_i \in P$, $|Occ(s_i, w)| + |Occ(\overleftarrow{s_i}, w)| \geq m(s_i)$.

En reprenant la preuve du théorème 5.22, on obtient le théorème suivant sur la version compression de Multi-SDLS :

Théorème 5.38 Soient P un ensemble de mots, m une application de P dans $\mathbb{N}$ et G le graphe $Graph\_Part(\bigcup_{w\in P} \cup_{i=1}^{reduc_m(w)} \{\{(w,i),(\overleftarrow{w},i)\}\}, f)$ où f est l'application telle que $f((u,i),(v,j)) = |ov(u,v)|$.

L'algorithme glouton défini pour Max-PHP et appliqué sur le graphe G a un ratio d'approximation optimal de $\frac{1}{2}$ pour Multi-SDLS$_c$.

On peut alors ainsi fusionner les problèmes précédents et trouver le ratio d'approximation optimal pour ces problèmes. La figure 5.8 nous donne des bornes inférieures des ratios d'approximation optimaux.

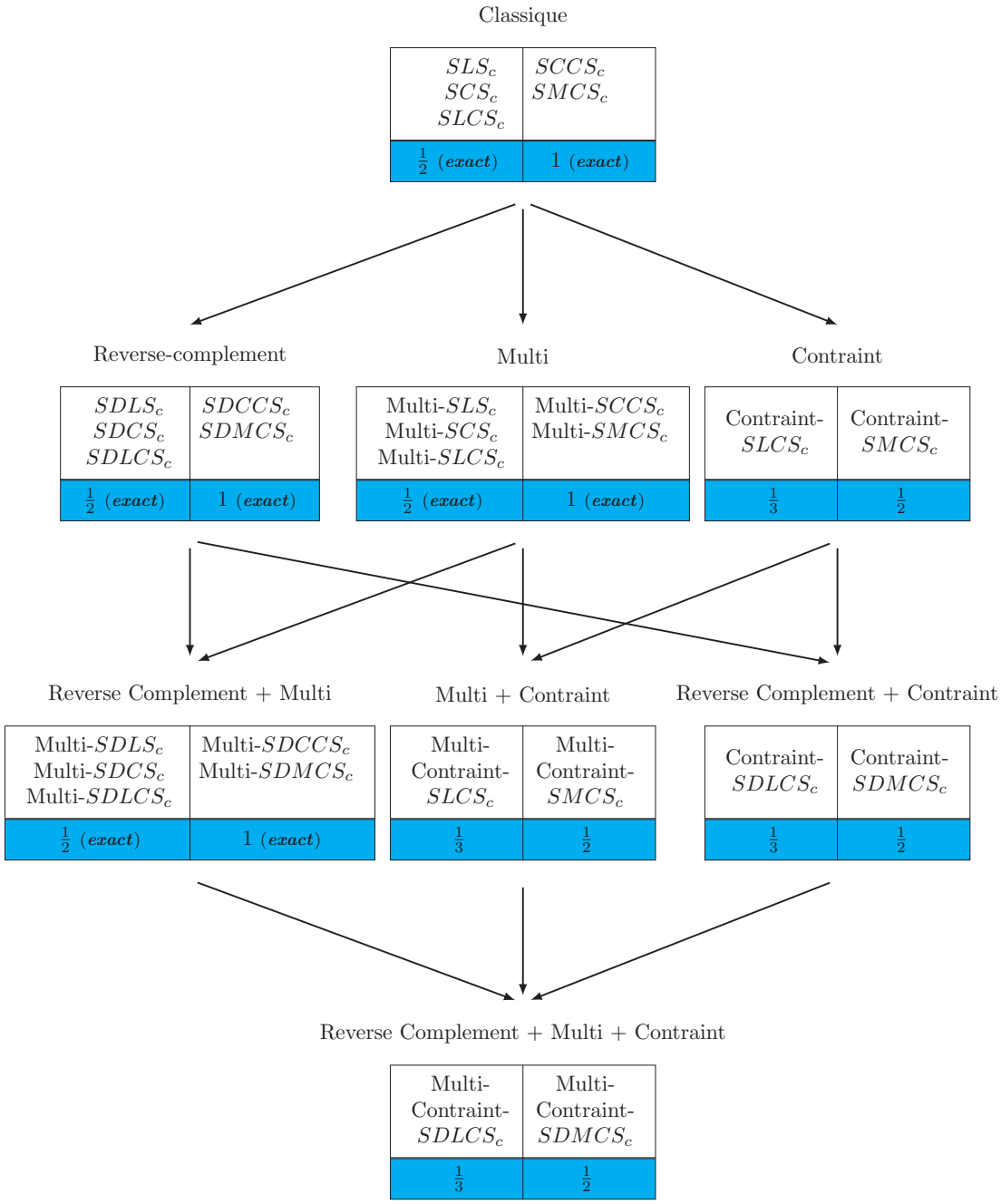


FIGURE 5.8 – Figure récapitulative des bornes inférieures des ratios d'approximation optimaux des algorithmes gloutons des problèmes définis dans ce chapitre et de leur fusion. Dans le cas où la borne est exacte, on le note juste à coté de la valeur de la borne.

Conclusion de la section Dans cette section, on a montré que l'on pouvait combiner les problèmes que l'on a vu jusqu'alors. Pour chacun de ces problèmes combinés, on a donné une borne inférieure sur le ratio d'approximation optimal de l'algorithme glouton.

Conclusion du chapitre On a mis en évidence dans ce chapitre des problèmes qui résument certaines difficultés que l'on peut rencontrer en pratique lors de l'assemblage. En effet, le problème Contraint-SLS peut être appliqué dans le cas où l'ensemble des chevauchements interdits est l'ensemble de toutes les sous-chaînes de l'instance de longueur plus petite qu'une borne. Cette borne correspondrait alors à ne pas regarder les chevauchements qui sont inférieurs à cette borne, pour éviter de prendre des chevauchements dont on n'est pas sûr. De même, le problème Multi-SLS a une utilité pratique, car lors de l'assemblage, chaque mot donné par le séquenceur a une couverture, c'est-à-dire un nombre de fois que le séquenceur va nous donner ce mot. Si la couverture d'un mot est bien supérieure à la moyenne, on est tenté de vouloir utiliser cette information en trouvant la superchaîne où ce mot apparaît plusieurs fois.

On a montré que l'étude que l'on a réalisée pour les algorithmes glouton des problèmes SLS_c, SCS_c et SCCS_c peut être appliquée à d'autres problèmes sur les superchaînes et ainsi obtenir des bornes inférieures sur les ratios d'approximation optimaux des algorithmes gloutons pour ces problèmes.

Dans le chapitre 8, on va montrer que pour les variantes de SCCS, on est capable de construire l'algorithme glouton en temps linéaire en la norme de l'instance.

Troisième partie

Construction de l'algorithme glouton pour SCCS et ses variantes ainsi que pour le graphe de De Bruijn

Chapitre 6

Arbres d'indexation

Dans ce chapitre, on va s'intéresser à deux structures d'indexation sous forme d'arbres très connues : l'arbre d'Aho-Corasick et l'arbre des suffixes. On va ensuite regarder plus en détail comment tronquer ces structures d'indexation. En effet, dans certains cas, avoir tout l'arbre ne nous intéresse pas ; par exemple dans la section 9.3, lors de la construction du graphe de De Bruijn, on peut se permettre de perdre l'information des sous-chaînes plus grandes que l'ordre du graphe de De Bruijn voulu.

Ce chapitre est organisé comme suit : dans les sections 6.1 et 6.2, on donne la définition et notre représentation de l'arbre d'Aho-Corasick et de l'arbre des suffixes. Dans la section 6.3, on définit une condition simple qu'un ensemble de mots doit satisfaire pour construire une structure d'indexation généralisée et l'ébauche d'une modification de l'algorithme de McCreight [58] pour le construire. Enfin dans la section 6.4, on introduit les arbres des suffixes tronqués et on présente un nouvel arbre des suffixes tronqués qui nous sera utile pour construire le graphe de De Bruijn dans la section 9.3. Les résultats sur la généralisation de l'arbre des suffixes ainsi que sur l'arbre des suffixes tronqué ont été publiés dans (VII) et (II).

Sommaire

6.1 Arbre d'Aho-Corasick

On va donner dans cette section une rapide description de l'arbre d'Aho-Corasick et de son utilité en général. Il est important de noter que l'arbre d'Aho-Corasick est connu depuis 1975 où il a été défini par Aho et al. [1]. Cette section n'a pas pour but de donner une nouvelle définition de l'arbre d'Aho-Corasick mais de bien le définir car cet arbre aura une utilité pour nous dans la section 7.2 où on va généraliser cette structure. On verra de plus dans la section suivante (section 6.2), le lien qu'il y a entre l'arbre d'Aho-Corasick et l'arbre des suffixes.

L'arbre d'Aho-Corasick est à la base d'un algorithme de recherche d'occurrences d'un ensemble de mots dans un texte. L'idée est de pré-traiter l'ensemble des mots et de les mettre sous forme d'arbre pour ensuite parcourir simultanément le texte et l'arbre.

On peut alors définir l'arbre d'Aho-Corasick de manière formelle :

Définition 6.1 ***Arbre d'Aho-Corasick (voir Figure 6.1)*** Soit P un ensemble de mots, l'*arbre d'Aho-Corasick* de P, noté $AC(P)$, est le graphe orienté (V, A) tel que :

$$V := \{w \mid w \text{ est un préfixe d'un élément } s_i \text{ de } P\}$$

$$A := \{(u, v) \in V \times V \mid u \text{ est le plus grand préfixe de } v \text{ dans } V\}$$

Remarque 6.2 En réalité l'arbre d'Aho-Corasick que l'on vient de définir est appelé un *trie*. Ce qu'on appelle habituellement arbre d'Aho-Corasick est l'automate correspondant à la fusion des deux arbres que sont le trie que l'on vient de définir et l'arbre des liens d'échec que l'on va définir après. On a fait ici la séparation de l'automate en deux arbres pour permettre une meilleure compréhension. Par la suite, on continuera d'assimiler l'arbre d'Aho-Corasick au trie résultant de son automate.

Comme pour tout $v \in V \setminus \{\varepsilon\}$, il existe un unique u tel que $(u, v) \in A$, on a bien que $AC(P)$ forme un arbre où u est le parent de v et v est un enfant de u.

Dans la suite du mémoire de thèse, comme un nœud de l'arbre d'Aho-Corasick de P correspond à un préfixe d'un mot de P, on confondra le nœud de l'arbre et le mot qu'il représente.

Remarque 6.3 Pour un mot de P sur un alphabet Σ, tous les préfixes des mots de P sont des nœuds de $AC(P)$, on a donc que pour tout arc (u, v) de $AC(P)$, il existe $a \in \Sigma$ tel que $u.a = v$. Pour représenter $AC(P)$, la plupart du temps on utilise des nœuds non labellisés mais pour chaque arc (u, v), on met une étiquette sur l'arc qui correspond à la lettre a où $u.a = v$ (voir Figure 6.1a).

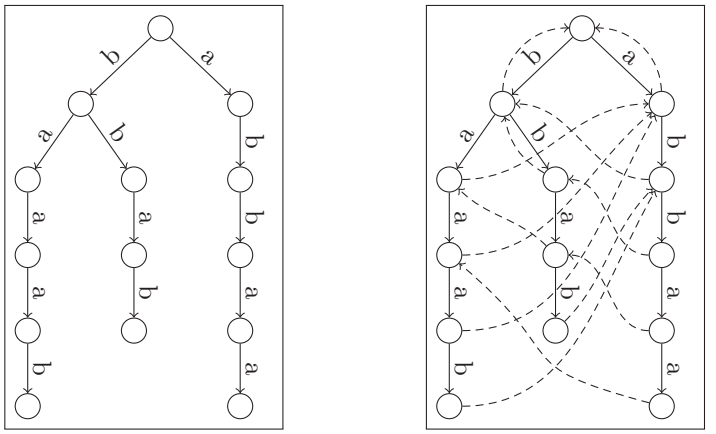
Dans la construction de l'arbre d'Aho-Corasick, on ajoute aussi des *liens d'échec* qui sont l'application le de V dans V telle que pour un nœud v, $le(v)$ est le plus grand suffixe de v qui est dans V (voir Figure 6.1b).

Définition 6.4 Posons le graphe (V, B) où V sont les nœuds de $AC(P)$ et

$$B := \{(u, v) \in V \times V \mid le(u) = v\}$$

On a alors que le graphe (V, B) est un arbre que l'on appelle *arbre des liens d'échec* de P et que l'on note $LE(P)$.

L'algorithme d'Aho-Corasick construit l'arbre d'Aho-Corasick en temps linéaire en la norme de l'instance ainsi qu'un algorithme pour rechercher les occurrences d'un ensemble de mots P dans un texte w. On commence par construire l'arbre d'Aho-Corasick de l'ensemble de mots P. Une fois cet arbre créé, on parcourt simultanément le texte w et l'arbre. On commence en mettant un pointeur sur le début du texte et un autre pointeur sur la racine de l'arbre. Après, on commence à regarder la première lettre du texte. Si elle correspond à une étiquette sur un des arcs entre la racine, on va sur l'enfant, sinon on prend le lien d'échec. Et après on recommence avec la position suivante. Si à un moment on passe par un mot de P, on marque cette position comme étant une occurrence de ce mot.



(a) Arbre d'Aho-Corasick sans liens d'échec

(b) Arbre d'Aho-Corasick avec liens d'échec

FIGURE 6.1 – Exemple d'arbres d'Aho-Corasick pour $P := \{baaab, bbab, abbaa\}$.

L'intérêt de cet algorithme est sa complexité : il est en temps linéaire en la taille des mots et du texte. En outre, c'est un des premiers algorithmes qui permet de trouver les itérations d'un ensemble de mots sur un texte sans dépendre, à chaque étape de la recherche, du nombre de mots.

On va s'intéresser à cet arbre, car l'information utile pour trouver le chevauchement entre deux mots est comprise dans l'arbre d'Aho-Corasick. En effet, un chevauchement w d'un mot u sur un mot v est un préfixe de v, donc un nœud de l'arbre d'Aho-Corasick. De plus, w est un suffixe de u, on peut alors trouver une chaîne de liens d'échec de u jusqu'à w. On a alors que u est dans le sous-arbre de w de l'arbre des liens d'échec et v est dans le sous-arbre de w de l'arbre d'Aho-Corasick.

Pour finir, on va regarder quelques propriétés de l'arbre d'Aho-Corasick qui sont connues et évidentes mais que l'on va formaliser.

Proposition 6.5 Soient P un ensemble de mots, $AC(P) = (V, A_C)$ l'arbre d'Aho-Corasick de P et $LE(P) = (V, A_L)$ l'arbre des liens d'échec de P. On a alors que :

— $|A_C| = |A_L| = |V| - 1$,

— $max(\{|w| \mid w \in P\}) + 1 \leq |V| \leq ||P|| + 1$.

Les bornes de la proposition 6.5 sont strictes (voir Exemple 6.7). La borne inférieure de $|V|$ vient du fait que chaque préfixe de chaque mot u de P est dans V, c'est-à-dire $|u| + 1$ préfixes en comptant le mot vide qui correspond à la racine de l'arbre d'Aho-Corasick de P. La borne supérieure de $|V|$ provient du fait qu'au maximum chaque préfixe de chaque mot de P correspond à un nœud différent de l'arbre d'Aho-Corasick de P et donc on a au maximum $||P|| + 1$ nœuds différents en comptant la racine qui correspond au mot vide.

Remarque 6.6 Même si les bornes de V sont strictes, elles peuvent être améliorées dans le cas où P est ou n'est pas factor-free.

— Dans le cas d'un ensemble de mots P **non factor-free**,

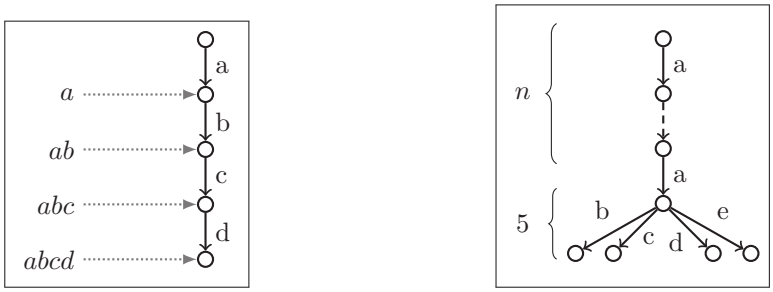
$$max(\{|w| \mid w \in P\}) + 1 \leq |V| \leq ||\mathbf{Factor - free(P)}|| + 1.$$

— Dans le cas d'un ensemble de mots P **factor-free**,

$$max(\{|w| \mid w \in P\}) + |\mathbf{P}| \leq |V| \leq ||P|| + 1.$$

Exemple 6.7 Soit $P = \{a, ab, abc, abcd\}$. On a alors que $max(\{|w| \mid w \in P\}) + 1 = |abcd| + 1 = 5$, que $|V| = 5$ (voir Figure 6.2a) et que $||Factor - free(P)|| + 1 = ||\{abcd\}|| + 1 = 5$.

Soit $P = \{a^n b, a^n c, a^n d, a^n e\}$. On a alors que $max(\{|w| \mid w \in P\}) + |P| = |a^n b| + |\{a^n b, a^n c, a^n d, a^n e\}| = n + 1 + 4 = n + 5$, que $|V| = n + 5$ (voir Figure 6.2b) et que $||P|| + 1 = 4n + 5$.



(a) $P = \{a, ab, abc, abcd\}$ (b) $P = \{a^n b, a^n c, a^n d, a^n e\}$

FIGURE 6.2 – Exemple d'arbres d'Aho-Corasick.

Soit $w = a_1 a_2 \dots a_n$ un mot. On pose w^{-1} le mot reverse de w, c'est-à-dire que $w^{-1} = a_n a_{n-1} \dots a_1$. On généralise la notation à un ensemble de mots P, on a que $P^{-1} = \{w^{-1} \mid w \in P\}$.

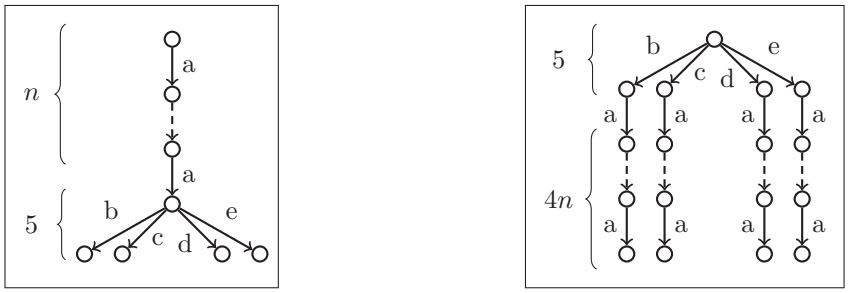
Remarque 6.8 Soit P un ensemble de mots. Il est difficile en général de comparer les tailles de $AC(P)$ et de $AC(P^{-1})$ (voir Exemple 6.9).

Exemple 6.9 Soit $P = \{a^n b, a^n c, a^n d, a^n e\}$. On a alors que $P^{-1} = \{ba^n, ca^n c, da^n, ea^n\}$ et donc pour $AC(P) = (V, A)$ et $AC(P^{-1}) = (V_R, A_R)$, on a que $|V| = n + 5$ (voir Figure 6.3a) et $|V_R| = 4n + 5$ (voir Figure 6.3b).

6.2 Arbre des suffixes

Dans le chapitre précédent, on a présenté rapidement l'arbre d'Aho-Corasick. Dans cette section, on va aborder l'arbre des suffixes, qui peut être vu comme une version modifiée d'un arbre d'Aho-Corasick. Les arbres des suffixes sont des structures d'indexation qui sont connues et souvent utilisées [17, 35]. L'intérêt ici de présenter une nouvelle représentation de l'arbre des suffixes va nous permettre dans le chapitre 9 de simplifier les explications.

Pour commencer, l'arbre des suffixes est une structure d'indexation au même titre que la table des suffixes ou le FM-index. Il est admis que ces structures d'indexation sont



(a) $P = \{a^n b, a^n c, a^n d, a^n e\}$

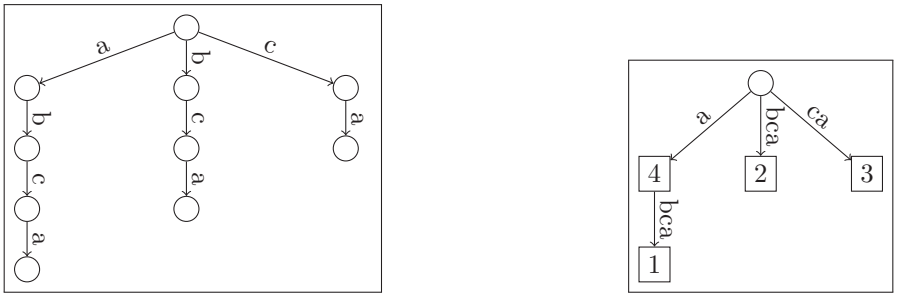
(b) $P^{-1} = \{ba^n, ca^n c, da^n, ea^n\}$

FIGURE 6.3 – Exemples de la différence entre l'arbre d'Aho-Corasick d'un ensemble de mots P et de P^{-1}.

équivalentes, du point de vu fonctionnel : ce que l'on peut faire avec une, on peut le faire avec à peu près la même complexité avec une autre. On choisit ici l'arbre des suffixes, car c'est sans doute la structure d'indexation la plus connue. En outre, les explications seront plus simples que sur une table LCP d'une table des suffixes.

L'*arbre des suffixes* d'un mot w, noté $ST(w)$, est une structure d'indexation sous forme d'arbre orienté de la racine vers les feuilles où on indexe l'ensemble des suffixes de w. Chaque nœud est représenté par une sous-chaîne de w et entre deux nœuds u et v, l'ancêtre commun de u et v représente le préfixe commun du mot relié à u et du mot relié à v. On peut donner une définition un peu plus formelle en utilisant l'arbre d'Aho-Corasick.

Définition 6.10 ***Arbre des suffixes (voir figure 6.4b)*** L'arbre des suffixes d'un mot w, noté $ST(w)$, est l'arbre $AC(Suff(w))$ où on contracte tous les arcs (u, v) où u n'est pas dans $Suff(w)$ et n'a qu'un seul enfant.



(a) arbre d'Aho-Corasick de $Suff(w)$

(b) arbre des suffixes de w

FIGURE 6.4 – Exemple d'arbres des suffixes pour $w = abca$ et d'arbre d'Aho-Corasick pour $Suff(w)$. Pour la représentation de l'arbre des suffixes on marque sur les nœuds qui correspondent à un suffixe, la position du début du suffixe dans le nœud. Ces nœuds sont représentés par des rectangles alors que les autres nœuds internes sont représentés par des cercles.

Comme pour l'arbre d'Aho-Corasick, on confond la notion de nœud et de mot représenté par un nœud.

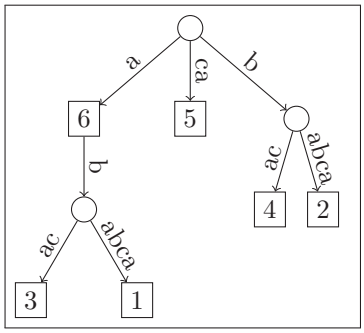
On peut alors encadrer la taille de l'arbre des suffixes d'un mot à l'aide de la taille de l'arbre des suffixes du mot où on ajoute une lettre à la fin qui n'apparaît pas à l'alphabet.

Proposition 6.11 Soient w un mot sur l'alphabet Σ et \$ une lettre qui n'appartient pas à Σ. Pour $ST(w) = (V, A)$ et $ST(w\$) = (V_\$, A_\$)$, on a

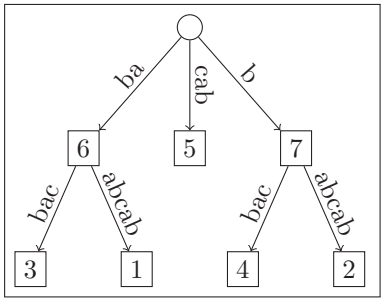
$$|V_\$| - |w| \leq |V| \leq |V_\$|.$$

Remarque 6.12 La proposition 6.11, nous dit que le nombre de nœuds de $ST(w)$ est plus petit ou égal au nombre de nœuds de $ST(w\$)$. On ne peut pas généraliser cette proposition au lien entre la taille de l'arbre des suffixes d'un mot et la taille de l'arbre des suffixes d'une sous-chaîne de ce mot (voir Exemple 6.13).

Exemple 6.13 Soient $w_1 = ababca$ et $w_2 = ababcab$, on a que w_1 est une sous-chaîne de w_2 et que le nombre de nœuds de $ST(w_1)$ est de 9 (voir Figure 6.5a) alors que le nombre de nœuds de $ST(w_2)$ est de 8 (voir Figure 6.5b).



(a) Arbre des suffixes de $w_1 = ababca$



(b) Arbre des suffixes de $w_2 = ababcab$

FIGURE 6.5 – Exemple d'un mot et d'une de ses sous-chaînes où la taille de l'arbre des suffixes du mot est plus petite que la taille de l'arbre des suffixes de sa sous-chaîne.

L'arbre des suffixes que l'on vient de définir n'est pas l'arbre des suffixes classique comme celui défini par Weiner [96]. Comme l'arbre des suffixes classique a la propriété de n'avoir aucun nœud avec un seul fils, le nombre de nœuds de l'arbre est linéaire en le nombre des feuilles. Grâce à la proposition 6.11, on a que l'arbre des suffixes $ST(w)$ est linéaire en $|w|$. Il existe de nombreux algorithmes linéaires pour construire l'arbre des suffixes classique d'un mot w ([96, 58, 92]). On peut en un parcours de l'arbre des suffixes classiques de w, modifier la structure pour obtenir $ST(w)$, on a donc que l'on peut construire $ST(w)$ en temps linéaire en $|w|$.

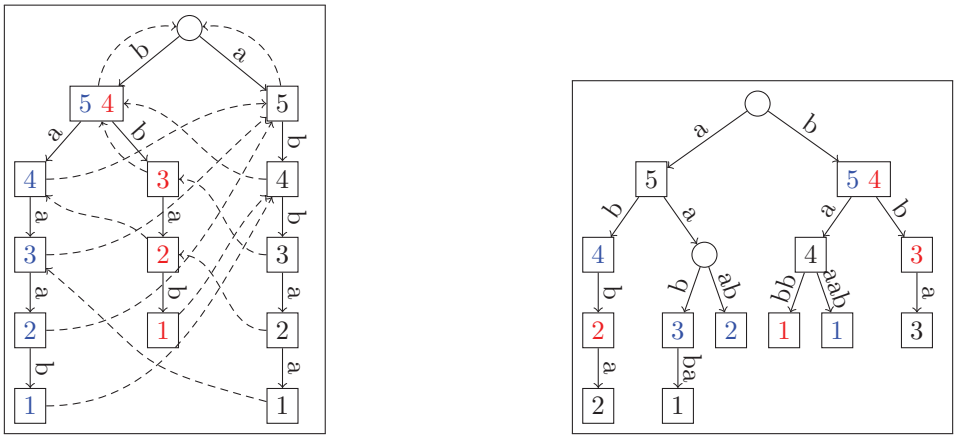
On peut définir de même l'*arbre des suffixes généralisé* d'un ensemble de mots P, noté $GST(P)$ qui correspond à la fusion des arbres des suffixes des différents mots de P. On a alors que $GST(P)$ est de taille linéaire en $||P||$ et on peut le construire en $O(||P||)$.

Pour un ensemble de mots, par la définition de l'arbre des suffixes généralisé, on a que le nombre de nœuds de $GST(P)$ est plus petit ou égal au nombre de nœuds de $AC(\bigcup_{w \in P} Suff(w))$. La proposition suivante va faire de même le lien entre les nœuds de $GST(P)$ et les nœuds de $AC(P^{-1})$. On a vu dans la section 1.7 que pour un mot w, le

mot w^{-1} est le renversé de w et que pour un ensemble de mots P, l'ensemble P^{-1} est l'ensemble de tous les mots renversé des mots de P.

Proposition 6.14 ***voir Figure 6.6*** Soient P un ensemble de mots, $GST(P) = (V_S, A_S)$ l'arbre des suffixes généralisé de P et $AC(P^{-1}) = (V_C, A_C)$ l'arbre d'Aho-Corasick de P^{-1}. On a alors que :

$$V_C^{-1} \subseteq V_S.$$



(a) Arbre d'Aho-Corasick de P^{-1} avec ses liens d'échec (14 nœuds).

(b) Arbre des suffixes de P (15 nœuds).

FIGURE 6.6 – Exemple du lien donné par la proposition 6.14 pour $P = \{baaab, aabba, babb\}$. Dans chaque nœud v de l'arbre des suffixes généralisé, on ajoute l'information de $Position - Suff_P(v)$: on associe à chaque mot de P une couleur unique et le numéro correspond à la position de début du suffixe dans le mot de la même couleur. Dans la mesure du possible, on ajoutera cette information sur chaque arbre des suffixes généralisé.

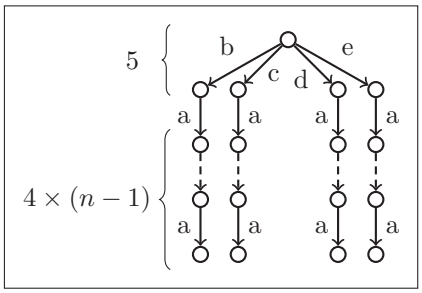
Remarque 6.15 La proposition 6.14 nous dit que le nombre de nœuds de $AC(P^{-1})$ est inférieur au nombre de nœuds de $GST(P)$ et ce pour tout ensemble de mots P. On ne peut pas faire le même lien entre $GST(P)$ et $AC(P)$ (voir Exemple 6.16).

Exemple 6.16 Soit $P = \{ba^n, ca^n, da^n, ea^n\}$. On a alors que $GST(P)$ a $n+4$ nœuds (voir Figure 6.7b) et que $AC(P)$ a $4n+1$ nœuds (voir Figure 6.7a).

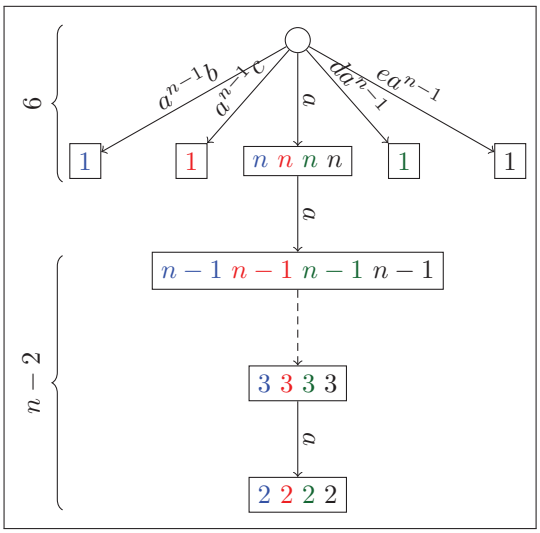
Le choix d'utiliser cette définition de l'arbre des suffixes généralisé est motivé par le fait que dans la définition classique, en ajoutant une lettre unique à la fin de chaque mot, la taille de l'alphabet est alors au moins linéaire en le nombre de mots de P.

Pour ne pas perdre d'information, pour un ensemble de mots P et pour chaque nœud v du $GST(P)$, on pose $Position - Suff_P(v)$ l'ensemble des paires (i, j) telles que le nœud v est le suffixe de s_i commençant à la position j (voir Figure 6.6b).

De la même manière qu'on a défini le lien d'échec sur l'arbre d'Aho-Corasick, on définit les liens suffixes. Pour un ensemble de mots P, le *lien suffixe* d'un nœud v du $GST(P)$, noté $ls(v)$, est le nœud u du $GST(P)$ tel que u est le suffixe de v de longueur $|v| - 1$.



(a) Arbre d'Aho-Corasick de P



(b) Arbre des suffixes de P

FIGURE 6.7 – Exemple d'un ensemble de mots $P = \{ba^n, ca^n, da^n, ea^n\}$ où la taille de l'arbre des suffixes de P est plus petite que la taille de son arbre d'Aho-Corasick.

Remarque 6.17 Dans la définition classique de l'arbre des suffixes, les liens suffixes sont définis essentiellement sur les nœuds internes de l'arbre. Dans notre définition, on définit les liens suffixes sur l'ensemble des nœuds, les feuilles de l'arbre comprises.

On peut alors définir l'arbre des liens suffixes.

Définition 6.18 ***voir Figure 6.8b*** Soit P un ensemble de mots. L'*arbre des liens suffixes* de P, noté $LS(P)$ est l'arbre (V, B) où V l'ensemble des nœuds de $GST(P)$ et

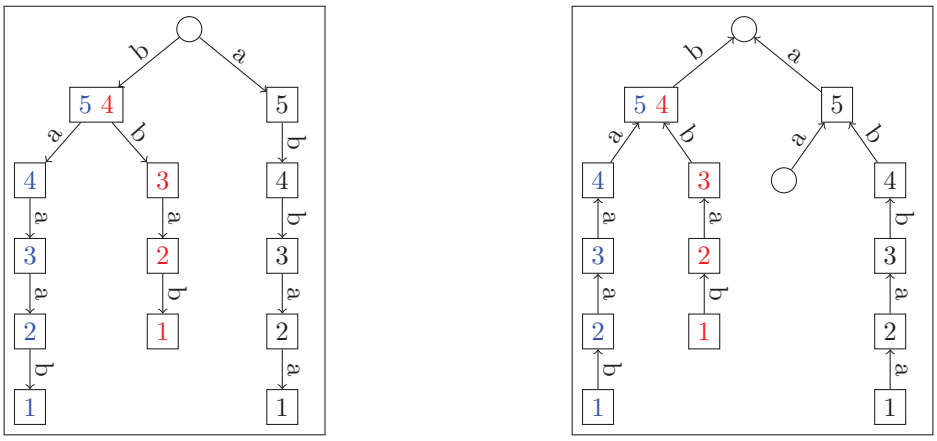
$$B := \{(u, v) \in V \times V \mid ls(u) = v\}.$$

La proposition 6.14, lie les nœuds de $GST(P)$ et les nœuds de $AC(P^{-1})$; on va aller un peu plus loin en exprimant le lien entre $AC(P^{-1})$ et $LS(P)$.

Proposition 6.19 ***voir Figure 6.8*** Soit P un ensemble de mots. Pour $LS(P) = (V_L, A_L)$ et $AC(P^{-1}) = (V_C, A_C)$, on a que

$$A_C = \{(v^{-1}, u^{-1}) \mid (u, v) \in A_L\}.$$

D'après la proposition 6.19, on a alors que $AC(P^{-1})$ est alors en bijection avec un sous-arbre de $LS(P)$.



(a) Arbre d'Aho-Corasick de P^{-1}

(b) Arbre des liens suffixes de P

FIGURE 6.8 – Exemple du lien donné par la proposition 6.19 pour $P = \{baaab, aabba, babb\}$.

Remarque 6.20 L'arbre des suffixes généralisé et l'arbre des liens suffixes sont deux arbres orientés mais où l'orientation n'est pas la même. En effet, pour un ensemble de mots P, dans $GST(P)$, on a des arcs d'un nœud vers un nœud plus profond alors que dans $LS(P)$ on a des arcs d'un nœud vers un nœud moins profond.

6.3 Généralisation de l'arbre des suffixes généralisé

On introduit la notion de dépendance par suffixe entre mots, et la notion de chaîne de mots dépendants par suffixe pour pouvoir définir un index qui généralise l'arbre des suffixes [58] et les arbres des suffixes tronqués que l'on peut trouver dans Na et al. [64] et dans Schulz et al. [80]. Cette généralisation va être utilisée dans la section suivante (section 6.4) pour définir un nouvel arbre des suffixes tronqué qui va ensuite nous servir dans la section 9.3 pour construire le graphe de De Bruijn.

Définition 6.21

1. Un mot x est dit *dépendant par suffixe* d'un autre mot y si $|ov(x, y)| = |x| - 1$.
2. Soit w un mot et m un entier positif plus petit que $|w| - 1$. Un m-uplet de m mots $(x_0, \ldots, x_{m-1})$ est une *chaîne de mots dépendants par suffixe* de w si x_0 est un préfixe de w et pour tout $i \in \{2, \ldots, m\}$, x_i est un préfixe du suffixe de w commençant à la position i et tel que $|x_i| \geq |x_{i-1}| - 1$.

Exemple 6.22 Le uplet $(abb, bb, babba, abbaa)$ est une chaîne de mots dépendants par

suffixe de *abbabbaa*. En effet, on a

$$\begin{array}{cccccccc} a & b & b & a & b & b & a & a. \\ a & b & b & & & & & \\ & b & b & & & & & \\ & & b & a & b & b & a & \\ & & & a & b & b & a & a \end{array}$$

Soient $P := \{s_1, \dots, s_n\}$ un ensemble de mots et $\mathcal{R} = \{C_1, \dots, C_n\}$ un ensemble de uplets tels que pour tout $i \in \{1, \dots, n\}$, C_i est une chaîne de mots dépendants par suffixe de s_i. Soit $\widehat{\mathcal{R}} := \{\widehat{C_1}, \dots, \widehat{C_n}\}$ l'ensemble des uplets tels que pour $i \in \{1, \dots, n\}$ et $j \in \{1, \dots, |C_i|\}$, $\widehat{C_1}(j) := |C_1(j)|$, *i.e.* $\widehat{\mathcal{R}}$ contient des uplets de longueurs.

Avec $\widehat{\mathcal{R}}$ et P, on peut facilement reconstruire $\mathcal{R}$. Par la suite, on utilise $\mathcal{R}$ pour démontrer les résultats, et $\widehat{\mathcal{R}}$ pour établir les complexités des algorithmes. En effet, dans le cas où C_i est le uplet de tous les suffixes de s_i, la taille de C_i est linéaire en $|s_i|^2$ alors que $\widehat{C_i}$ est linéaire en $|s_i|$.

Soit w un mot ; w peut être dans différents uplets de $\mathcal{R}$. On définit alors $N(w)$ l'ensemble des (i, j) tels que $w = C_i(j)$. En d'autres termes, $N(w)$ est l'ensemble des coordonnées des éléments de $\mathcal{R}$ qui sont égaux à w. On pose de plus

$$\overline{\mathcal{R}} := \bigcup_{i \in \{1,\dots,n\}} \cup_{w_j \in C_i} \{w_j\}$$

Exemple 6.23 Soient $P = \{abba, ababb\}$ et $\mathcal{R} = \{(ab, bba), (ab, bab, ab)\}$. On a alors $\widehat{\mathcal{R}} = \{(2, 3), (2, 3, 2)\}$ et que $\overline{\mathcal{R}} = \{ab, bab, bba\}$.

Comme on l'a fait avec l'arbre des suffixes, on va définir une version contractée de l'arbre d'Aho-Corasick mais au lieu de prendre l'ensemble des suffixes d'un mot, on va prendre en entrée $\overline{\mathcal{R}}$.

On pose $T(\mathcal{R})$, l'arbre d'Aho-Corasick de $\overline{\mathcal{R}}$ dans lequel on a contracté tous les arcs (u, v) où u n'est pas dans $\overline{\mathcal{R}}$ et n'a qu'un seul enfant (voir Figure 6.9).

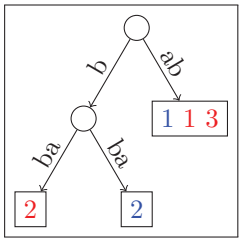


FIGURE 6.9 – Exemple de $T(\mathcal{R})$ pour $P = \{abba, ababb\}$ et $\mathcal{R} = \{(ab, bba), (ab, bab, ab)\}$. Dans chaque nœud v de $T(\mathcal{R})$, on ajoute l'information de $N(v)$: on associe à chaque mot de P une couleur unique et le numéro correspond à la position de début du suffixe dans le mot de la même couleur. Pour les étiquettes sur les arcs, la lecture se fait toujours dans le sens des arcs.

Remarque 6.24 Pour un mot w, $N(w)$ est l'équivalent de $Position - Suff_P(w)$ dans l'arbre $T(\mathcal{R})$.

On note par $\mathcal{N}$ et $\mathcal{L}$ respectivement l'ensemble des nœuds et des feuilles de $T(\mathcal{R})$. De plus, on définit pour chaque nœud de v de $T(\mathcal{R})$ deux poids :

— $s(v)$ est le nombre de fois qu'un élément dans un uplet de $\mathcal{R}$ est égal à v, *i.e.* $s(v) := |N(v)|$.

— $t(v)$ est le nombre de fois que le premier élément d'un uplet de $\mathcal{R}$ est égal à v, *i.e.* $t(v) := |\{(i,1) \in N(v) \mid i \in \{1,\dots,n\}\}|$.

Soit w un mot. On pose $Succ(w) := \{(i,j) \mid (i,j-1) \in N(w) \text{ et } j \leq |C_i|\}$. On définit $\mathcal{H}$ comme le sous-ensemble de $\mathcal{L}$ tel que :

$$\mathcal{H} := \{u \in \mathcal{L} \mid \exists C \in \mathcal{R} \text{ et } j < |C| \text{ tel que } u = C(j)\}$$

Il est équivalent de dire que $\mathcal{H} = \{u \in \mathcal{L} \mid Succ(u) \neq \emptyset\}$. Une fonction m de $\mathcal{H}$ vers $\mathcal{N}$ est appelée *lien possible* si pour tout nœud v de $\mathcal{H}$, $\exists (i,j) \in Succ(v)$ tel que $m(v) = C_i(j)$.

On va présenter un algorithme qui construit $T(\mathcal{R})$ et calcule pour chaque nœud v de $\mathcal{N}$, les poids $s(v)$ et $t(v)$ et un lien possible P_0.

Construction de $T(\mathcal{R})$. Maintenant, on va donner un algorithme pour construire $T(\mathcal{R})$. On va utiliser une version de l'algorithme de McCreight [58] donnée par Na et al. [64] sur notre entrée et on va construire pour chaque feuille v, $s(v)$, $t(v)$ et $P_0(v)$. Pour construire $T(\mathcal{R})$, on commence avec un arbre qui ne contient que la racine. Pour chaque mot w de chaque chaîne C de $\mathcal{R}$, on crée alors ou on actualise (s'il existe déjà) le nœud w comme suit. Supposons qu'on garde en mémoire le nœud v qui a été traité juste avant w.

Si w est le premier mot de C, on descend alors de la racine en comparant w avec les étiquettes de l'arbre. Si on crée un nœud w, alors on initialise $s(w)$ et $t(w)$ à 1 et $P_O(w)$ à nil. Si w existe déjà, on incrémente $s(w)$ et $t(w)$ de 1.

Si w n'est pas le premier mot de C, on commence à partir de v et comme dans l'algorithme de McCreight, on crée ou arrive sur un nœud représentant w. Si on doit créer ce nœud, $s(w)$ est initialisé à 1, $t(w)$ à 0 et $P_O(w)$ à nil. Sinon, on incrémente $s(w)$ de 1. Dans tous les cas on pose $P_0(v) = w$.

La boucle continue avec le mot suivant jusqu'à la fin et on obtient $T(\mathcal{R})$.

Théorème 6.25 Pour un ensemble de chaînes de mots dépendants par suffixe $\mathcal{R}$, on peut construire $T(\mathcal{R})$ en temps et en espace linéaire en $||P||$.

Preuve On va commencer par montrer que $T(\mathcal{R})$ est de taille linéaire en $||P||$. Son nombre de feuilles est borné par $\sum_{C\in\mathcal{R}} |C|$. Son nombre de nœuds est alors au plus $2 \times \sum_{C\in\mathcal{R}} |C| - 1 \leq ||P||$, et le nombre d'arcs est au plus $2 \times ||P||$. On a alors que la taille de $T(\mathcal{R})$ est linéaire en $||P||$.

On a clairement que la construction de $T(\mathcal{R})$ calcule les deux poids $s()$ et $t()$ et le lien possible $P_0()$ correctement. Pour la complexité, pour chaque chaîne de mots dépendants par suffixe C_i de $\mathcal{R}$, la longueur d'une traversée d'un arbre est égale à $|w_i|$, grâce à l'utilisation des liens suffixes. Comme pour l'algorithme McCreight, on a une complexité en $O(||P||)$. ■

On peut alors définir quelques instances d'ensemble de chaînes de mots dépendants par suffixe et retrouver des structures bien connues :

— Si $\mathcal{C} := \cup_{w \in P}\{\text{uplet de suffixes de } w\}$, on a alors que $T(\mathcal{C})$ est l'arbre des suffixes généralisé de P. On a alors la fonction qui à une feuille donne son lien suffixe est un exemple de lien possible.

— Si $B_k := \cup_{w \in P}\{\text{uplet de } k\text{-mers de } w \text{ et suffixes de longueur } k' < k \text{ de } w\}$, on a alors que $T(B_k)$ est l'arbre des suffixes généralisé et tronqué à k défini dans [80] (qui est une généralisation de l'arbre des suffixe tronqué à k de Na et al. [64]).

— Si $A_k := \cup_{w \in P}\{\text{uplet de } k+1\text{-mer de } w \text{ et suffixes de longueur } k \text{ de } w\}$, on a alors que $T(A_k)$ est l'arbre des suffixes tronqué que l'on va définir dans la sous-section 6.4.

Conclusion de la section On a défini dans cette section, une généralisation de l'arbre des suffixes qui nous permet de construire des structures tronquées de l'arbre des suffixes directement en temps linéaire en la norme de l'instance. L'intérêt de cette généralisation est dans l'utilisation des chaînes de mots dépendants par suffixe. En effet, on a montré dans cette section que cet ensemble de mots est une condition suffisante pour la construction d'un arbre de suffixes tronqué en temps linéaire en la norme de l'instance.

6.4 Notre arbre des suffixes tronqué

Dans cette section, on va s'intéresser plus particulièrement à une application de la section précédente où on va construire un arbre des suffixes tronqué qui nous servira dans la section 9.3 comme base pour ancrer le graphe de De Bruijn. On a la nécessité de définir un nouvel arbre des suffixes tronqué car aucun des arbres suffixes tronqués existants n'est linéaire en la taille du graphe de De Bruijn.

On va commencer par définir l'ensemble des chaînes de mots dépendants par suffixe qui va correspondre à notre arbre des suffixes tronqué.

Définition 6.26 Soient $P := \{s_1, \ldots, s_n\}$ un ensemble de mots et k un entier tel que pour tout i entre 1 et n, on a $k \leq s_i$.

1. Pour tout $i \in \{1, \ldots, |P|\}$ et $j \in \{1, \ldots, |s_i| - k + 1\}$, on note par $A_{k,i}$ le uplet tel que sa j^{eme} composante est définie par

$$A_{k,i}(j) := \begin{cases} s_i[j, j+k] & \text{si } j \leq |s_i| - k \\ s_i[j, |s_i|] & \text{si } j = |s_i| - k + 1 \end{cases}$$

2. et A_k est l'ensemble de ces uplets :

$$A_k := \bigcup_{i=1}^{n} \{A_{k,i}\}.$$

Exemple 6.27 Soient $P = \{abbbaa, bbbabb, aaabbb\}$ et $k = 3$. On a alors que

— $A_k = \{(abbb, bbba, bbaa, baa), (bbba, bbab, babb, abb), (aaab, aabb, abbb, bbb)\}$,

— $\overline{A_k} = \{abbb, bbba, bbaa, baa, bbab, babb, abb, aaab, aabb, bbb\}$.

Proposition 6.28 Soit $P := \{s_1, \dots, s_n\}$ un ensemble de mots.

1. Pour tout $i \in \{1, \dots, n\}$, $A_{k,i}$ est une chaîne de mots dépendants par suffixe de s_i.
2. De plus, $\overline{A_k} = Fact_{k+1}(P) \cup Suff_k(P)$.

Preuve

1. Pour tout $j \in \{1, \dots, |A_{k,i}| - k\}$, il est facile de voir que $A_{k,i}(j)$ est un mot dépendant par suffixe de $A_{k,i}(j+1)$.
2. Pour le second point

$$\begin{aligned}
\overline{A_k} &= \bigcup_{i=1}^{n}\Big(\bigcup_{j=1}^{|s_i|-k+1}\{A_{k,i}(j)\}\Big) \\
&= \bigcup_{i=1}^{n}\Big(\bigcup_{j=1}^{|s_i|-k}\{A_{k,i}(j)\}\bigcup\{A_{k,i}(|s_i|-k+1)\}\Big) \\
&= \bigcup_{i=1}^{n}(Fact_{k+1}(\{s_i\})\bigcup Suff_k(\{s_i\})) \\
&= Fact_{k+1}(P) \cup Suff_k(P).
\end{aligned}$$

Pour un ensemble de mots P et un entier k, on peut alors construire l'ensemble A_k et ainsi construire l'arbre $T(A_k)$ (voir Figure 6.10). En appliquant l'algorithme décrit dans

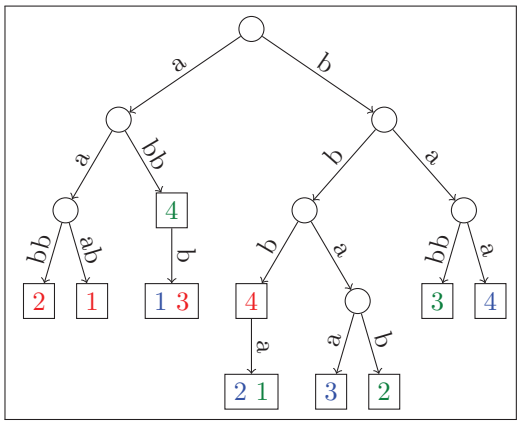


FIGURE 6.10 – Exemple de $T(A_k)$ pour $P = \{abbbaa, bbbabb, aaabbb\}$ et $k = 3$.

la sous-section 6.3 à A_k (Définition 6.26), et en utilisant le théorème 6.25, on obtient le résultat suivant :

Corollaire 6.29 On peut construire $T(A_k)$ en $O(||P||)$ en temps et en espace.

D'après la proposition 6.28, on a que $\overline{A_k} = Fact_{k+1}(P) \cup Suff_k(P)$. On peut alors borner la taille de l'arbre $T(A_k)$ et l'espace nécessaire pour sa construction.

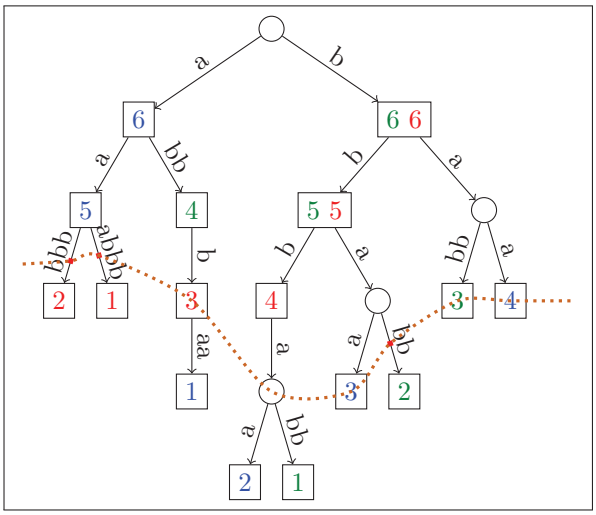


FIGURE 6.11 – Exemple d'arbre des suffixes pour $P = \{abbbaa, bbbabb, aaabbb\}$. Le trait vert passe par toutes les feuilles de $T(A_k)$ (voir Figure 6.10).

Proposition 6.30 La complexité en espace de $T(A_k)$ est en $O(min(||P||, |Fact_{k+1}(P)| + |Suff_k(P)|))$.

On dit que $T(A_k)$ est un arbre des suffixes tronqué, car il peut être vu comme l'arbre des suffixes où on aurait supprimé tous les nœuds plus grands que $k+1$ (voir Figure 6.11).

Remarque 6.31 Pour voir la réduction en pratique, on pourra se référer à la Table 1 dans (VII).

Conclusion de la section On a défini dans cette section, un nouvel arbre des suffixes tronqué dont la borne donnée par la proposition 6.30 nous permettra d'obtenir une bonne complexité pour construire le graphe de De Bruijn (voir Chapitre 9).

Conclusion du chapitre Dans ce chapitre, on a proposé une méthode pour généraliser la construction d'arbre des suffixes tronqué. On a ensuite appliqué cette méthode à l'ensemble des $k+1$-mers et l'ensemble des suffixes de taille k d'un ensemble de mots pour obtenir un nouvel arbre des suffixes tronqué qui va nous permettre une construction du graphe de De Bruijn en ne prenant pas plus en mémoire que la taille du graphe de De Bruijn lui-même (voir section 9.3).

Chapitre 7

Autres graphes de chevauchements

On a vu dans la section 3.4 la définition ainsi que quelques propriétés du graphe de chevauchements d'un ensemble de mots. Dans ce chapitre, on va s'intéresser à d'autres graphes qui vont modéliser aussi les chevauchements entre les mots. On va commencer par présenter un des graphes les plus utilisés en assemblage : le graphe de De Bruijn. On va ensuite présenter trois nouveaux graphes : l'arbre d'Aho-Corasick généralisé ainsi que son application le graphe hiérarchique des chevauchements et le graphe glouton. Nous avons déjà introduit le graphe hiérarchique des chevauchements et le graphe glouton dans (I), (IV), (V) et (XII). L'introduction de l'arbre d'Aho-Corasick généralisé dans ce chapitre, a pour but d'améliorer la compréhension du graphe hiérarchique des chevauchements en généralisant la définition de ce dernier ainsi que les propriétés qui en découlent. De plus, dans la sous-section 8.2.1, on utilisera l'arbre d'Aho-Corasick généralisé pour définir une version généralisée du graphe glouton.

Sommaire

7.1 Graphe de De Bruijn

Le premier graphe auquel on va s'intéresser dans ce chapitre est le graphe dit de De Bruijn. Plus exactement, on va parler ici des graphes de De Bruijn, car en effet dans la littérature on trouve souvent plusieurs définitions différentes avec le même nom. Parmi ces définitions, on commencera par présenter le graphe de De Bruijn original comme défini par De Bruijn dans [20]. On s'intéressera ensuite à deux applications de ce graphe pour l'assemblage de génome. Nous montrerons dans le chapitre 9 qu'en utilisant l'arbre des suffixes ou l'arbre des suffixes tronqué d'un ensemble de mots, on peut construire

de manière linéaire en la taille de cet ensemble les deux graphes de De Bruijn pour l'assemblage que l'on va présenter ci-dessous.

On commence par définitir le graphe de De Bruijn originel qui lui n'est pas associé à l'assemblage. Pour définir ce graphe, pour un entier k strictement positif, un mot de longueur k est appelé un *k-mer*.

Définition 7.1 ***Graphe de De Bruijn original (voir Figure 7.1)*** Soient Σ un alphabet fini et k un entier strictement plus grand que 1. Le *graphe de De Bruijn original* d'ordre k sur Σ, noté $dBG_k(\Sigma)$, est le graphe orienté $G := (V_k, A_k)$ tel que

$$V_k := \Sigma^k$$

$$A_k := \{(u, v) \in V \times V \mid |ov(u, v)| = k - 1\}$$

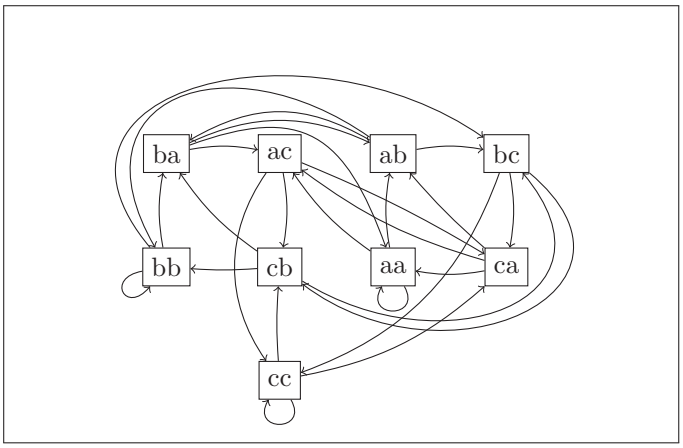


FIGURE 7.1 – Exemple de graphe de De Bruijn original $dBG_k(\Sigma)$ pour $\Sigma = \{a, b, c\}$.

Une des raisons de l'intérêt de ce graphe est que l'on peut retrouver tout mot de taille au moins k comme un parcours sur le graphe de De Bruijn original d'ordre k sur l'alphabet du mot. En effet, pour un mot w de taille au moins k sur un alphabet Σ, on peut décomposer w en k-mers :

$$w = \bigoplus_{i=1}^{|w|-k+1} w[i, i + k - 1].$$

Pour i entre 1 et $|w|-k+1$, chaque mot $w[i, i+k-1]$ appartient à Σ^k et est donc un nœud du graphe de De Bruijn original d'ordre k sur Σ. On associe le mot w, au chemin commençant en $w[1, k]$ puis prenant le nœud $w[2, k+1]$ et ainsi de suite jusqu'à $w[|w| - k+1, |w|]$ dans $dBG_k(\Sigma)$ (pour i entre 1 et $|w| - k$, les arcs $(w[i, i + k - 1], w[i + 1, i + k])$ sont bien des arcs de $dBG_k(\Sigma)$ car $|ov(w[i, i + k - 1], w[i + 1, i + k])| = |w[i + 1, i + k - 1]| = k - 1$).

Attention, ce chemin dans le graphe de De Bruijn original associé à un mot n'est pas forcément simple ; en effet si un même k-mer apparaît à deux endroits distincts de w, le chemin obtenu à partir de w passera deux fois par le nœud correspondant à ce k-mer.

Soit $P = \{s_1, \ldots, s_n\}$ un ensemble de mots sur un alphabet Σ. On peut alors prendre le sous-graphe du graphe de De Bruijn où on ne garde que les k-mers qui apparaissent dans les mots de P. On note ce graphe $dBG_k^-(P)$. Pour rappel, l'ensemble des k-mers de

P est noté $Fact_k(P)$, *i.e.* $Fact_k(P) = \Sigma^k \cap Fact(P)$. On définit formellement $dBG_k^-(P)$ comme suit.

Définition 7.2 $dBG_k^-(P)$ ***(voir Figure 7.2)*** Soient P un ensemble de mots sur Σ et k un entier strictement plus grand que 1. Le Graphe $dBG_k^-(P)$ de P, est le graphe orienté $G := (V_{P,k}, A_{P,k}^-)$ tel que

$$V_{P,k} := Fact_k(P)$$

$$A_{P,k}^- := \{(u,v) \in V_{P,k} \times V_{P,k} \mid |ov(u,v)| = k-1\}$$

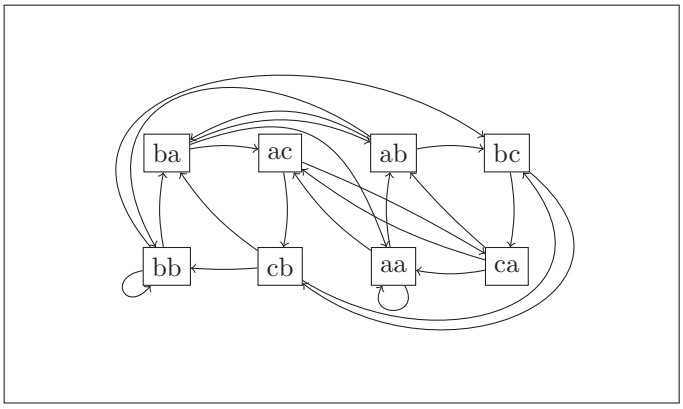


FIGURE 7.2 – Exemple de graphe de Bruijn $dBG_k^-(P)$ pour $P = \{bbabcaa, baacbac\}$.

Comme pour le graphe de De Bruijn original, on peut décomposer chaque mot de P et trouver un chemin correspondant à ce mot dans $dBG_k^-(P)$. Or pour certaines instances de P, certaines arêtes de $dBG_k^-(P)$ ne sont pas utilisées par les chemins obtenus à l'aide des mots de P : ces arêtes correspondent à des k-mers de P qui ont un chevauchement de taille $k-1$ mais où les k-mers ne sont pas successifs dans un même mot. On définit alors $dBG_k^+(P)$, le sous-graphe de $dBG_k^-(P)$ où on enlève ces arêtes.

Définition 7.3 $dBG_k^+(P)$ ***(voir Figure 7.3)*** Soient P un ensemble de mots sur Σ et k un entier strictement plus grand que 1. Le Graphe $dBG_k^+(P)$ de P, est le graphe orienté $G := (V_{P,k}, A_{P,k}^+)$ tel que

$$V_{P,k} := Fact_k(P)$$

$$A_{P,k}^+ := \{(u,v) \in V_{P,k} \times V_{P,k} \mid u \oplus v \in Fact_{k+1}(P)\}$$

On peut remarquer que $dBG_k^-(P)$ et $dBG_k^+(P)$ sont des sous-graphes du graphe des chevauchements de $Fact_k(P)$, où on ne garde que les arcs de poids $k-1$ pour $dBG_k^-(P)$ et où on ne garde que les arcs de poids $k-1$ tels que les deux k-mers sont successifs dans un mot de P pour $dBG_k^+(P)$.

On peut alors s'intéresser à la taille des trois graphes de Bruijn, c'est-à-dire à la taille de leur ensemble de nœuds et de leur ensemble d'arêtes. Ce sujet a été largement étudié (voir par exemple Crochemore et al. [17]), on redonne ici les résultats principaux car on

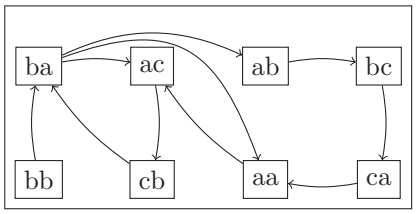


FIGURE 7.3 – Exemple de graphe de Bruijn $dBG_k^+(P)$ pour $P = \{bbabcaa, baacbac\}$.

s'intéresse plus précisément aux liens peu connus qu'il peut y avoir entre $dBG_k^-(P)$ et $dBG_k^+(P)$.

Pour $dBG_k(\Sigma)$, il est facile de voir que $|V_k| = c^k$ et que $|A_k| = c^{k+1}$ où $|\Sigma| = c$.

Pour $dBG_k^-(P)$ et $dBG_k^+(P)$, il est plus compliqué de donner une taille précise, mais on peut borner leur taille. De manière immédiate, on a que $|Fact_k(P)| \leq ||P||$. On a alors que $|V_{P,k}| \leq ||P||$. Comme $|A_{P,k}^+| = |Fact_{k+1}(P)|$, on a que $|A_{P,k}^+| \leq ||P||$ et donc que la taille de $dBG_k^+(P)$ est linéaire en la norme de P. Grâce à la proposition 7.7, on a que $|A_{P,k}^-| \leq c \times |A_{P,k-1}^+| \leq c \times ||P||$, on a donc que la taille de $dBG_k^-(P)$ est linéaire en $|\Sigma| \times ||P||$.

On va maintenant regarder un peu plus en détail comment varie le nombre de k-mers d'un ensemble de mots P sur un alphabet Σ de taille c. On va définir deux entiers, α_P et β_P qui seront les bornes minimales et maximales d'un intervalle où le nombre de k-mers de P sera intéressant à explorer.

Posons

$$N_{P,k} = \sum_{w_i \in P} \max(|w_i| - k + 1, 0)$$

qui correspond à une borne maximale du nombre de k-mers différents qu'il peut y avoir dans P. En effet, il a au plus $|w_i| - k + 1$ différents k-mers dans le mot w_i.

On a alors $|Fact_k(P)| \leq N_{P,k}$ et $|Fact_k(P)| \leq c^k$, où c^k est le nombre de k-mers différents que l'on peut faire avec c lettres différentes.

On a alors que $|Fact_k(P)|$ est égale à c^k pour des k petits (c'est toujours vrai pour $k = 1$), d'où la proposition suivante :

Proposition 7.4 Soit $k \geq 2$, on a alors

$$\big(|Fact_k(P)| = c^k\big) \Rightarrow \big(|Fact_{k-1}(P)| = c^{k-1} \text{ et } |Fact_1(P)| = c\big).$$

On va alors poser :

$$\alpha_P := \max(\{k \geq 1 \mid |Fact_k(P)| = c^k\}).$$

De même, $|Fact_k(P)|$ est égale à $N_{P,k}$ pour des k grands.

Proposition 7.5 Soit $k \geq 1$, on a alors

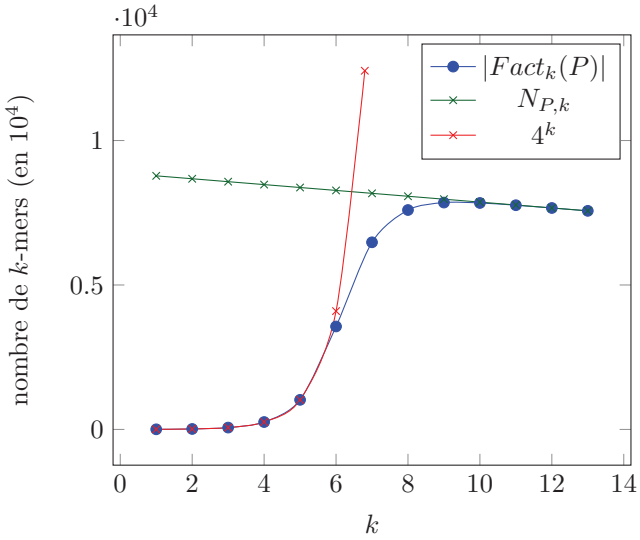
$$|Fact_k(P)| = N_{P,k} \Rightarrow |Fact_{k+1}(P)| = N_{P,k+1} \text{ et } |Fact_{max_{w\in P}|w|}(P)| = N_{P,max_{w\in P}|w|}$$

.

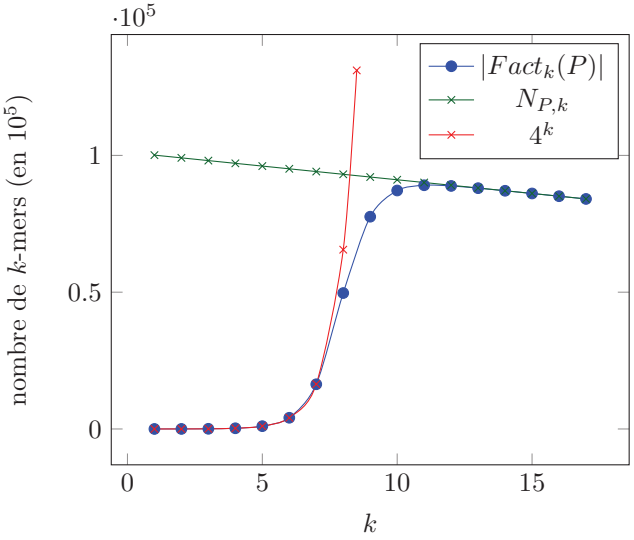
On pose de même :

$$\beta_P := \min(\{k \geq 1 \mid |Fact_k(P)| = N_{P,k}\}).$$

Sur la figure 7.4, on peut voir les résultats des propositions 7.4 et 7.5.



(a)



(b)

FIGURE 7.4 – La figure (a) montre le nombre de k-mers différents (en bleu) d'un tirage aléatoire de 100 mots d'une longueur entre 70 et 100 sur un alphabet de taille 4. La figure (b) montre la même chose pour un tirage aléatoire de 1000 mots d'une longueur 100 sur un alphabet de taille 4. De manière générale, en vert et en rouge, on a deux courbes qui maximisent la courbe bleue et coïncide avec de petites valeurs de k pour la rouge et avec de grandes valeurs de k pour la verte.

On a alors que pour tout $k \leq \alpha_P$, $|Fact_k(P)| = c^k$ et pour $k > \alpha_P$, $|Fact_k(P)| < c^k$. De même pour $k \geq \beta_P$, $|Fact_k(P)| = N_{P,k}$ et pour $k < \beta_P$, $|Fact_k(P)| < N_{P,k}$. Comme

$A^+_{P,k} \subseteq A^-_{P,k}$. On a $|A^+_{P,k}| \leq |A^-_{P,k}|$. Par la définition de $dBG^-_k(P)$, si $|Fact_k(P)| = |V_k|$ alors $|A^-_{P,k}| = |A_k|$.

Proposition 7.6 Pour $k \leq \alpha_P$, $dBG^-_k(P)$ est isomorphe à dBG_k.

On obtient donc que pour $k \leq \alpha_P$, le graphe de De Bruijn $dBG^-_k(P)$, n'apporte aucune information utile.

Pour $k \geq \beta_P$, on a la même chose avec le graphe de De Bruijn $dBG^+_k(P)$. En effet si $|Fact_k(P)| = N_{S,k}$, alors tous les k-mers de P sont différents, du coup $dBG^+_k(P)$ est un ensemble de m graphes chaînes distincts où m est le nombre de mots de P supérieur ou égal à k.

Voyons maintenant les liens entre $dBG^+_k(P)$ et $dBG^-_k(P)$, *i.e.* entre $|A^+_{P,k}|$ et $|A^-_{P,k}|$.

Proposition 7.7 Pour $k \geq 1$,

$$\frac{1}{c} \times |A^-_{P,k+1}| \leq |A^+_{P,k}| \leq |A^-_{P,k}|$$

Remarque 7.8 En utilisant la borne donnée par la proposition 7.7, on peut de la même façon borner la taille de $A^-_{P,k}$ par la taille de $A^+_{P,k}$ et de $A^+_{P,k+1}$:

$$|A^+_{P,k+1}| \leq |A^-_{P,k}| \leq c \times |A^+_{P,k}|$$

On a donc un lien entre $dBG^+_k(P)$ et $dBG^-_k(P)$, que l'on va encore plus développer dans la partie 9.4 où on verra une façon de construire ces deux graphes de De Bruijn en temps linéaire en la norme de P.

Enfin on va définir les versions contractées des graphes de De Bruijn et plus exactement celle de dBG^+_k.

Définition 7.9 $CdBG^+_k(P)$ Soient P un ensemble de mots sur Σ et k un entier strictement plus grand que 1. Le Graphe $CdBG^+_k(P)$ de P, est le graphe orienté $dBG^+_k(P)$ où on a contracté récursivement tous les arcs (u, v) tels que $d^{out}(u) = d^{in}(v)$ dans $dBG^+_k(P)$.

On a une définition similaire de la version contractée de $dBG^-_k(P)$, que l'on note $CdBG^-_k(P)$.

Conclusion de la section On a défini dans cette section le lien entre les différentes versions des graphes de De Bruijn, ce qui nous a permis de borner la taille de l'un grâce à la taille de l'autre. Ce lien entre les tailles, qui est assez trivial, n'est donné ici que comme préambule à la section 9.4 où on va montrer que les graphes $dBG^+_k(P)$ et $dBG^-_k(P)$ viennent d'un même graphe.

7.2 Généralisation de l'arbre d'Aho-Corasick

On a vu dans la section 3.4 que le graphe des chevauchements de P est de taille quadratique en le cardinal de P. Malheureusement, le graphe des chevauchements

n'exploite pas le fait que plusieurs mots peuvent partager les mêmes chevauchements ou des chevauchements inclus les uns dans les autres. Dans cette section, on va présenter un ensemble de graphes qui vont résumer l'information incluse dans le graphe de chevauchements. Pour ce faire, on va commencer par présenter une généralisation de l'arbre d'Aho-Corasick ainsi que certaines de ces propriétés. Dans un second temps, on va présenter deux graphes qui seront des cas particuliers de cette généralisation et qui nous permettrons de stocker l'information du graphe de chevauchements dans un graphe plus petit en taille.

7.2.1 L'arbre d'Aho-Corasick généralisé

Dans la section 6.1, on a vu que l'arbre d'Aho-Corasick d'un ensemble de mots P fusionné avec l'arbre des liens d'échec forme un graphe composé de deux arbres définis sur un même ensemble de nœuds qui est l'ensemble des préfixes des mots de P. En reprenant la définition que l'on a donné dans la définition 6.1 (page 129), on a un arc de u vers v dans l'arbre d'Aho-Corasick si et seulement si u est le plus grand préfixe (parmi tous les préfixes de P) de v. De même, en reprenant la définition 6.4 (page 128), un arc de u vers v est dans l'arbre des liens d'échec si et seulement si v est le plus grand suffixe (parmi tous les préfixes de P) de u.

Au lieu d'utiliser l'ensemble des préfixes d'un ensemble de mots, on peut prendre un ensemble de mots quelconque et définir une généralisation de la fusion entre l'arbre d'Aho-Corasick et l'arbre des liens d'échec.

Définition 7.10 L'*arbre d'Aho-Corasick généralisé* d'un ensemble de mots S, noté $GAC(S)$, est le graphe orienté $(S, R \cup B)$ avec deux ensembles d'arcs R et B où :

$$R := \{(x, y) \in S \times S \mid y \text{ est le plus long suffixe de } x \text{ dans } S\},$$

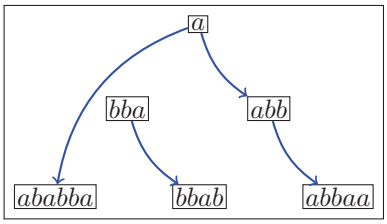
$$B := \{(y, x) \in S \times S \mid y \text{ est le plus long préfixe de } x \text{ dans } S\}.$$

De manière générale, on note $GAC_R(S) := R$ et $GAC_B(S) := B$.

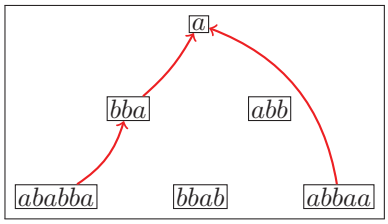
Soient u un mot et deux préfixes x et y de u. On a alors que x est un préfixe de y ou que y est un préfixe de x. De même, pour deux suffixes d'un même mot, le plus petit des suffixes est suffixe du plus grand des suffixes. On a alors que les graphes $(S, GAC_R(S))$ et $(S, GAC_B(S))$ sont des sous-graphes d'arbres, *i.e.* des forêts (voir Figure 7.5). Soient deux nœuds u et v de $GAC(S)$. Si l'arc (u, v) est dans $GAC_B(S)$, on a alors que u est un préfixe de v, on dira alors que u est le parent de v dans $(S, GAC_B(S))$. De même, si l'arc (u, v) est dans $GAC_R(S)$, v est un suffixe de u et on dira que v est le parent de u dans $(S, GAC_R(S))$.

Comme le nombre d'arcs d'un arbre est linéaire en son nombre de nœuds, on a que la taille d'un arbre est linéaire en son nombre de nœuds. Or on a que le graphe $GAC(S)$ est composé de deux sous-graphes d'arbre : $(S, GAC_R(S))$ et $(S, GAC_B(S))$, on a alors la proposition suivante.

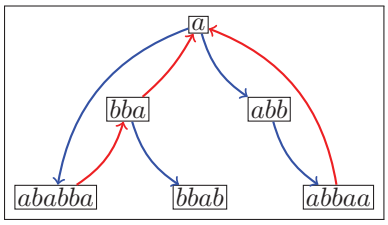
Proposition 7.11 Soit S un ensemble de mots. La taille de $GAC(S)$ est linéaire en $|S|$.



(a) La forêt $(S, GAC_B(S))$



(b) La forêt $(S, GAC_R(S))$



(c) Le graphe $GAC(S)$

FIGURE 7.5 – Exemple d'arbre d'Aho-Corasick généralisé pour l'ensemble de mots $S = \{a, bba, abb, ababba, bbab, abbaa\}$. La figure (a) est la forêt $(S, GAC_B(S))$ où les arcs sont en bleus et la figure (b) est la forêt $(S, GAC_R(S))$ où les arcs sont en rouges. La figure (c) est le graphe $GAC(S)$ où les arcs en bleus sont les éléments de $GAC_B(S)$ et les arcs en rouges sont les éléments de $GAC_R(S)$.

Preuve La taille de $GAC(S)$ est le nombre de nœuds de $GAC(S)$ plus le nombre d'arcs de $GAC(S)$. Comme l'ensemble des nœuds est S, le nombre de nœuds est $|S|$. De plus comme $(S, GAC_R(S))$ ainsi que $(S, GAC_B(S))$ sont des sous-graphes d'arbre, le nombre d'arcs est inférieur à $2 \times |S|$. ■

À l'aide de l'arbre d'Aho-Corasick généralisé, pour un ensemble de mots P, on peut alors retrouver certains arbres connus :

- Soit $Prefixe(P)$ l'ensemble des préfixes de mots de P. On retrouve alors l'arbre d'Aho-Corasick classique :
  - $AC(P) = (Prefixe(P), GAC_B(Prefixe(P)))$,
  - $LE(P) = (Prefixe(P), GAC_R(Prefixe(P)))$.
- Pour $V_{GST(P)}$ l'ensemble des nœuds de l'arbre des suffixes généralisé de P, on a alors que
  - $GST(P) = (V_{GST(P)}, GAC_B(V_{GST(P)}))$,
  - $LS(P) = (V_{GST(P)}, GAC_R(V_{GST(P)}))$.

Remarque 7.12 L'arbre d'Aho-Corasick généralisé ne prend pas forcément un ensemble de mots factor-free. Dans le cas où on prend un ensemble de mots P qui est factor-free, on a alors que les deux ensembles $GAC_R(P)$ et $GAC_B(P)$ sont vides.

Conclusion de la sous-section On a mis en évidence dans cette sous-section un graphe qui généralise l'arbre d'Aho-Corasick ainsi que l'arbre des suffixes généralisé. Dans la sous-section suivante, on va donner quelques propriétés de ce nouveau graphe.

7.2.2 Propriétés sur l'arbre d'Aho-Corasick généralisé

Dans cette sous-section en utilisant les propriétés de l'arbre d'Aho-Corasick généralisé, on va définir un type de parcours de ce graphe pour passer d'une feuille à l'autre.

On va commencer par définir le lien entre deux mots suffixes ou préfixes l'un de l'autre dans l'arbre d'Aho-Corasick généralisé.

Proposition 7.13 Soient S un ensemble de mots.

1. Pour u et v deux mots de S tels que u est un préfixe de v. Il existe alors un unique chemin de u vers v dans le graphe $(S, GAC_B(S))$.
2. Pour u et v deux mots de S tels que u est un suffixe de v. Il existe alors un unique chemin de v vers u dans le graphe $(S, GAC_R(S))$.

Pour rappel, l'ensemble $Ov(P)$ est l'ensemble de tous les chevauchements maximaux entre deux mots de P, *i.e.*

$$Ov(P) = \{ov(u, v) \mid (u, v) \in P \times P\}$$

Remarque 7.14 L'auto-chevauchement d'un mot u de P, c'est-à-dire le chevauchement maximal de u sur lui-même est un élément de $Ov(P)$.

Soit P un ensemble de mots. Dans la suite, on va regarder quelques propriétés de l'arbre d'Aho-Corasick généralisé sur un ensemble de mots S tel que $Ov(P) \cup P \subseteq S$.

Exemple 7.15 Soient $P = \{ababba, bbab, abbaa\}$ et $S = \{\varepsilon, b, a, ab, bba, abb, abba, ababba, bbab, abbaa\}$. On a alors que $Ov(P) = \{\varepsilon, b, a, ab, bba, abba\}$ et donc $Ov(P) \cup P \subseteq S$. On va utiliser cet exemple pour illustrer notre propos dans la suite de la section.

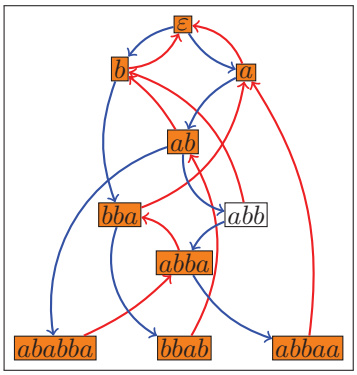


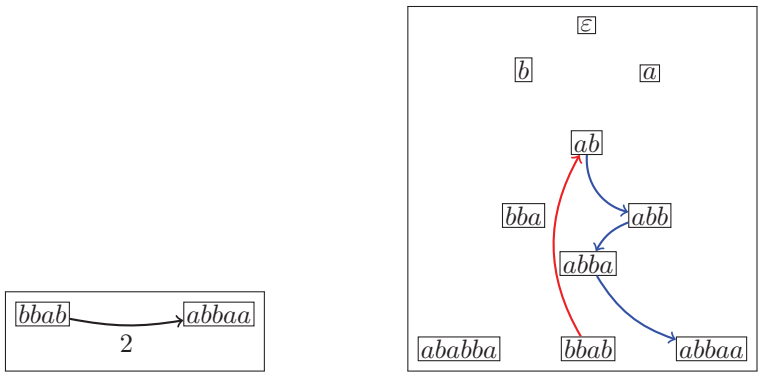
FIGURE 7.6 – Exemple d'arbre d'Aho-Corasick généralisé pour $S = \{\varepsilon, b, a, ab, bba, abb, abba, ababba, bbab, abbaa\}$ défini dans l'exemple 7.15. L'ensemble de mots $P = \{ababba, bbab, abbaa\}$ est tel que $Ov(P) \cup P \subseteq S$ (les nœuds de $Ov(P) \cup P$ sont en oranges.

Dans la section 3.4, on a vu que chercher un chemin hamiltonien dans le graphe de chevauchements de P revient à résoudre le problème SLS_c et donc SLS. En effet, tous les nœuds sont les mots de P et chaque arc du graphe de chevauchements correspond à

un chevauchement. On veut alors trouver un chemin qui passe une et une seule fois par chaque nœud du graphe de chevauchements.

Pour s_1 et s_2 deux mots de P, on a alors que le chevauchement maximal de u sur v correspond à un arc dans le graphe des chevauchements. On va regarder le pendant de cet arc dans l'arbre d'Aho-Corasick généralisé de S.

Comme $P \subseteq S$, s_1 et s_2 sont deux nœuds de $GAC(S)$. Comme $Ov(P) \subseteq S$, $ov(s_1, s_2)$ est aussi un nœud de $GAC(S)$. D'après la proposition 7.13, il existe un chemin p_1 de s_1 vers $ov(s_1, s_2)$ dans $(S, GAC_R(S))$ et il existe un chemin p_2 de $ov(s_1, s_2)$ vers s_2 dans $(S, GAC_B(S))$, car $ov(s_1, s_2)$ est un suffixe de s_1 et un préfixe de s_2. Le chemin de s_1 vers s_2 qui correspond au chemin où on prend p_1 puis de p_2 est appelé le *Red-Blue path* de s_1 vers s_2 et noté $RB - path_S(s_1, s_2)$ (voir Figure 7.7).



(a) (b)

FIGURE 7.7 – Lien entre un arc du graphe de chevauchements et un Red-Blue path pour S et P définis dans l'exemple 7.15. La figure (a) représente l'arc de *bbab* vers *abbaa* dans le graphe des chevauchements de P. La figure (b) représente le Red-Blue path de *bbab* vers *abbaa* dans $GAC(S)$ qui est la fusion du chemin rouge de $(S, GAC_R(S))$ qui va de *bbab* vers $ov(bbab, abbaa) = ab$ et du chemin bleu de $(S, GAC_B(S))$ qui va de $ov(bbab, abbaa) = ab$ vers *abbaa*.

On peut alors borner la taille du chemin entre u et v dans l'arbre d'Aho-Corasick généralisé à l'aide des tailles de u et de v.

Proposition 7.16 Soient u et v deux mots de P. On a alors $|RB - path_S(u, v)| \leq |pr(u, v)| + |su(u, v)|$.

Preuve Soient p_1 le chemin de s_1 vers $ov(s_1, s_2)$ dans $(S, GAC_R(S))$ et p_2 le chemin de $ov(s_1, s_2)$ vers s_2 dans $(S, GAC_B(S))$, on a alors $RB - path_S(u, v)$ est le chemin p_1 puis le chemin p_2. Or $|p_1| \leq |pr(u, v)|$ car p_1 passe par les suffixes de s_1 qui sont plus grands que $|ov(s_1, s_2)|$. De même, on a que $|p_2| \leq |su(u, v)|$ et donc que $|RB - path_S(u, v)| \leq |pr(u, v)| + |su(u, v)|$. ■

On peut de plus, trouver un algorithme qui permet de calculer le chemin $RB - path_S(u, v)$ dans $GAC(S)$.

Proposition 7.17 L'algorithme 7.1 appliqué à deux mots u et v de P a une complexité en $O(|pr(u,v)| + |su(u,v)|)$.

Algorithme 7.1 : Algorithme pour trouver $RB - path_S(u,v)$ pour deux mots u et v de S.

Entrée : Soient u et v deux mots de P
Sortie : Le chemin $RB - path_S(u,v)$

```
1  p1 ← ∅;
2  p2 ← ∅;
3  u' ← u;
4  v' ← v;
5  while |u'| ≠ |v'| do
6      if |u'| > |v'| then
7          w ← le parent de u dans (S, GAC_R(S)) ;
8          p1 ← p1.((u', w));
9          u' ← w;
10     if |u| < |v| then
11         w ← le parent de v dans (S, GAC_B(S)) ;
12         p2 ← ((w, v')).p2;
13         v' ← w;
14 return p1.p2.
```

Comme on a que $|ov(u,v)| = min(\{|w| \mid (w',w) \in RB - path_S(u,v)\})$ et grâce aux propositions 7.16 et 7.17, on a la proposition suivante :

Proposition 7.18 Soit u et v deux mots de P, à l'aide de $GAC(S)$, on peut trouver $ov(u,v)$ en $O(|pr(u,v)| + |su(u,v)|)$ et donc trouver $|ov(u,v)|$ avec la même complexité.

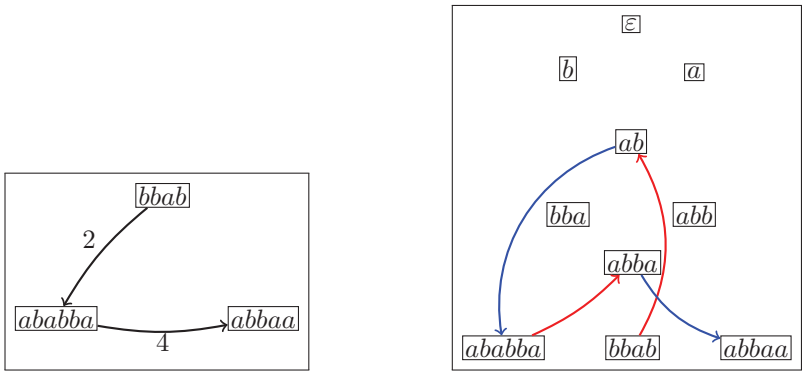
On peut alors associer à chaque chemin hamiltonien du graphe des chevauchements de P, un chemin sur l'arbre d'Aho-Corasick généralisé de S. Soit $p := ((u_1,u_2),\dots,(u_{n-1},u_n))$ un chemin hamiltonien du graphe des chevauchements, le *Red-Blue chemin* associé à p, noté $RB - Linear_S(p)$ est un uplet de Red-Blue path $(RB - path_S(u_1,u_2)\dots RB - path_S(u_{n-1},u_n))$ (voir Figure 7.8). Pour c un circuit hamiltonien, ou pour C une couverture cyclique, on définit de même le *Red-Blue circuit* associé à c, noté $RB - Circular_S(c)$ (voir Figure 7.9), et la *Red-Blue couverture cyclique* associée à C, notée $RB - CC_S(C)$ (voir Figure 7.10).

En utilisant la proposition 7.16 et en sommant sur l'ensemble des arcs, on peut borner la taille du Red-Blue chemin (respectivement du Red-Blue circuit et Red-Blue couverture cyclique) par la superchaîne linéaire (respectivement la superchaîne circulaire et la couverture circulaire des mots) construit à partir d'un chemin (respectivement un circuit et une couverture cyclique) sur le graphe de chevauchements (les notations utilisées dans le corollaire suivant sont définies dans la section 2.5).

Corollaire 7.19 Soient P un ensemble de mots, S un ensemble de mots tel que $Ov(P) \cup P \subseteq S$ et G le graphe des chevauchements de P.

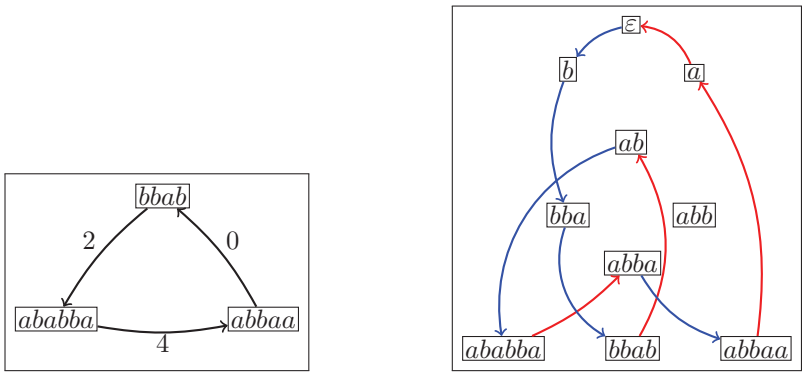
— Soit $p := ((u_1,u_2),\dots,(u_{n-1},u_n))$ un chemin hamiltonien sur G. On a

$$|RB - Linear_S(p)| \leq 2 \times |sol_{HP \to LS}(p).u_n|$$



(a) (b)

FIGURE 7.8 – Lien entre un chemin du graphe de chevauchements et un Red-Blue chemin pour S et P définis dans l'exemple 7.15. La figure (a) représente le chemin $((bbab, ababba), (ababba, abbaa))$ dans le graphe de chevauchements. La figure (b) représente le Red-Blue chemin $((bbab, ab), (ab, ababba), (ababba, abba), (abba, abbaa))$ dans $GAC(S)$.



(a) (b)

FIGURE 7.9 – Lien entre un circuit du graphe de chevauchements et un Red-Blue circuit pour S et P définis dans l'exemple 7.15. La figure (a) représente le circuit $((bbab, ababba), (ababba, abbaa), (abbaa, bbab))$ dans le graphe de chevauchements. La figure (b) représente le Red-Blue circuit $((bbab, ab), (ab, ababba), (ababba, abba), (abba, abbaa), (abbaa, a), (a, \varepsilon), (\varepsilon, b), (b, bba), (bba, bbab))$ dans $GAC(S)$.

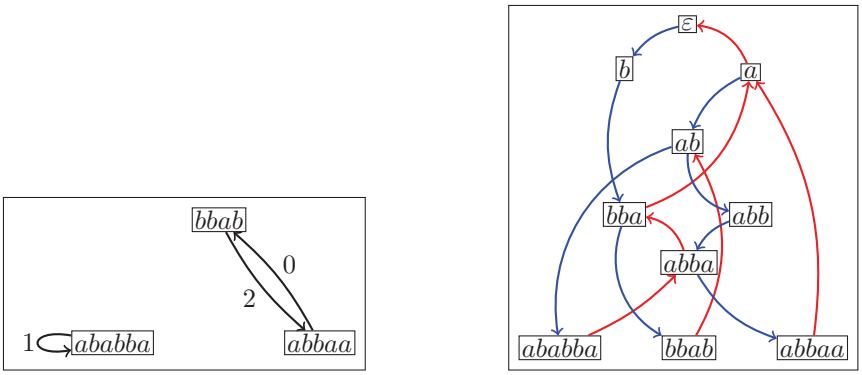
— Soit c un circuit hamiltonien sur G. On a

$$|RB - Circular_S(c)| \leq 2 \times |sol_{HC \to CS}(c)|$$

— Soit C une couverture cyclique sur G. On a

$$|RB - CC_S(C)| \leq 2 \times |sol_{HCC \to CCS}(C)|$$

On a donc que $RB-Linear_S(p)$, $RB-Circular_S(c)$ et $RB-CC_S(C)$ sont de longueurs linéaires en $||P||$. Ces parcours de l'arbre d'Aho-Corasick généralisé sont alors en temps



(a) (b)

FIGURE 7.10 – Lien entre une couverture cyclique du graphe de chevauchements et une Red-Blue couverture cyclique pour S et P définis dans l'exemple 7.15. La figure (a) représente la couverture cyclique $\{((bbab, abbaa), (abbaa, bbab)), ((ababba, ababba))\}$ dans le graphe de chevauchements. La figure (b) représente la Red-Blue couverture cyclique $\{((bbab, ab), (ab, abb), (abb, abba), (abba, abbaa), (abbaa, a), (a, \varepsilon), (\varepsilon, b), (b, bba), (bba, bbab))$ $, ((ababba, abba), (abba, bba), (bba, a), (a, ab), (ab, ababba))\}$ dans $GAC(S)$.

linéaire en $||P||$, c'est-à-dire en la taille de l'instance.

Conclusion de la sous-section Dans cette sous-section, on a montré que pour un ensemble de mots P et pour un ensemble de mots S tels que $Ov(P) \cup P \subseteq S$, à partir du graphe $GAC(S)$, on peut retrouver les informations contenues sur le graphe des chevauchements de P.

7.2.3 Graphes hiérarchiques des chevauchements

Dans la sous-section 7.2.1, grâce à la proposition 7.11 on a montré que la taille de l'arbre d'Aho-Corasick généralisé d'un ensemble de mots S est linéaire en le cardinal de S. De plus, dans la sous-section 7.2.2, la proposition 7.18 nous dit que si pour un ensemble de mots P, $Ov(P) \cup P \subseteq S$, on a alors que toute l'information contenue dans le graphe de chevauchements de P, c'est-à-dire les tailles des chevauchements entre tous u et v de P, est contenue dans l'arbre d'Aho-Corasick généralisé de S et est accessible en $O(|pr(u, v)| + |su(u, v)|)$.

On va dans cette sous-section présenter deux graphes issus d'arbres d'Aho-Corasick généralisés qui nous paraissent intéressants : le graphe hiérarchique des chevauchements (HOG) ainsi que le graphe hiérarchique des chevauchements étendu (EHOG). On va définir ces deux structures car la taille de la première est bornée linéairement par la taille du graphe de chevauchements mais on a un algorithme en temps linéaire pour construire le deuxième.

On va alors définir naturellement le graphe hiérarchique des chevauchements qui peut être vu comme une version contractée du "Hierarchical Graph" dont on retrouve la définition dans Golovnev et al. [32].

Définition 7.20 ***Graphe hiérarchique des chevauchements (voir Figure 7.11)*** Soit P un ensemble de mots. Le *graphe hiérarchique des chevauchements* de P, noté $HOG(P)$ est l'arbre d'Aho-Corasick généralisé pour l'ensemble $Ov(P) \cup P$.

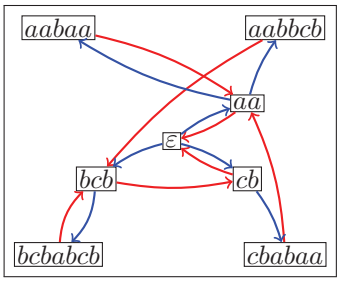


FIGURE 7.11 – Exemple de graphe hiérarchique des chevauchements pour $P = \{aabaa, aabbcb, cbabaa, bcbabcb\}$.

Théorème 7.21 Soit P un ensemble de mots. La taille de $HOG(P)$ est linéaire en $min(|P|^2, ||P||)$.

Preuve D'après la proposition 7.11, comme $|Ov(P) \cup P| \leq 2|P|^2$, la taille de $HOG(P)$ est linéaire en $|P|^2$. De plus, comme l'ensemble $Ov(P) \cup P$ est inclus dans l'ensemble de préfixes de P, il est aussi inclus dans l'ensemble des nœuds de $AC(P)$, on a que $|Ov(P) \cup P| \leq ||P||$ et donc d'après la proposition 7.11 que la taille de $HOG(P)$ est linéaire en $||P||$. ■

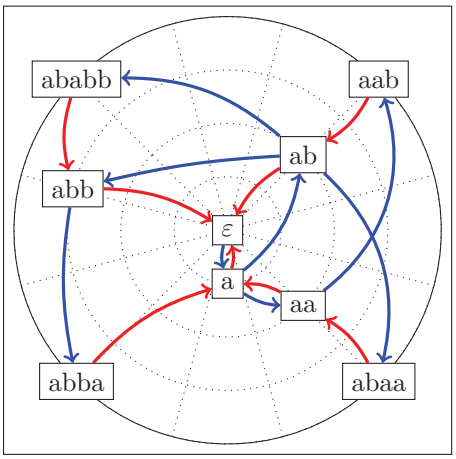
Remarque 7.22 Dans la preuve du théorème 7.21, pour borner la taille de $HOG(P)$ (voir Figure 7.12a) on utilise le fait que $Ov(P) \cup P$ est inclus dans l'ensemble des nœuds de $AC(P)$ (voir Figure 7.12b), mais on a aussi que $Ov(P) \cup P$ est inclus dans l'ensemble des nœuds de $GST(P)$ (voir Figure 7.12c).

Pour un ensemble de mots P, on n'a pas d'algorithme en $O(||P||)$ pour construire directement $HOG(P)$. On va donc définir un graphe un peu plus grand mais qui va avoir un algorithme de construction en temps linéaire en $||P||$: le graphe hiérarchique des chevauchements étendu. Pour cela, on définit $Ov^*(P)$ l'ensemble des chevauchements pas forcément maximaux des mots de P, *i.e.*

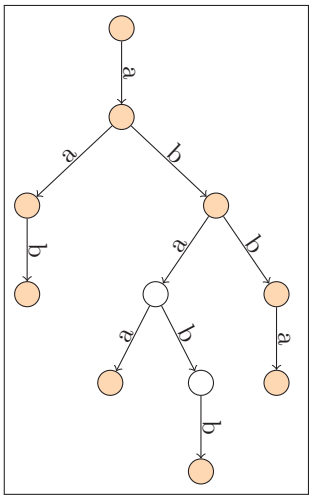
$$Ov^*(P) = \bigcup_{(x,y) \in P \times P} Prefixe(x) \cap Suffixe(y)$$

Remarque 7.23 Soient u et v deux mots de P. Comme $ov(u, v)$ est un préfixe de v et un suffixe de u, $ov(u, v) \in Prefixe(v) \cap Suffixe(u)$ et donc $Ov(P) \subseteq Ov^*(P)$.

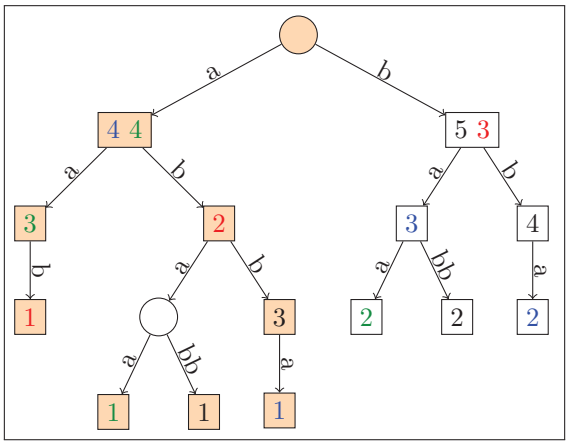
Définition 7.24 ***Graphe hiérarchique des chevauchements étendu (voir Figure 7.13)*** Soit P un ensemble de mots. Le *graphe hiérarchique des chevauchements étendu* de P, noté $EHOG(P)$ est l'arbre d'Aho-Corasick généralisé pour l'ensemble $Ov^*(P) \cup P$.



(a) Graphe hiérarchique des chevauchements de P



(b) Arbre d'Aho-Corasick de P



(c) Arbre des lien suffixes de P

FIGURE 7.12 – Exemple du le lien entre les nœuds de $HOG(P)$ et les nœuds de $AC(P)$ et de $GST(P)$ donné par la remarque 7.22 (pour $P = \{ababb, aab, abaa, abba\}$). Dans les figures (b) et (c), les nœuds en orange sont les nœuds de $HOG(P)$.

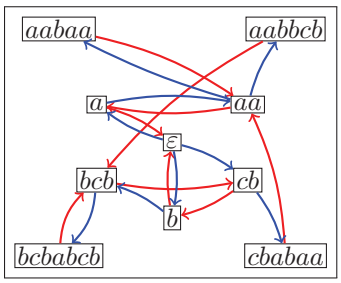


FIGURE 7.13 – Exemple de graphe hiérarchique des chevauchements étendu pour $P = \{aabaa, aabbcb, cbabaa, bcbabcb\}$.

On a alors le théorème suivant :

Théorème 7.25 Soit P un ensemble de mots. Le graphe $EHOG(P)$ peut être construit en temps et en espace linéaire en $||P||$.

Preuve En parcourant $LE(P)$ à partir des feuilles de $AC(P)$, on passe exactement par l'ensemble des nœuds de $Ov^*(P) \cup P$, c'est-à-dire les nœuds de $EHOG(P)$. Pour construire les arcs de $EHOG(P)$, c'est-à-dire $GAC_B(Ov^*(P) \cup P)$ et $GAC_R(Ov^*(P) \cup P)$, on parcourt $AC(P)$ et $LE(P)$. Comme le nombre de nœuds de $LE(P)$ est linéaire en $||P||$, on a un algorithme en temps linéaire pour construire $EHOG(P)$. ■

Conclusion du chapitre Dans cette sous-section, on a mis en avant deux nouveaux graphes : le graphe hiérarchique des chevauchements et le graphe hiérarchique des chevauchements étendu d'un ensemble de mots P. Le graphe hiérarchique des chevauchements étendu peut être construit et est de taille linéaire en la norme de l'instance. Le graphe hiérarchique des chevauchements est quant à lui, même si on ne donne pas d'algorithme linéaire de construction, de taille linéaire en la norme de l'instance tout en étant plus petit que la taille du graphe des chevauchements correspondant.

7.3 Graphe Glouton (Superstring Graph)

Dans la section précédente, on a présenté différent parcours d'un arbre d'Aho-Corasick généralisé que l'on peut appliquer sur le graphe hiérarchique des chevauchements étendu.

On va présenter dans cette section un nouveau graphe que l'on va plonger dans le graphe hiérarchique des chevauchements étendu et qui va permettre de générer les solutions de l'algorithme glouton pour SCCS : le graphe glouton. Ces résultats ont été publiés dans (V). À l'origine, on a défini ce graphe en tant que « Graphe des nœuds internes » dans (XII). Ce graphe servait à l'origine pour reconstruire l'arbre des suffixes d'un ensemble de mots inconnus.

Dans une première partie, on va voir une définition du graphe glouton, ainsi que son lien avec l'algorithme glouton pour SCCS. Ensuite, on va donner une définition formelle de ce graphe glouton ce qui va nous permettre enfin de donner un algorithme linéaire pour construire une solution de l'algorithme glouton pour SCCS. Il est intéressant de remarquer que l'on va présenter deux définitions différentes pour définir le même graphe glouton. On présente ici les deux définitions car la première sert à comprendre les propriétés du graphe glouton alors que la deuxième définition est une définition constructive qui nous permet de justifier la construction linéaire de ce graphe.

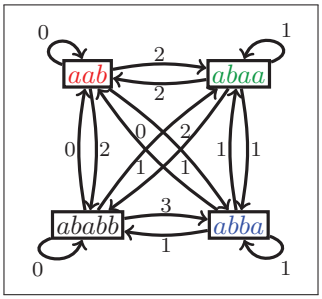
7.3.1 Lien entre le graphe glouton et l'algorithme glouton

Pour l'instant, on va se concentrer sur le problème SCCS et sur l'algorithme $Greedy_{CCS}$ (voir section 4.1.1 (page 86)).

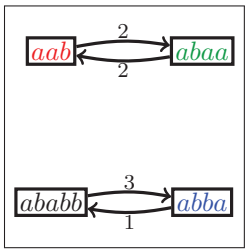
On a vu dans la partie 7.2, que pour une couverture cyclique C du graphe des chevauchements, on pouvait obtenir une Red-Blue Couverture Cyclique associée à C qui utilise la même permutation des nœuds. On a donc une équivalence entre une Red-Blue Couverture Cyclique associée à C et la couverture de mots circulaires $sol_{HCC \to CCS}(C)$ (défini dans la section 2.5). On a donc que chercher l'ensemble des couvertures circulaires de $PSCCS(P)$, revient à chercher des Red-Blue Couvertures Cycliques sur $EHOG(P)$.

Soit $w \in PSCCS(P)$. On note $G_P(w)$ le sous-graphe de $EHOG(P)$ dont l'ensemble des nœuds et l'ensemble des arcs est exactement l'ensemble des nœuds et l'ensemble des arcs de la Red-Blue Couverture Cyclique $RB-CC(sol_{CCS\to HCC}(w))$ pour la couverture de mots circulaires w.

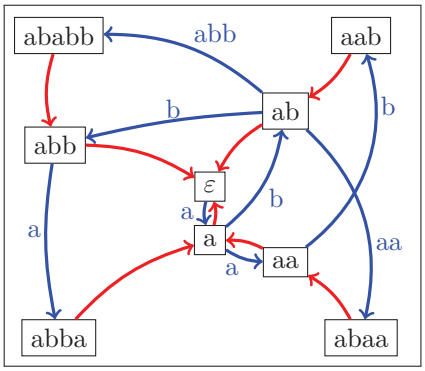
Remarque 7.26 Pour un mot w de $PSCCS(P)$, $RB-CC(sol_{CCS\to HCC}(w))$ est la Red-Blue couverture cyclique du $EHOG(P)$ qui correspond à w (voir Figure 7.14).



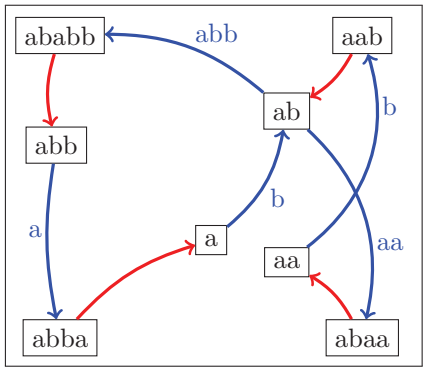
(a) Graphe de chevauchements de P



(b) Couverture cyclique $sol_{CCS\to HCC}(w)$ sur $OG(P)$



(c) Graphe hiérarchique des chevauchements étendu de P



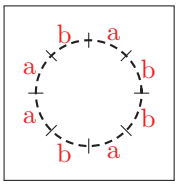
(d) Le graphe $G_P(w)$

FIGURE 7.14 – Exemple de graphe $G_P(w)$ pour $P = \{ababb, abba, aab, abaa\}$ et $w = \{\langle ababb\rangle, \langle aab\rangle\} = \{\langle ababb \oplus abba\rangle, \langle aab \oplus abaa\rangle\} \in PSCCS(P)$. La figure (d) est la Red-Blue Couverture Cyclique $RB-CC(sol_{CCS\to HCC}(w))$ sur $EHOG(P)$, c'est-à-dire $G_P(w)$. On a rajouté sur les figures (c) et (d) les labels sur les arcs bleus pour passer d'un mot à l'autre pour pouvoir lire les mots circulaires de w sur le graphe $G_P(w)$.

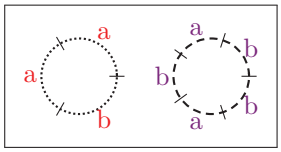
Comme toute solution de l'algorithme $Greedy_{CCS}$ est une solution de $PSCCS(P)$, on peut construire $G_P(w)$ pour chaque solution w de $Greedy_{CCS}$. On a alors la proposition suivante :

Proposition 7.27 ***voir Figure 7.15*** Soient w_1 et w_2 deux solutions de l'algorithme $Greedy_{CCS}$ sur l'ensemble de mots P, on a alors que $G_P(w_1) = G_P(w_2)$.

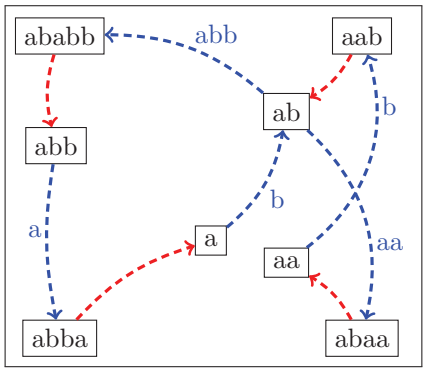
La proposition 7.27 établit une propriété clé sur les solutions de l'algorithme glouton pour SCCS. On va donner une première explication qui va permettre de donner l'intuition au lecteur et ensuite on va donner une preuve plus formelle mais moins explicative.



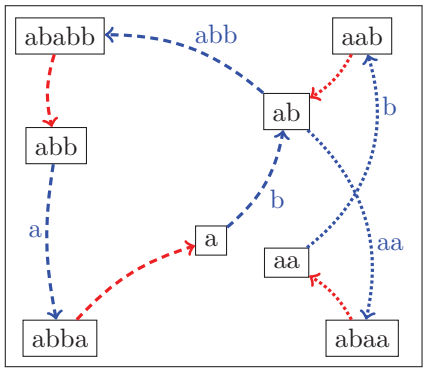
(a) w_1 où $|w_1| = 1$



(b) w_2 où $|w_2| = 2$



(c) $G_P(w_1)$



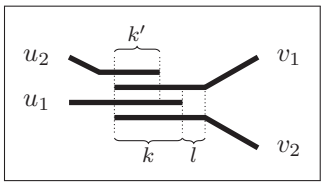
(d) $G_P(w_2)$

FIGURE 7.15 – Exemple de deux solutions $w_1 = \{\langle pr(aab \oplus ababb \oplus abba \oplus abaa, aab \oplus ababb \oplus abba \oplus abaa)\rangle\}$ et $w_2 = \{\langle pr(aab \oplus abaa, aab \oplus abaa)\rangle, \langle pr(ababb \oplus abba, ababb \oplus abba)\rangle\}$ de l'algorithme glouton pour SCCS pour $P = \{ababb, aab, abaa, abba\}$. La solution de la figure (a) a un seul mot circulaire qui est représenté avec des tirets sur la figure (c). La solution de la figure (b) a deux mots circulaires qui sont représentés l'un en pointilléet l'autre avec des tirets sur la figure (d).

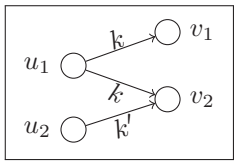
Preuve ***Explication*** Soit w une solution de l'algorithme glouton $Greedy_{CCS}$ pour SCCS. Nous allons expliquer pourquoi les choix faits par l'algorithme $Greedy_{CCS}$, tant que le chevauchement maximal est respecté, n'influencent pas $G_P(w)$.

L'algorithme $Greedy_{CCS}$ doit faire un choix quand plusieurs chevauchements de même longueur sont dépendants, *i.e.* que les chevauchements ont soit le premier mot en commun soit le deuxième mot en commun. Il est clair que deux chevauchements qui n'ont pas le même mot de début ou le même mot de fin sont indépendants et peuvent être pris l'un à la suite de l'autre, ce qui n'influence pas le graphe.

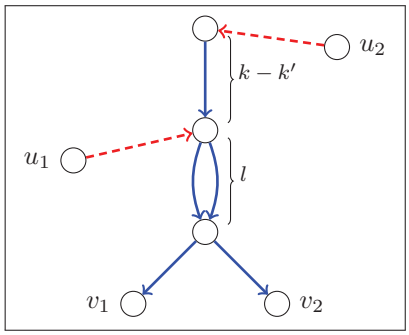
Soient $ov(u_1, v_1)$ et $ov(u_1, v_2)$ deux chevauchements dépendants qui sont un choix pour l'algorithme $Greedy_{CCS}$ tels que $ov(u_1, v_1)$ est le chevauchement choisi par $Greedy_{CCS}$ pour obtenir w. Soit u_2 le prédécesseur de v_2 dans w. Par la définition de $Greedy_{CCS}$, on a que $|ov(u_1, v_1)| \geq |ov(u_2, v_2)|$. On a alors qu'il existe un Red-Blue path de u_2 à v_1 dans $G_P(w)$. En effet, comme $|ov(u_1, v_1)| \geq |ov(u_2, v_2)|$, on a que $ov(u_2, v_2)$ est un préfixe de $ov(u_1, v_2)$, et donc un préfixe de $ov(u_1, v_1)$. On a alors que dans le $EHOG(P)$, le nœud $ov(u_2, v_2)$ est un ancêtre de $ov(u_1, v_1)$ et $ov(u_1, v_1)$ est un nœud sur les Red-Blue paths de u_1 à v_1 et de u_2 à v_2. On a alors que les deux Red-Blue paths de u_1 à v_2 et u_2 à v_1 sont dans $G_P(w)$ (voir Figure 7.16).



(a) sur les mots



(b) dans le graphe des chevauchements



(c) dans le HOG et dans $G_P()$

FIGURE 7.16 – Exemple avec trois chevauchements : deux qui sont dépendants : de u_1 vers v_1 et de u_1 vers v_2 et un autre chevauchement de u_2 vers v_2.

Preuve Soient w_1 et w_2 deux solutions différentes de l'algorithme $Greedy_{CCS}$ sur P. On pose $G_P(w_1) := (V_1, R_1 \cup B_1)$ et $G_P(w_2) := (V_2, R_2 \cup B_2)$.

On va procéder par l'absurde. On assume que $G_P(w_1) \neq G_P(w_2)$; on a alors que $((R_1 \cup B_1) \setminus (R_2 \cup B_2)) \cup ((R_2 \cup B_2) \setminus (R_1 \cup B_1)) \neq \emptyset$. On peut assumer sans perte de généralisation que $(R_1 \cup B_1) \setminus (R_2 \cup B_2) \neq \emptyset$. Soit $e := (u, v)$ un arc de $(R_1 \cup B_1) \setminus (R_2 \cup B_2)$ tel que $max(|u|, |v|)$ est maximum par rapport à tous les arcs de $(R_1 \cup B_1) \setminus (R_2 \cup B_2)$. Comme w_1 et w_2 sont deux couvertures circulaires solutions de l'algorithme $Greedy_{CCS}$, on a alors que w_1 et w_2 sont dans $PSCCS(P)$ et dont il existe deux permutations σ_1 et σ_2 telles que $w_1 = CC(P, \sigma_1)$ et $w_2 = CC(P, \sigma_2)$. Par la définition de $G_P(w_1)$, il existe i et j tels que $\sigma_1(i) = j$ et e est un arc du Red-Blue path de s_i à s_j (formellement $e \in RB - path(s_i, s_j)$).

On a alors deux cas :

Case $e \in B_1$:
Soit i_2 le prédécesseur de j dans σ_2 (*i.e.*, $j = \sigma_2(i_2)$). On a alors que $ov(s_{i_2}, s_j)$ est un préfixe de $ov(s_i, s_j)$, sinon comme l'algorithme glouton choisit toujours les plus grands chevauchements il existerait $j_1 := \sigma_1(i_2)$, tel que $|ov(s_i, s_j)| < |ov(s_{i_2}, s_{j_1})|$, ce qui voudrait dire que e ne serait pas maximal. Comme $ov(s_{i_2}, s_j)$ est un préfixe de $ov(s_i, s_j)$ on a que $e \in RB - path(s_{i_2}, s_j)$ et donc que $e \in B_2$, ce qui contredit l'hypothèse.

Case $e \in R_1$:
Soit j_2 le successeur de i dans σ_2 ($j_2 := \sigma_2(i)$). En reprenant l'argument du choix glouton, il est clair que $ov(s_i, s_{j_2})$ doit être un suffixe de $ov(s_i, s_j)$ et donc que $e \in R_2$. On a alors aussi une absurdité.

Ceci termine la preuve. ■

On a donc que l'ensemble des solutions de l'algorithme $Greedy_{CCS}$ peut être résumé dans un graphe que l'on appelle le graphe glouton.

Définition 7.28 ***Graphe glouton (voir Figure 7.17)*** Soit P un ensemble de mots. Le *graphe glouton* de P, noté $SG(P)$, est le graphe $G_P(w)$ pour toute solution w de l'algorithme $Greedy_{CCS}$.

Remarque 7.29 On peut construire le graphe glouton pour tout arbre d'Aho-Corasick généralisé de S avec $Ov(P) \cup P \subseteq S$ et donc on peut le construire sur $HOG(P)$.

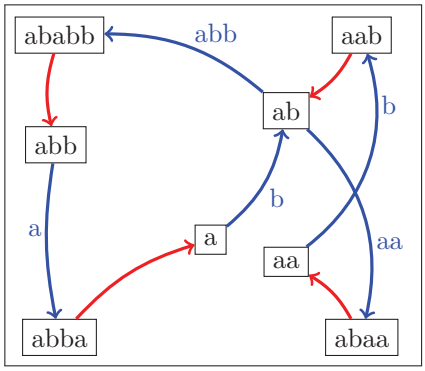


FIGURE 7.17 – Exemple de graphe glouton pour $P = \{ababb, aab, abaa, abba\}$.

Comme le graphe glouton est créé à partir d'une Red-Blue Couverture Cyclique, on a que dans chaque composante connexe du graphe glouton, le nombre d'arcs qui arrivent dans un nœud est identique au nombre d'arcs qui sortent de ce même nœud. On a donc que chaque composante connexe du graphe glouton est eulérienne.

Proposition 7.30 Le graphe glouton d'un ensemble de mots est semi-eulérien.

Soit C un ensemble de circuits d'un graphe G. On dit que C *couvre* le graphe G si chaque arc de G est inclus dans un seul circuit de C.

Proposition 7.31 Soit P un ensemble de mots. L'ensemble des ensembles de circuits qui couvrent $SG(P)$ est égal à l'ensemble des ensembles de circuits $RB - CC(sol_{CCS \to HCC}(w))$ pour w une solution de $Greedy_{CCS}$ pour P.

Pour prouver la proposition 7.31, on va commencer par définir un lemme. Un *croisement* de $SG(P)$ est un nœud qui a des arcs entrants rouges et bleus et des arcs sortants rouges et bleus.

Lemme 7.32 ***Absence de croisement*** Le graphe glouton d'un ensemble de mots n'admet aucun croisement.

Preuve ***Preuve (Lemme 7.32)*** Soit P un ensemble de mots. On cherche à prouver que $SG(P)$ n'admet pas de croisement. Comme seul les nœuds qui correspondent aux mots de P peuvent avoir un arc rouge sortant et un arc bleu entrant, les nœuds de $Ov^*(P) \setminus P$ n'admettent aucun croisement.

Comme P est factor-free, les nœuds de P sont des feuilles dans les arbres (V, R) et (V, B) où $EHOG(P) := (V, R \cup B)$ et donc aucun arc rouge ne peut entrer dans ce nœud ni aucun arc bleu peut en sortir, ils n'admettent donc aucun croisement non plus. ■

Preuve ***Preuve (Proposition 7.31)*** Pour comprendre cette égalité, il suffit de noter qu'il y a un fort lien entre le fait que l'algorithme $Greedy_{CCS}$ choisisse le plus grand chevauchement disponible et l'absence de croisement dans $SG(P)$.

Supposons qu'il existe un ensemble de circuits C qui couvre G et aucune solution w de $Greedy_{CCS}$ tel que $C = RB - CC(sol_{CCS\to HCC}(w))$. On a alors qu'à un moment C diffère des choix possibles par l'algorithme $Greedy_{CCS}$, et prend un chevauchement qui serait impossible pour $Greedy_{CCS}$. On va regarder alors le chevauchement le plus grand qui diffère entre ceux pris de C et de l'algorithme $Greedy_{CCS}$. Posons que le choix fait par C est le chevauchement de v_1 sur v_4 alors qu'un choix pouvant être fait par $Greedy_{CCS}$ est le chevauchement de v_3 vers v_4 avec $|ov(v_1, v_4)| < |ov(v_3, v_4)|$. Posons alors v_2 le mot utilisé par C pour faire le chevauchement de v_3 vers v_2. On a alors que $RB - path(v_1, v4)$ et $RB - path(v_3, v_2)$ sont dans C et que $|ov(v_1, v_2)| < |ov(v_3, v_4)|$ (car sinon le chevauchement de v_1 vers v_2 serait le plus grand chevauchement qui diffère). On a alors que $ov(v_3, v_4)$ est un croisement dans $SG(P)$ (voir Figure 7.18). On a donc une absurdité.

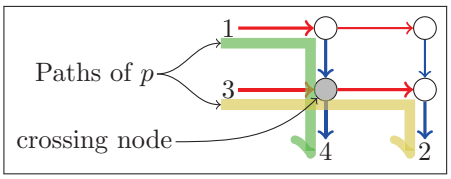


FIGURE 7.18 – Illustration de la preuve de la Proposition 7.31. Cette figure montre quatre nœuds v_1, v_2, v_3 et v_4 et les nœuds (cercles) représentant leurs chevauchements. Le nœud $ov(v_3, v_4)$ (nœud en gris) est un exemple de croisement.

Réciproquement, supposons qu'il existe une solution w de $Greedy_{CCS}$ et qu'il n'existe pas d'ensemble de circuits C qui couvre G tel que $C = RB - CC(sol_{CCS\to HCC}(w))$. On a alors qu'il existe un arc de $RB - CC(sol_{CCS\to HCC}(w))$ qui n'est pas un arc de $SG(P)$. Prenons $e := (u, v)$ un de ces arcs avec l'extrémité v la plus longue dans le graphe hiérarchique des chevauchements. Par définition de $HOG(P)$, $e \in R$. Soient v_1 et v_2 deux nœuds de P tels que $e \in RB - path(v_1, v_2)$. Comme e n'est pas dans $SG(P)$, pour tout mot u de P tel que $RB - path(v_1, u)$ est dans G_P, $u \neq v_2$. On a alors qu'il existe v_4 un mot de P tel que $RB - path(v_1, v_4)$ est dans $SG(P)$ et $|ov(v_1, v_2)| < |ov(v_1, v_4)|$. Comme w est une solution de l'algorithme glouton $Greedy_{CCS}$, il existe v_3 dans P et non dans G_P tel que $RB - path(v_3, v_4)$ est dans $RB - CC(sol_{CCS\to HCC}(w))$ et $|ov(v_1, v_4)| < |ov(v_3, v_4)|$. Alors il existe un arc de $RB - CC(sol_{CCS\to HCC}(w))$ et qui n'est pas dans $SG(P)$ qui soit d'extrémité plus profonde que e. On obtient alors aussi une contradiction. ■

Conclusion de la sous-section Dans cette sous-section, on a défini le graphe glouton, un nouveau graphe que l'on a ancré dans le graphe hiérarchique des chevauchements étendu. On a montré que toutes les solutions de l'algorithme glouton pour SCCS sont des parcours de ce graphe, et qu'inversement tous les parcours de ce graphe sont des solutions de l'algorithme glouton pour SCCS.

7.3.2 Construction du graphe glouton

Maintenant que l'on a défini le graphe glouton pour un ensemble de mots, on va le caractériser et donner un algorithme linéaire en la taille de l'instance pour le construire. Pour cet algorithme, on définit et calcule des poids (respectivement $n(.)$ et $d(.)$) pour chaque nœud du $EHOG(P) := (V, R \cup B)$. Enfin, on utilisera ces poids pour calculer le sous-ensemble des arcs du graphe hiérarchique des chevauchements étendu de P qui vont appartenir au graphe glouton de P.

Comme l'algorithme $Greedy_{CCS}$ sélectionne les chevauchements par longueurs décroissantes, on peut voir cela comme un algorithme récursif qui progresse le long des arcs de $EHOG(P)$. En effet, en calculant la longueur de chaque mot qui est un nœud du $EHOG(P)$, on peut obtenir une profondeur sur le graphe hiérarchique des chevauchements étendu. Utilisons ce côté récursif de $Greedy_{CCS}$ pour expliquer le graphe glouton.

Soit v un nœud du graphe hiérarchique des chevauchements étendu ; quand $Greedy_{CCS}$ regarde les chevauchements de taille $|v|$, il a déjà regardé les chevauchements de tailles plus grandes. On définit deux poids $n()$ et $d()$ tels que :

— $n(v)$ est le nombre de mots de P ayant v comme **suffixe** et qu'on a pas encore fusionné par la **droite** avec un autre mot de P en utilisant un plus grand chevauchement.

— $d(v)$ est le nombre de mots de P ayant v comme **préfixe** et qu'on a pas encore fusionné par la **gauche** avec un autre mot de P en utilisant un plus grand chevauchement.

En d'autres mots, $n(v)$ est le nombre de feuilles dans le sous-arbre de v dans (V, R) que l'on a pas encore fusionnées par la droite et $d(v)$ est le nombre de feuilles dans le sous-arbre de v dans (V, B) que l'on a pas encore fusionnées par la gauche.

Pour commencer la caractérisation du graphe glouton, on remarque qu'une feuille dans les arbres (V, R) et (V, B) est un mot de P. Pour ces nœuds, on pose alors $n(v) := 1$ et $d(v) := 1$.

Pour les autres nœuds, on va utiliser une récursion sur $Greedy_{CCS}$. En effet, on peut déterminer combien de mots de P n'ont pas encore été fusionnés avec un autre mot en regardant le poids $n(.)$ des enfants de v dans (V, R) et en regardant le poids $d(.)$ des enfants de v dans (V, B) (voir Figure 7.19).

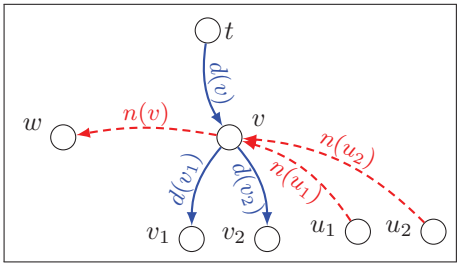


FIGURE 7.19 – Schéma général des arcs entrants et sortants des nœuds v du EHOG. Les arcs rouges correspondent aux arcs de R et les arcs bleus à ceux de B. Si le poids est de zéro, l'arc n'est pas ajouté dans le graphe glouton.

Soit $a(v)$ la différence entre la somme des $n(u')$ des fils u' de v dans (V, R) et la somme

des $d(u)$ des fils u de v dans (V, B) :

$$a(v) := \sum_{u' \in Children_{(V,R)}(v)} n(u') - \sum_{u \in Children_{(V,B)}(v)} d(u).$$

On a alors que la valeur absolue de $a(v)$ est le plus grand nombre de mots de P que $Greedy_{CCS}$ peut fusionner avec d'autres mots de P en utilisant le chevauchement v. Une fois la valeur $a(v)$ calculée, on peut calculer $n(v)$ et $d(v)$:

$$\text{Si } a(v) \geq 0, \text{ on a que } \begin{cases} n(v) := a \\ d(v) := 0, \end{cases}$$

$$\text{Si } a(v) < 0, \text{ on a que } \begin{cases} n(v) := 0 \\ d(v) := -a, \end{cases}$$

Avec cette définition récursive des poids $n(.)$ et $d(.)$, on peut construire le graphe glouton à partir du graphe hiérarchique des chevauchements étendu de P. Dans tous les cas, si le poids d'un arc est de 0, on ne fait pas apparaître cet arc dans le graphe glouton. De manière similaire, si un nœud n'est touché par aucun arc du graphe glouton, il est dit isolé, il n'apparaît pas dans le graphe glouton.

Proposition 7.33 Soient P un ensemble de mots et $EHOG(P) = (V, R, B)$ le graphe hiérarchique des chevauchements étendu de P. On pose $G' = (V', R', B')$ tel que

$$V' = V \setminus U$$

$$R' = \{(u, w)^{n(u)} \mid u \in V,\ w \text{ le parent de } u \text{ dans } (V, R)\}$$

$$B' = \{(t, v)^{d(v)} \mid v \in V,\ t \text{ le parent de } v \text{ dans } (V, B)\}$$

et $U = \{v \in V \mid v \textit{ n'est pas une extrémité d'un arc de } R' \cup B'\}$ où $n(.)$ et $d(.)$ sont les poids sur les nœuds définis plus haut et e^k est une k-multi-arête avec e un arc et k un entier. On a alors que G' est le graphe glouton de P.

Preuve L'égalité des définitions vient de la définition de $n(.)$ et de $d(.)$. ■

Remarque 7.34 Soient e un arc d'un graphe et k un entier. La notation e^k de k-multi-arête peut être comprise comme k copies du même arc e ou comme un arc e muni d'une pondération égale à k.

En supposant que l'on a pré-calculé les poids $n(.)$ et $d(.)$ de tous les nœuds du graphe hiérarchique des chevauchements étendu, la proposition 7.33 donne une construction immédiate du graphe glouton. Or, comme le pré-calcul des poids $n(.)$ et $d(.)$ est linéaire en la taille du graphe hiérarchique des chevauchements étendu, on obtient le théorème suivant.

Théorème 7.35 Le graphe glouton de P est de taille linéaire en la taille du graphe hiérarchique des chevauchements étendu de P. De plus, si on a déjà construit le graphe hiérarchique des chevauchements étendu, on peut construire le graphe glouton en temps linéaire en $||P||$.

Preuve La proposition 7.33 nous donne une définition récursive de graphe glouton. En effet, comme le graphe hiérarchique des chevauchements étendu est de taille linéaire en $||P||$, on peut construire les pondération $n(.)$ et $d(.)$ en temps linéaire en $||P||$ et donc le graphe glouton en temps linéaire en $||P||$. ■

Conclusion de la sous-section On a donné dans cette sous-section, une définition récursive et constructive de l'algorithme glouton. On a montré de plus qu'en utilisant cette définition, en ayant construit préalablement le graphe hiérarchique des chevauchements étendu, on peut construire le graphe glouton en temps linéaire en la norme de l'instance.

7.3.3 Construction linéaire de l'algorithme glouton pour SCCS

On vient de montrer dans la sous-section précédente que si on a le graphe hiérarchique des chevauchements étendu d'un ensemble de mots P, on peut construire le graphe glouton de P en temps linéaire en la norme de P.

On va alors définir l'algorithme 7.2 comme suit. Soit P un ensemble de mots.

- On va commencer par construire le graphe hiérarchique des chevauchements étendu de P : le théorème 7.25 nous dit que l'on peut construire $EHOG(P)$ en temps linéaire en $||P||$.
- On va construire le graphe glouton de P que l'on va ancrer sur le graphe hiérarchique des chevauchements étendu de P : le théorème 7.35 nous dit que l'on peut construire le graphe glouton de P en temps linéaire à l'aide du graphe hiérarchique des chevauchements étendu de P.
- On prend un parcours couvrant du graphe glouton de P que l'on labellise en une couverture circulaire de mots C, solution de l'algorithme glouton pour SCCS : d'après la proposition 7.31, tout parcours semi-eulérien du graphe glouton est une solution de l'algorithme glouton pour SCCS.
- La couverture circulaire de mots C est alors une solution optimale de SCCS : le corollaire 4.24 nous dit que l'algorithme $Greedy_{CCS}$ est exact pour SCCS.

On obtient le théorème 7.36.

Théorème 7.36 L'algorithme 7.2 résout SCCS en temps et en espace linéaire en la taille de l'instance.

Remarque 7.37 Le théorème 7.36 améliore la complexité du meilleur algorithme connu pour résoudre SCCS en passant de $O(|P|^3 + ||P||)$ en $O(||P||)$.

On sait, par la proposition 7.30, que le graphe glouton de P est semi-eulérien. On a donc que chaque composante connexe du graphe glouton est eulérienne. Comme on peut trouver un parcours semi-eulérien sur un graphe semi-eulérien en temps linéaire en

Algorithme 7.2 : L'algorithme du graphe glouton pour SCCS.

Entrée : Soit P un ensemble de mots.
1 Construire $SG(P)$ le graphe glouton de P;
2 On crée un ensemble vide w;
3 Calculer un ensemble de circuits $c = (c_1, \ldots, c_n)$ couvrant $SG(P)$;
4 **for** $i \in [1, n]$ **do**
5 On traverse c_i : on liste les mots de P qui sont des nœuds dans c_i et on crée w_i en concaténant les $pr(s_j, s_k)$ où s_k est le successeur de s_j;
6 on ajoute w_i à w;
7 **return** w

le nombre d'arcs du graphe, on peut trouver de manière linéaire en la norme de P, un parcours semi-eulérien sur le graphe glouton. Or, un parcours semi-eulérien du graphe glouton permet de trouver une des solutions de $Greedy_{CCS}$ avec le plus petit nombre de mots circulaires. En effet, si ce n'était pas le cas, il existerait une solution de l'algorithme glouton pour SCCS avec strictement moins de mots circulaires et donc en ancrant cette solution sur le graphe hiérarchique des chevauchements, on aurait un chemin entre deux composantes connexes du graphe glouton, ce qui est absurde d'après la proposition 7.31. Dans la suite, on va expliciter ce résultat.

Soit $SG(P)$ le graphe glouton de P. On peut alors décomposer $SG(P)$ en composantes connexes : $SG(P) = \{C_1, \ldots, C_m\}$. Soit W un ensemble de circuits qui couvrent $SG(P)$, on peut alors partitionner l'ensemble des circuits de W de telle façon que $W := S_1 \cup \ldots \cup S_m$ et pour tout i entre 1 et m, S_i couvre C_i.

L'algorithme 7.2, sur la troisième ligne (en bleu) calcule un ensemble de circuits qui couvrent $SG(P)$. L'algorithme glouton $Greedy_{CCS}$ est un algorithme non-déterministe dans le sens où pour une même instance, il peut donner deux solutions différentes. D'après le théorème 4.23, on sait que toutes les solutions de $Greedy_{CCS}$ pour une même instance sont de la même taille. Or, comme on peut le voir sur la Figure 7.15, le nombre de circuits que l'on trouve peut changer.

On peut alors définir le problème suivant :

Problèmme 7.38 Min-Cardinality-Greedy-SCCS Soient P un ensemble de mots. On cherche une couverture circulaire de mots C solution de l'algorithme glouton $Greedy_{CCS}$ qui soit de cardinalité la plus petite possible, *i.e.* on cherche à minimiser $|C|$.

Ce problème consiste à chercher la meilleure des solutions dans l'algorithme $Greedy_{CCS}$, c'est-à-dire la solution qui regroupe le plus de mots ensemble.

Comme on a que chaque composante de $SG(P)$ est eulérienne, on peut trouver en temps linéaire sur chaque composante connexe, c'est-à-dire sur $SG(P)$, un ensemble de circuits semi-eulériens W' sur $SG(P)$, c'est-à-dire que tel que $W' := S'_1 \cup \ldots \cup S'_m$ et pour tout i entre 1 et m, $|S'_i| = 1$. On a alors que W' est une couverture cyclique de $SG(P)$. On peut alors définir l'algorithme 7.3 qui correspond à l'algorithme 7.2 où on a seulement remplacé la ligne bleue. En effet, la seule différence est qu'au lieu de chercher un parcours couvrant sur le graphe glouton, on cherche un parcours semi-eulérien (*i.e.* une couverture cyclique) sur le graphe glouton pour minimiser le nombre de circuits.

On obtient alors le corollaire suivant :

Algorithme 7.3 : L'algorithme du graphe glouton pour Min-Cardinality-Greedy-SCCS.

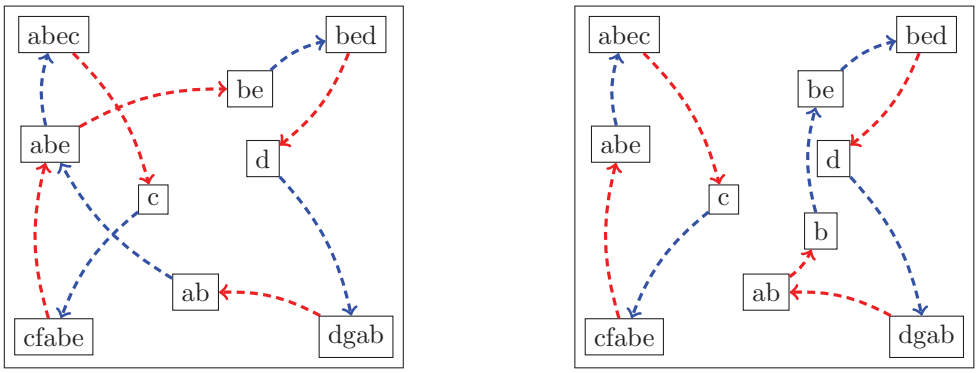
Entrée : Soit P un ensemble de mots.

Construire $SG(P)$ le graphe glouton de P;
On crée un ensemble vide w;
Calculer une Couverture Cyclique $c = (c_1, \ldots, c_n)$ de G_P;
for $i \in [1, n]$ **do**
    On traverse c_i : on liste les mots de P qui sont des nœuds dans c_i et on crée w_i en concaténant les $pr(s_j, s_k)$ où s_k est le successeur de s_j;
    on ajoute w_i à w;
return w

Corollaire 7.39 L'algorithme 7.3 résout Min-Cardinality-Greedy-SCCS en temps et en espace linéaire en la taille de l'instance.

Comme pour tout ensemble de mots P, $OPT_{\text{Min-Cardinality-Greedy-SCCS}}(P) \subseteq OPT_{SCCS}(P)$, on a que l'algorithme 7.3 résout aussi SCCS en temps linéaire en la taille de l'instance.

Comme on peut le voir sur la Figure 7.20, il existe certains ensembles de mots P tels que $OPT_{\text{Min-Cardinality-Greedy-SCCS}}(P) \neq OPT_{SCCS}(P)$. On a donc qu'il existe des solutions optimales de SCCS qui ne sont pas des solutions de l'algorithme $Greedy_{CCS}$.



(a) Solution optimale non gloutonne

(b) Solution optimale gloutonne

FIGURE 7.20 – Exemple de deux solutions optimales de SCCS pour $P = \{ababb, aab, abaa, abba\}$ où une est une solution de l'algorithme glouton pour SCCS et l'autre non. On montre le graphe $G_P(.)$ pour ces deux solutions.

Conclusion de la sous-section Dans cette sous-section, on a montré que l'on pouvait, en utilisant le graphe hiérarchique des chevauchements, ainsi que le graphe glouton, construire de manière linéaire en la norme de l'instance, l'ensemble des solutions optimales de SCCS. On a de plus montré que l'on pouvait construire toujours en la norme de l'instance, des solutions de l'algorithme glouton pour SCCS avec le moins de mots circulaires possibles.

Conclusion de la section Dans cette section, on a amélioré la complexité du meilleur algorithme connu pour résoudre SCCS en passant de $O(|P|^3 + ||P||)$ en $O(||P||)$.

Conclusion du chapitre En généralisant l'arbre d'Aho-Corasick et en l'appliquant sur un ensemble de mots et ses chevauchements maximaux, on a pu créer le graphe hiérarchique des chevauchements. On a montré que ce graphe tout en étant plus petit que le graphe des chevauchements, permettait de retrouver les informations contenues sur le graphe des chevauchements. On a en plus utilisé ce graphe pour définir et ancrer un nouveau graphe. Ce nouveau graphe, que l'on a appelé le graphe glouton, permet de générer les solutions de l'algorithme glouton pour SCCS et nous a permis d'améliorer la complexité du meilleur algorithme connu pour résoudre SCCS. Dans le chapitre 8, on va voir que l'on peut généraliser ce résultat pour l'appliquer à des variantes de SCCS.

Chapitre 8

Constructions linéaires des variantes de SCCS

Dans le chapitre 5, on s'est intéressé notamment aux variantes de SCCS dans le cas de l'ADN et du complémentaire renversé (SDCCS) et dans le cas de la multiplicité (Multi-SCCS). On s'est intéressé aussi à la variante de SMCS dans le cas où on interdisait certains chevauchements (Contraint-SMCS). Or dans le chapitre 7, on a vu un graphe (le graphe glouton) qui nous permet de générer des solutions de l'algorithme glouton pour SCCS.

Dans ce chapitre, on va montrer que pour chacun de ces trois cas, on peut définir un pendant au graphe glouton qui va permettre de générer toutes les solutions de l'algorithme glouton sur le problème regardé. Pour ancrer chacun de ces nouveaux graphes, on va utiliser les propriétés de chacun des problèmes vus au chapitre 5. Il est important de remarquer que pour chacun de ces problèmes, aucune construction de l'algorithme glouton n'est connue. Dans ce chapitre, on va omettre la plupart des preuves pour éviter d'alourdir le rapport car elles sont similaires à celles du chapitre 7.

Sommaire

8.1 Le cas du Complémentaire-renversé

Dans la section 5.1, on a défini le problème SDCCS où on cherche à trouver une ADN-couverture circulaire de mots d'un ensemble de mots de norme minimale. On a montré alors le lien entre le problème Max-PHCC sur le graphe des chevauchements reverse-complement et le problème SDCCS. Or, pour un ensemble de mots P sur $\{A, T, C, G\}$, on a que le graphe des chevauchements reverse-complement de P est un sous-graphe du graphe des chevauchements de $P \cup \overleftarrow{P}$. On va donc s'intéresser ici au graphe hiérarchique des chevauchements de $P \cup \overleftarrow{P}$.

Pour rappel, le graphe des chevauchements reverse-complement de P est un graphe qui a comme nœud chaque mot u de P ainsi que son complémentaire-renversé $\overleftarrow{u}$. L'idée en utilisant une solution de Max-PHCC sur le graphe des chevauchements reverse-complement de P est de trouver un parcours qui pour chaque mot u de P ne passe que par l'un des deux nœuds u ou $\overleftarrow{u}$.

On va ici utiliser une autre méthode pour construire l'algorithme glouton pour SDCCS. On va prendre le graphe hiérarchique des chevauchements de $P \cup \overleftarrow{P}$ et on va fusionner les nœuds u du $HOG(P \cup \overleftarrow{P})$ avec leur complémentaire-renversé $\overleftarrow{u}$. On va alors définir un nouveau graphe que l'on va parcourir comme le ferait le graphe glouton. Ces résultats ont été publiés dans (I) et (VI).

Pour créer ce graphe, on va commencer par redéfinir le problème SDCCS, c'est-à-dire la variante du problème SCCS pour le cas du complémentaire-renversé. En effet, comme on l'a fait pour le problème SCCS, on va pouvoir définir les solutions de SDCCS à l'aide de permutations. On se concentre ici sur le problème SDCCS mais on peut de même redéfinir les problèmes SDLS et SDCS à l'aide de permutations.

Pour commencer, on va redéfinir le problème SDCCS en modifiant la définition d'une ADN-couverture circulaire des mots d'un ensemble de mots. Un ADN-mot est un ensemble de deux mots $\{w, \overleftarrow{w}\}$. Pour simplifier la notation, on écrit un « ADN-mot w » pour « un ADN-mot $\{w, \overleftarrow{w}\}$ ». On obtient alors que pour un mot linéaire w, l'ADN-mot w est égal à l'ADN-mot de $\overleftarrow{w}$. La longueur de l'ADN-mot w est la longueur de w. On peut étendre la définition de l'ADN-mot à l'ADN-mot circulaire qui est l'ensemble composé de deux mots circulaires $\langle w \rangle$ et $\langle \overleftarrow{w} \rangle$. On dit qu'un mot x est une sous-chaîne d'un ADN-mot y s'il est sous-chaîne de y ou de $\overleftarrow{y}$. Une ADN-couverture circulaire de mots d'un ensemble de mots P est alors un ensemble C d'ADN-mots circulaires tel que pour tout mot s_i de P, il existe $\langle c_j \rangle \in C$ tel que s_i est une sous-chaîne de $\langle c_j \rangle$.

On peut alors définir une permutation spécifique au cas du reverse-complement et des couvertures circulaires de mots issues de ces permutations.

Définition 8.1 Soit P un ensemble de mots sur $\{A, T, C, G\}$.

1. (ADN-permutation)
   Une *ADN-permutation* σ de P est une permutation de $P \cup \overleftarrow{P}$ telle que pour tout x dans $P \cup \overleftarrow{P}$, $\sigma(\overleftarrow{\sigma(x)}) = \overleftarrow{x}$ et $\sigma(x) \neq \overleftarrow{x}$.
2. (ADN-permutation semi-circulaire)
   Une *ADN-permutation semi-circulaire* σ de P est une ADN-permutation de P telle que $Part_\sigma = 2$.
3. (ADN-superchaîne circulaire issue d'une ADN-permutation)
   Soit σ_c une ADN-permutation semi-circulaire de P. L'*ADN-superchaîne circulaire issue* de σ_c est l'ADN-mot circulaire défini pour tout $x \in P$,
   $$ADN - Circular(P, \sigma_c) \quad := \quad \langle pr(x, \sigma_c(x)).pr(\sigma_c(x), \sigma_c^2(x)) \dots pr(\sigma_c^{|P|-1}(x), x) \rangle.$$
4. (ADN-couverture circulaire de mots)
   Soit σ une ADN-permutation de P. L'*ADN-couverture circulaire de mots issue* de σ est
   $$ADN - CC(P, \sigma) := \bigcup_{\sigma_c \in Part_\sigma} \{ADN - Circular(P, \sigma_c)\}.$$

On pose $PDCC(P)$ l'ensemble des ADN-couvertures circulaires de mots de P issues d'une ADN-permutation. On a alors la proposition suivante.

Proposition 8.2 Soit P un ensemble de mots sur $\{A, T, C, G\}$. On a que

$$OPT_{\mathsf{SDCCS}}(P) \subseteq PDCC(P).$$

Preuve On suppose que $OPT_{\mathsf{SDCCS}}(P) \setminus PDCC(P) \neq \emptyset$ et on veut montrer que l'on arrive à un résultat absurde. Soit $C \in OPT_{\mathsf{SDCCS}}(P) \setminus PDCC(P)$. On peut trouver chaque mot de P comme sous-chaîne d'un élément de $C := \{C_1, \ldots, C_m\}$. Par la définition de C, chaque élément de C est un ADN-mot. On pose $\langle w_i \rangle \in C_i$ pour i entre 1 et m, et on a qu'il existe une partition de $P := \{P_1, \ldots, P_m\}$ telle que pour chaque s_j dans P_i, s_j ou $\overleftarrow{s_j}^{\varphi}$ est une sous-chaîne de $\langle w_i \rangle$. On peut alors décomposer chaque $\langle w_i \rangle$ en un cycle d'éléments de P par ordre d'apparition et construire une ADN-permutation semi-circulaire σ_i de P_i qui à chaque élément de P_i associe l'élément suivant dans $\langle w_i \rangle$. En prenant σ l'ADN-permutation de P telle que $\sigma := \sigma_1 \ldots \sigma_m$, on a que $||C|| \geq |ADN - CC(P, \sigma)|$. Si $||C|| = |ADN - CC(P, \sigma)|$ on a que $C = ADN - CC(P, \sigma) \in PDCC(P)$, ce qui contredit l'hypothèse. Dans le cas contraire, w_i n'est pas une solution optimale de SDCCS, ce qui est absurde. ■

On peut alors redéfinir le problème SDCCS à l'aide des ADN-permutations.

Problèmme 8.3 ***Shortest DNA Cyclic Cover of Strings*** **(SDCCS) *version 2***
Soit P un ensemble de mots sur $\{A, T, C, G\}$. On cherche une ADN-couverture circulaire de mots issue d'une ADN-permutation de P qui soit de norme minimale.

Maintenant que l'on a redéfini le problème SDCCS à l'aide des ADN-permutations, on va définir un nouveau graphe qui va nous permettre, à partir d'une ADN-permutation, de trouver un ensemble de cycles sur ce graphe. On peut remarquer que grâce à la proposition 5.15, pour $HOG(P \cup \overleftarrow{P}) = (V, R \cup B)$, on a que

$$B = \{(\overleftarrow{v}, \overleftarrow{u}) \mid (u, v) \in R\}.$$

L'information contenue dans R est donc redondante avec l'information contenue dans B. On peut alors, pour construire le graphe hiérarchique des chevauchements de $P \cup \overleftarrow{P}$, se restreindre à conserver uniquement l'ensemble des nœuds V et les arcs B entre un nœud et un nœud plus profond.

On veut fusionner les nœuds du graphe hiérarchique des chevauchements de $P \cup \overleftarrow{P}$ pour ne garder qu'un unique représentant de chaque paire constituée d'un mot et de son complémentaire-renversé. On pose X l'ensemble des représentants des nœuds du graphe hiérarchique des chevauchements de $P \cup \overleftarrow{P}$, c'est-à-dire de $(P \cup \overleftarrow{P}) \cup Ov(P \cup \overleftarrow{P})$, tel que le mot u de $(P \cup \overleftarrow{P}) \cup Ov(P \cup \overleftarrow{P})$ est dans X si et seulement si u est le plus petit mot par l'ordre lexicographique entre lui-même et son complémentaire-renversé, *i.e.* $u \preceq \overleftarrow{u}$. On a alors que $X \cup \overleftarrow{X} = (P \cup \overleftarrow{P}) \cup Ov(P \cup \overleftarrow{P})$.

Remarque 8.4 En prenant un autre ensemble X' de représentant de $(P \cup \overleftarrow{P}) \cup Ov(P \cup \overleftarrow{P})$, on obtiendrait des résultats équivalents.

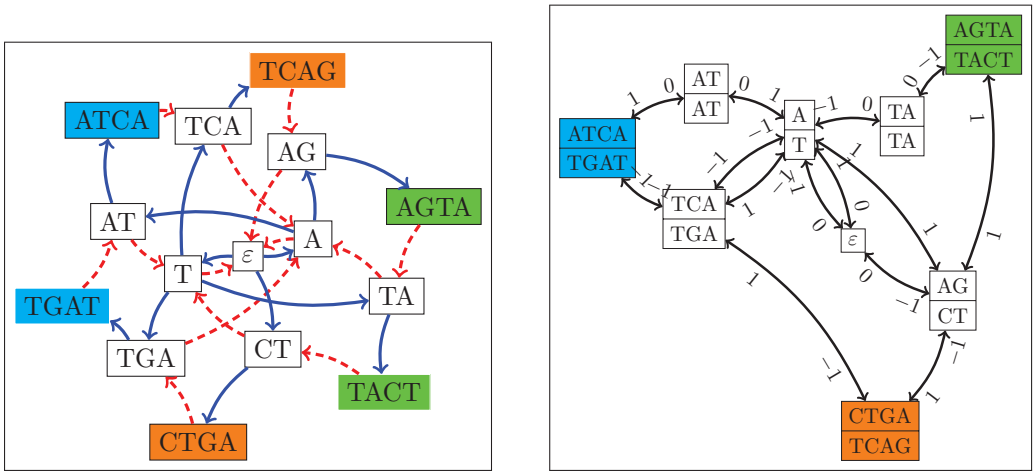
Pour un mot w de $(P \cup \overleftarrow{P}) \cup Ov(P \cup \overleftarrow{P})$, on note $a(w)$ comme le représentant dans X du mot w, et $d(w)$ comme l'élément de $\{-1, 0, 1\}$ tel que

$$d(w) = \begin{cases} 0 & \text{Si } w = \overleftarrow{w} \\ 1 & \text{Si } w \neq \overleftarrow{w} \text{ et } w \in X \\ -1 & \text{Si } w \neq \overleftarrow{w} \text{ et } w \notin X \end{cases}$$

L'application $a(.)$ peut être vue comme un pointeur qui pour un mot de $(P \cup \overleftarrow{P}) \cup Ov(P \cup \overleftarrow{P})$ donne son représentant dans X et l'application $d(.)$ peut être vue comme une vérification qu'un mot est dans X ou pas. La valeur 0 indique que le mot et son complémentaire-renversé sont égaux.

Définition 8.5 ***Graphe hiérarchique des chevauchements fusionné (voir Figure 8.1)*** Soient P un ensemble de mots et $HOG(P \cup \overleftarrow{P}) = (V, R \cup B)$.

Le *graphe hiérarchique des chevauchements fusionné* de P, dénoté par $GHOG(P)$, est le graphe non-orienté labellisé sur ses arêtes (V, E, l), tel que $V = X$ et pour tout $(u, v) \in B$, $(a(u), a(v)) \in E$ et $l((a(u), a(v))) : \begin{cases} a(u) \mapsto d(u) \\ a(v) \mapsto d(v) \end{cases}$.



(a) $HOG(P)$ (b) $GHOG(P)$

FIGURE 8.1 – Exemple de graphe hiérarchique des chevauchements (a) et de graphe hiérarchique des chevauchements fusionné (b) pour $P = \{ATCA, TCAG, AGTA\}$. On a associé à chaque mot de P une couleur différente.

Comme on a défini le RB-path sur le graphe hiérarchique des chevauchements pour aller d'un nœud de P vers un autre nœud de P, on définit le RF-path sur le graphe hiérarchique des chevauchements fusionné. Le *RF-path* entre deux nœuds u et v de $((P \cup \overleftarrow{P}) \cap X)$ est l'ensemble d'arêtes, noté $RF - path(u, v)$, tel que :

$$RF - path(u, v) = \bigcup_{(x,y) \in p} \{(a(x), a(y))\}$$

où p le plus petit des RB-path de u vers v, de $\overleftarrow{u}$ vers v, de u vers $\overleftarrow{v}$ ou de $\overleftarrow{u}$ vers $\overleftarrow{v}$.

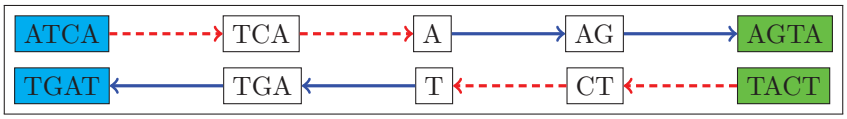
Remarque 8.6 Le RB-path sur le graphe hiérarchique des chevauchements est un chemin alors que le RF-path sur le graphe hiérarchique des chevauchements fusionné est un ensemble d'arêtes. Cet ensemble d'arêtes pour deux nœuds u et v forme un sous-graphe tel qu'il existe un chemin eulérien qui aille de u vers v dans ce graphe. Par la définition du RF-path, on a que $RF - path(u, v) = RF - path(v, u)$.

On a alors la propriété suivante sur la suite des labels le long d'un chemin eulérien du sous-graphe défini par un RF-path.

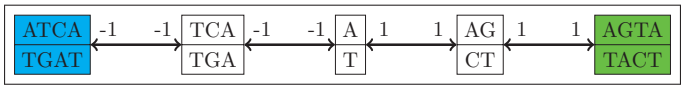
Proposition 8.7 Soient u et v deux mots de $((P \cup \overleftarrow{P}) \cap X)$. Pour $p = (e_1 = (u_1, u_2), \ldots, e_m = (u_m, u_{m+1}))$ le chemin eulérien de $RF - path(u, v)$ de u vers v, on a qu'il existe i entre 2 et m tel que :

- Pour tout $j < i$, $|u_j| \geq |u_{j+1}|$ et $l((u_{j-1}, u_j))(u_j) = l((u_j, u_{j+1}))(u_j)$,
- Pour tout $j \geq i$, $|u_j| \leq |u_{j+1}|$ et $l((u_{j-1}, u_j))(u_j) = l((u_j, u_{j+1}))(u_j)$,
- $l((u_{i-1}, u_i))(u_i) = -l((u_i, u_{i+1}))(u_i)$.

La figure 8.2 nous donne un exemple de RB-path sur le graphe hiérarchique des chevauchements et de chemin eulérien sur le RF-path sur le graphe hiérarchique des chevauchements fusionné.



(a) RB-path



(b) Chemin eulérien sur le RF-path

FIGURE 8.2 – Exemple de RB-path sur le graphe hiérarchique des chevauchements et de RF-path sur le graphe hiérarchique des chevauchements fusionné pour $P = \{ATCA, TCAG, AGTA\}$.

Soit C une ADN-couverture circulaire de mots de $PDCC(P)$. On pose alors $DG_P(C)$ comme étant le sous-graphe du graphe hiérarchique des chevauchements fusionné où l'ensemble des arêtes E est tel que

$$E = \bigcup_{u \in X} \{RF - Path(u, \sigma(u))\}$$

où σ est l'ADN-permutation de P telle que $C = ADN - CC(P, \sigma)$.

Comme pour la proposition 7.27 mais appliquée au cas du complémentaire-renversé, on a que pour deux solutions C_1 et C_2 de l'algorithme glouton pour SDCCS, les graphes $DG_P(C_1)$ et $DG_P(C_2)$ sont égaux. De plus, on peut prouver, comme dans le cas classique, que tout parcours de ce graphe correspond aussi à une solution de l'algorithme glouton pour SDCCS. Comme on peut construire le graphe hiérarchique des chevauchements fusionné d'un ensemble de mots P en temps linéaire en la norme P et que l'on peut construire le graphe correspondant à une solution de l'algorithme glouton pour SDCCS aussi en temps

linéaire en la norme de P, on obtient le théorème suivant qui est la version complémentaire-renversé du théorème 7.36.

Théorème 8.8 Il existe un algorithme qui résout exactement le problème SDCCS en temps linéaire en $||P||$.

Conclusion de la section Dans cette section, on a donné un algorithme linéaire en la norme de l'instance pour trouver une solution optimale pour le problème SDCCS. Pour cela, on a défini un nouveau graphe (le graphe hiérarchique des chevauchements fusionné) et on a construit sur ce graphe un sous-graphe qui va générer les solutions de l'algorithme glouton pour SDCCS. Comme l'algorithme glouton pour SDCCS est exact, on peut construire en temps linéaire en la norme de P une solution optimale du problème SDCCS. Le problème SDCCS est donc dans **P**.

8.2 Généralisation du graphe glouton

Dans cette partie, on va s'intéresser aux problèmes Multi-SCCS et Contraint-SMCS, c'est-à-dire à la variante de SCCS où on a en plus une multiplicité sur les mots et à la variante de SMCS où on a un ensemble de chevauchements interdits.

Dans la section 7.3, on a défini le graphe glouton sur le graphe hiérarchique des chevauchements. On va présenter une généralisation du graphe glouton : le graphe glouton généralisé. La volonté de généraliser le graphe glouton vient du fait que les problèmes Multi-SCCS et Contraint-SMCS sont des généralisations du problème SCCS. En effet, pour un ensemble de mots P qui est factor-free, on a que $OPT_{\mathsf{SCCS}}(P) = OPT_{\mathsf{Multi\text{-}SCCS}}(P, \{1\}^P)$ où $\{1\}^P$ est l'application de P vers $\{1\}$. De même, on a que $OPT_{\mathsf{SMCS}}(P) = OPT_{\mathsf{Contraint\text{-}SMCS}}(P, \emptyset)$ pour un ensemble de chevauchements interdits vide et $OPT_{\mathsf{SCCS}}(P) \subseteq OPT_{\mathsf{SMCS}}(P)$.

Le graphe glouton généralisé permettra ensuite de généraliser le lien qu'il y a entre les parcours du graphe glouton et les solutions de l'algorithme glouton pour le problème SCCS. Enfin, on va appliquer ces nouveaux résultats aux problèmes Multi-SCCS et Contraint-SMCS.

8.2.1 Graphe glouton généralisé

Commençons par définir le graphe glouton généralisé que l'on va plonger [1] dans l'arbre d'Aho-Corasick généralisé vu dans la section 7.2. Ce nouveau graphe nous permettra de générer les solutions d'un algorithme glouton pour un problème que l'on définira dans un deuxième temps.

Définition 8.9 ***Graphe glouton généralisé*** Soient S un ensemble de mots et m une multiplicité sur S. Le *graphe glouton généralisé*, noté $GSG(S, m)$, est le graphe orienté $G = (V, R, B)$ tel que

$$\begin{aligned} V &= S \setminus U \\ R &= \{(u, w)^{n(u)} \mid u \in S \text{ et } w \text{ est le parent de } u \text{ dans } (S, GAC_R(S))\} \\ B &= \{(t, v)^{d(v)} \mid v \in S \text{ et } t \text{ est le parent de } v \text{ dans } (S, GAC_B(S))\} \end{aligned}$$

1. l'un est sous-graphe de l'autre

où $U = \{v \in S \mid v \text{ n'est pas une extrémité d'un arc de } R \cup B\}$ et pour tout $v \in S$,

$$\begin{array}{rcl} n(v) & = & min(\{m(v) - a(v), 0\}) + a(v) \\ d(v) & = & min(\{m(v) - a(v), 0\}) + a(v) - m(v) \\ a(v) & = & \sum_{u' \in Children_{(V,R)}(v)} n(u') - \sum_{u \in Children_{(V,B)}(v)} d(u). \end{array}$$

On parle de graphe glouton généralisé car pour un ensemble de mots P, on a que $SG(P) = GSG(P \cup Ov(P), \{1\}^P)$ où $\{1\}^P$ est l'application de P vers $\{1\}$.

En utilisant l'arbre d'Aho-Corasick généralisé d'un ensemble de mots, on peut alors construire le graphe glouton généralisé qui correspond à une pondération de cet ensemble de mots.

Proposition 8.10 Soient S un ensemble de mots et m une multiplicité sur S. Si on a en mémoire l'arbre d'Aho-Corasick généralisé de S, on peut construire le graphe glouton généralisé $GSG(S, m)$ en temps linéaire en $|S|$.

Remarque 8.11 Pour un ensemble de mots S et une multiplicité m de S, la complexité de la construction de $GSG(S, m)$ ne dépend pas de $Set(S, m)$. De plus, comme m est une multiplicité de S, la taille de m est linéaire en la taille de S.

8.2.2 Problème lié au graphe glouton généralisé

Dans la sous-section précédente, on a présenté le graphe glouton généralisé comme une généralisation du graphe glouton. On a montré dans la section 7.3, qu'en parcourant le graphe glouton d'un ensemble de mots P, on construit les solutions de l'algorithme glouton pour SCCS. Dans cette sous-section, on va mettre en évidence le problème qui, comme SCCS est relié au graphe glouton, est relié au graphe glouton généralisé : le problème Multi-Restreint-SMCS. Ce problème va correspondre à une variante de SMCS où on a en plus une multiplicité sur les mots ainsi qu'un ensemble de chevauchements autorisés. On montrera dans les sous-sections 8.2.3 et 8.2.4 que le problème Multi-Restreint-SMCS est une généralisation des problèmes Multi-SCCS et Contraint-SMCS.

La notion de chevauchements autorisés du problème Multi-Restreint-SMCS va modifier la façon de voir le problème Contraint-SMCS. En effet, comme on l'a vu dans la section 5.3, les entrées du problème Contraint-SMCS sont un ensemble de mots P et un ensemble de chevauchements interdits F. Or, au lieu de prendre ces deux ensembles de mots P et F, le problème Multi-Restreint-SMCS va prendre un ensemble S qui pourra être vu comme un sous-ensemble de $P \cup Ov(P)$ pour un ensemble de mots P. On parle ici de généralisation car dans le problème Contraint-SMCS, si on interdit un mot comme chevauchement, on interdit aussi l'ensemble de ces sous-chaînes. Dans le problème Multi-Restreint-SMCS que l'on va examiner ici, si on interdit un mot comme chevauchement en ne le faisant pas apparaître dans les chevauchements autorisés, on pourra quand même autoriser une de ces sous-chaînes.

Soient S un ensemble de mots et m une application de S vers $\mathbb{N}$. En reprenant les notations que l'on a vu dans la section 5.2, on a que $Reduc(S, m)$ est l'ensemble des éléments v de S tels que $m(v) > 0$. On a donc que $Reduc(S, m)$ est un sous-ensemble de $GSG(S, m)$.

Soient u et v deux mots de $Reduc(S, m)$ et donc de S, on peut se demander s'il existe un chemin de u vers v dans l'arbre d'Aho-Corasick généralisé de S.

Dans le graphe hiérarchique des chevauchements de P, il existe forcément un chemin entre deux éléments x et y de P car le nœud $ov(x,y)$ correspondant au chevauchement de ces deux mots est dans le graphe hiérarchique des chevauchements et d'après la proposition 7.13, il existe forcément un chemin qui va de x vers y en passant par $ov(x,y)$. Dans le graphe $GAC(S)$, l'existence du chemin entre deux nœuds u et v de S dépend de la présence d'un chevauchement de u vers v dans S.

Pour définir l'existence d'un chemin entre deux mots de $Reduc(S,m)$, on va définir des permutations correspondant aux parcours d'un graphe glouton généralisé et les mots qui sont issus de ces parcours.

Définition 8.12 Soient S un ensemble de mots et m une application de S vers $\mathbb{N}$.

1. (Ensemble chevauchement)
   Soient x et y deux éléments de $Set(S,m)$. On pose
   $$Chevauchement(x,y) = \big(Suffixe(\{x[1]\}) \setminus \{x[1]\}\big) \cap \big(Prefixe(\{y[1]\}) \setminus \{y[1]\}\big)$$
   où si $x=(a,i)$ et $y=(b,j)$, on a que $x[1]=a$ et $y[1]=b$.
2. (Ensemble Interdit)
   Soit σ une permutation de $Set(S,m)$. On définit
   $$Interdit(S,m,\sigma) = \{x \in Set(S,m) \mid Chevauchement(x,\sigma(x)) \cap S = \emptyset\}.$$
3. (MI-permutation)
   Une *MI-permutation* σ de (S,m) est une permutation de $Set(S,m)$ telle que
   $$|Interdit(S,m,\sigma)| \leq 1.$$
4. (MI-path)
   Soient u et v deux mots de (S,m). Le *MI-path* de u vers v est un ensemble de deux parcours de $GAC(S)$ tels que
   $$MI{-}Path(u,v) = \{\big((u_1,u_2),(u_2,u_3),\dots,(u_{k-1},u_k)\big),\big((v_1,v_2),(v_2,v_3),\dots,(v_{l-1},v_l)\big)\}$$
   où $u_1=u$, u_i est le plus grand suffixe de u_{i-1} dans S, $\big(Suffixe(u_k)\setminus\{u_k\}\big)\cap S=\emptyset$, $v_1=v$, v_i est le plus grand préfixe de v_{i-1} dans S et $\big(Prefixe(v_l)\setminus\{v_l\}\big)\cap S=\emptyset$.
5. (MI-graphe)
   Soit σ_c une MI-permutation de (S,m). Le *MI-graphe* de σ_c sur $GAC(S)$ est le sous-graphe de $GAC(S)$ tel que
   $$MI-Graphe(S,m,\sigma) = \bigcup_{x\in(S,m)} MI-Path(x[1],\sigma(x)[1]).$$
6. (Mi-mixte)
   Soit σ_c une MI-permutation circulaire de (S,m). Le *MI-mixte issu* de σ_c est le mot linéaire ou circulaire tel que
   $$MI-mixte(S,m,\sigma_c) = \begin{cases} \langle pr(x[1],\sigma_c(x)[1])\dots pr(\sigma_c^{|Set(S,m)|-1}(x)[1],x[1])\rangle \\ \qquad \text{si } |Interdit(S,m,\sigma)|=0 \text{ avec } x\in(S,m), \\ pr(\sigma_c(x)[1],\sigma_c^2(x)[1])\dots pr(\sigma_c^{|Set(S,m)|-1}(x)[1],x[1])x[1] \\ \qquad \text{si } Interdit(S,m,\sigma)=\{x\}. \end{cases}$$

7. (MI-MC)
Soit σ une MI-permutation de (S, m). La *MI-couverture mixte de mots issue* de σ est définie telle que

$$MI - MC(S, m, \sigma) = \{MI - mixte(S_1, m_1, \sigma_1) \dots MI - mixte(S_k, m_k, \sigma_k)\}$$

où l'ensemble $Part_\sigma = \{(S_1, m_1), \dots, (S_k, m_k)\}$ est tel que pour tout $i \in \{1, \dots, k\}$, (S_i, m_i) est l'élément de $Part_\sigma$ correspondant à σ_i.

Remarque 8.13 Pour un ensemble de mots S et une pondération m de S, on a que

$$Set(S, reduc_m) = Set(Reduc(S, reduc_m), reduc_m).$$

On a maintenant toutes les notations pour définir de manière formelle le problème Multi-Restreint-SMCS.

Problèmme 8.14 Multi-Restreint-SMCS Soient S un ensemble de mots et m une application de S vers $\mathbb{N}$ qui donnera une multiplicité pour chaque mot de S. On cherche une MI-couverture mixte de mots M telle qu'il existe une MI-permutation τ de $Set(S, reduc_m)$ telle que $M = MI - MC(Reduc(S, reduc_m), reduc_m, \tau)$, que $Overlap(M) \setminus S = \emptyset$ et qui soit de longueur minimale.

Remarque 8.15 Quand on parle ici du problème Multi-Restreint-SMCS, ce problème est « multi » car on a en entrée un ensemble de mots S et une multiplicité sur les mots de S et ce problème est « restreint » car on ne garde que les solutions qui ont leurs chevauchements dans S. Au lieu de donner un ensemble de mots interdits comme on le fait dans le problème Contraint-SMCS, on donne directement l'ensemble des mots que l'on veut avoir dans la couverture mixte de mots finale à l'aide de la multiplicité ainsi que les chevauchements que l'on peut prendre pour créer cette couverture mixte de mots.

Comme on l'a fait pour expliquer la proposition 2.37, à partir d'une couverture mixte de mots M qui est solution de Multi-Restreint-SMCS d'un ensemble de mots S et d'une multiplicité m de S, on peut construire une MI-permutation τ qui pour un mot de $Set(S, reduc_m)$ donnera le prochain mot de $Set(S, reduc_m)$ dans M. On a alors qu'en reconstruisant M à l'aide de τ, on obtient que

$$|M| \geq MI - MC(Reduc(S, reduc_m), reduc_m, \tau).$$

On a alors que l'ensemble des solutions optimales de Multi-Restreint-SMCS sont des MI-couvertures mixtes de mots issues de MI-permutations. On peut alors définir l'algorithme glouton pour Multi-Restreint-SMCS comme étant l'algorithme glouton de Max-HDCC sur une variante du graphe des chevauchements : le MI-graphe des chevauchements.

Définition 8.16 ***MI-Graphe des chevauchements*** Le *MI-graphe des chevauchements* de (S, m) est le graphe orienté pondéré $G = (V, A, w)$ où

$$\begin{aligned} V &:= Set(S, reduc_m) \\ A &:= \{(u, v) \in V \times V \mid Chevauchement(u, v) \cap S \neq \emptyset\} \end{aligned}$$

et

$$\begin{array}{rccc} w: & A & \to & \mathbb{N} \\ & (u,v) & \mapsto & max(\{|w| \mid w \in Chevauchement(u,v)\}) \end{array}$$

On a alors de manière naturelle que les solutions de l'algorithme glouton pour le problème Multi-Restreint-SMCS sont des MI-couvertures mixtes de mots issues de MI-permutations.

Remarque 8.17 En utilisant le système héréditaire défini pour l'algorithme glouton pour Max-HDCC, on peut montrer que l'algorithme glouton pour Multi-Restreint-SMCS$_c$ a une ratio d'approximation optimale supérieur ou égal à $\frac{1}{2}$.

On peut alors généraliser la proposition 7.27. En effet, pour un ensemble de mots S, une permutation m de S et pour deux solutions M_1 et M_2 de l'algorithme glouton sur Multi-Restreint-SMCS, il existe deux MI-permutation τ_1 et τ_2 de $Set(S, reduc_m)$ telles que $M_1 = MI - MC(S, reduc_m, \tau_1)$ et $M_2 = MI - MC(S, reduc_m, \tau_2)$ et on a alors que les MI-graphes $MI - Graphe(S, reduc_m, \tau_1)$ et $MI - Graphe(S, reduc_m, \tau_2)$ sont égaux. On peut même aller un peu plus loin, en montrant que $MI - Graphe(S, reduc_m, \tau_1)$ et $MI - Graphe(S, reduc_m, \tau_2)$ sont égaux au graphe $GSG(S, reduc_m)$. En utilisant le même type de preuve que la proposition 7.31, on peut montrer que le graphe glouton généralisé d'un ensemble de mots S et d'une pondération m sur S permet de générer les solutions de l'algorithme glouton sur Multi-Restreint-SMCS. On obtient alors le théorème suivant :

Théorème 8.18 Soient S un ensemble de mots et m une pondération sur S. Si on a en mémoire l'arbre d'Aho-Corasick généralisé de S, on peut trouver une solution de l'algorithme glouton sur Multi-Restreint-SMCS pour (S, m) en temps linéaire en $|S|$.

8.2.3 Application du graphe GSG au problème Multi-SCCS

Pour le problème Multi-SCCS, on a en entrée un ensemble de mots P et une application m de P dans $\mathbb{N}$. Dans la sous-section précédente, on a défini le problème Multi-Restreint-SMCS qui est une généralisation du problème Multi-SCCS. En effet, chercher une solution optimale du problème Multi-SCCS pour (P, m) revient à chercher une solution optimale du problème Multi-Restreint-SMCS pour $(P \cup Ov(P), m')$ où m' est la pondération sur $P \cup Ov(P)$ telle que pour tout $x \in P$, $m'(x) = m(x)$ et pour tout $x \in \big(P \cup Ov(P)\big) \setminus P$, $m'(x) = 0$.

En remarquant que l'arbre d'Aho-Corasick généralisé de $P \cup Ov(P)$ est le graphe hiérarchique des chevauchements de P et en appliquant les théorèmes 7.25 et 8.18, on peut trouver une solution de l'algorithme glouton pour Multi-SCCS en temps linéaire en $||P||$. De plus, le corollaire 5.30 nous donne que l'algorithme glouton sur le problème Multi-SCCS est optimal. On a alors le théorème suivant :

Théorème 8.19 Il existe un algorithme qui résout exactement le problème Multi-SCCS en temps linéaire en $||P||$.

8.2.4 Application du graphe GSG au problème Contraint-SCCS

Comme pour le problème Multi-SCCS, le problème Contraint-SMCS est un cas particulier du problème Multi-Restreint-SMCS. En effet, chercher une solution optimale

du problème Contraint-SMCS pour (P, F) revient à chercher une solution optimale du problème Multi-Restreint-SMCS pour $(P \cup Ov^*(P, F), m)$ où m est la pondération sur $P \cup Ov^*(P, F)$ telle que pour tout $x \in P$, $m(x) = 1$ et pour tout $x \in P \cup Ov^*(P, F) \setminus P$, $m(x) = 0$.

En appliquant les théorèmes 7.25 et 8.18, on peut trouver une solution de l'algorithme glouton pour Contraint-SMCS en temps linéaire en $||P||$.

D'après le théorème 5.36, l'algorithme glouton pour Contraint-SMCS$_c$ a une ratio d'approximation optimal supérieur ou égal à $\frac{1}{2}$.

Théorème 8.20 Il existe un algorithme qui donne un ratio d'approximation supérieur ou égal à $\frac{1}{2}$ pour le problème Contraint-SMCS$_c$ en temps linéaire en $||P||$.

Conclusion de la section Dans cette section, on a mis en évidence le problème Multi-Restreint-SMCS et on a défini le graphe glouton généralisé qui permet de générer les solutions de l'algorithme glouton pour ce problème. En appliquant ce problème, on a pu trouver un algorithme linéaire en la norme de l'instance pour construire les solutions des algorithmes gloutons des problèmes Multi-SCCS et Contraint-SMCS. Enfin, comme on a vu dans les sections 5.2 et 5.3, une borne du ratio d'approximation optimal de l'algorithme glouton pour les variantes compression de ces problèmes, on a pu mettre en évidence que le problème Multi-SCCS est dans **P**.

Conclusion du chapitre Dans ce chapitre, on a montré que l'on pouvait généraliser facilement la construction linéaire des solutions optimales du problème SCCS aux problèmes SDCCS et Multi-SCCS. On a donc montré que ces deux problèmes étaient aussi dans **P**. En outre, on a montré qu'on pouvait construire de manière linéaire les solutions de l'algorithme glouton pour le problème Contraint-SMCS.

Chapitre 9

Construction du graphe de De Bruijn

On va voir dans ce chapitre, comment construire les graphes de De Bruijn (classiques et contractés) (voir section 7.1).

Pour commencer, on va donner une nouvelle définition du graphe de De Bruijn en utilisant les mots. Ceci va nous permettre après d'appliquer cette définition à l'arbre des suffixes et à l'arbre des suffixes tronqué (que l'on a présenté au chapitre 6). La construction du graphe de De Bruijn à l'aide de l'arbre des suffixes a été publiée dans (X) et (II) et à l'aide de l'arbre des suffixes tronqué a été publiée dans (VII) et (II).

Enfin, on montrera comment utiliser le graphe glouton pour construire les deux graphes de De Bruijn (dBG_k^+ et dBG_k^-), ce qui nous permettra de mettre en valeur un lien fort entre ces deux graphes. Nous avons publié ce lien ainsi qu'une étude de l'algorithme d'IDBA et une comparaison avec une solution gloutonne dans (IV).

On pourra retrouver dans (III) une étude de la complexité de chercher un chemin hamiltonien sur le graphe de De Bruijn.

Pour simplifier les choses, on ne va parler dans les trois premières sections que du graphe de De Bruijn dBG_k^+. En effet les résultats de dBG_k^+ sont facilement généralisés au cas du graphe dBG_k^-.

Sommaire

9.1 Caractérisation constructive

On va commencer par définir quelques notations. Soient $P := \{s_1, \ldots, s_n\}$ un ensemble de mots et w un mot de $Fact(P)$.

— $Occ(w, P) = \bigcup_{i=1}^{n} Occ(w, s_i)$ est l'ensemble des paires (i, j) telles que $j \in Occ(w, s_i)$. L'ensemble $Occ(w, P)$ est appelé le support de w dans P.

— $RC(w)$ (resp. $LC(w)$) est l'ensemble des extensions à droite (resp. à gauche) de w dans P, *i.e.* l'ensemble des mots w' tels que $ww' \in Fact(P)$ (resp. $w'w \in Fact(P)$).

— $\lceil w \rceil$ est le mot ww' où w' est le plus long mot de $RC(w)$ tel que $Occ(w, P) = Occ(ww', P)$; en d'autres mots, tels que w et ww' ont exactement le même support.

— $\lfloor w \rfloor$ est le mot w' où w' est le plus long préfixe de w tel que $Occ(w, P) \neq Occ(w', P)$.

— $d(w) := |\lceil w \rceil| - |w|$.

Remarque 9.1 Soit w un mot de $Fact(P)$. Les ensembles $RC(w)$ et $LC(w)$ sont respectivement le « Right Context » et le « Left Context » de w.

	1	2	3	4	5	6	7
s_1	b	a	c	b	a	b	
s_2	c	b	a	b	c	a	a
s_3	b	c	a	a	c	b	
s_4	c	b	a	a	c		
s_5	b	b	a	c	b	a	a

FIGURE 9.1 – Exemple pour $P = \{bacbab, cbabcaa, bcaacb, cbaac, bbacbaa\}$: $Occ(ba, P) = \{(1,1), (1,4), (2,2), (4,2), (5,2), (5,5)\}$,
$RC(ba) = \{\varepsilon, c, cb, cba, cbab, b, bc, bca, bcaa, a, ac, cbaa\}$ et $LC(ba) = \{\varepsilon, c, ac, bac, b, bbac\}$.

En d'autres termes, $\lceil w \rceil$ est la plus longue extension de w qui a le même support que w dans P et $\lfloor w \rfloor$ est la plus courte réduction de w qui a un support différent de w dans P. Soient deux mots w_1 et w_2 tels que $Occ(w_1, P) \neq \emptyset$, on a alors les implications suivantes :

— $Occ(w_1, P) \subset Occ(w_2, P)$ implique que w_2 est un préfixe strict de w_1.

— w_2 est un préfixe de w_1 implique que $Occ(w_1, P) \subseteq Occ(w_2, P)$.

— $Occ(w_1, P) = Occ(w_2, P)$ implique que w_2 est un préfixe de w_1 ou w_1 est un préfixe de w_2.

On peut alors retrouver le graphe de De Bruijn avec les définitions précédentes :

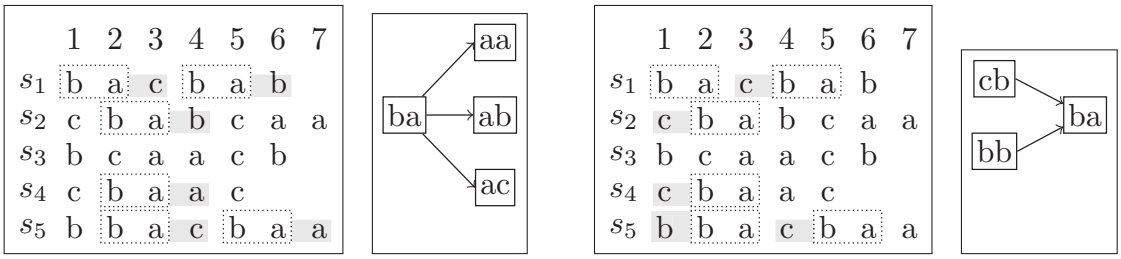
Proposition 9.2 Soit (V_k, E_k^+) le graphe de De Bruijn dBG_k^+ de P. On a alors
— $E_k^+ = \{(u, v) \in V_k \times V_k \mid u[|u| - k + 2, |u|] = v[1, k-1] \text{ et } v(k) \in RC(u)\}$.
— $E_k^+ = \{(u, v) \in V_k \times V_k \mid u[|u| - k + 2, |u|] = v[1, k-1] \text{ et } u(1) \in LC(v)\}$.

On rappelle que $v(1)$ est le premier caractère de v et $v(|v|)$ est le dernier caractère de v. De plus on a que $u[|u|-k, |u|]$ est le suffixe de taille $k-1$ de u et $v[1, k-1]$ est le préfixe de taille $k-1$ de v.

Soit k un entier positif. On définit les trois sous-ensembles de $Fact(P)$ suivants :

— $InitExact_k := \{w \in Fact(P) \mid |w| = k \text{ et } d(w) = 0\}$,

— $Init_k := \{w \in Fact(P) \mid |w| \geq k \text{ et } d(w[1, k]) = |w| - k\}$,



(a) (b) (c) (d)

FIGURE 9.2 – Exemple d'arcs de dBG_k^+. (a) correspond aux extensions à droite de ba qui correspondent à des lettres, et (b) aux successeurs du nœud ba dans dBG_2^+ ; un pour chaque élément de $\{x \in RC(w) \mid |x| = 1\}$. (c) correspond aux extensions à gauche de ba qui correspondent à des lettres, et (d) aux prédécesseurs du nœud ba dans dBG_2^+.

— $SubInit_k = InitExact_{k-1}$.

Un mot de $InitExact_k$ est soit le suffixe d'un certain s_i de P ou a au moins deux extensions à droite. On a, de plus, que le préfixe de longueur k d'un mot de $Init_k \setminus InitExact_k$ n'a qu'une seule extension à droite.

On peut alors trouver un lien entre les nœuds de $Init_k$ et de $InitExact_k$.

Proposition 9.3 $InitExact_k = Init_k \cap \{w \in Fact(P) \mid |w| = k\}$

Preuve Soit $w \in InitExact_k$. On a alors que $w[1,k] = w$ et $|w| - k = 0$. Cela veut donc dire que $d(w[1,k]) = |w| - k$ et donc $w \in Init_k$. ■

Pour w un élément de $Init_k$, $w[1,k]$ est un k-mer de P. Soit deux mots différents w_1 et w_2 de $Init_k$, $w_1[1,k]$ et $w_2[1,k]$ sont deux k-mers différents de P. De plus, pour chaque k-mer w' de P, il existe un mot $w \in Init_k$ tel que $w' = w[1,k]$. De cette propriété, on a alors la proposition suivante :

Proposition 9.4 Soit f l'application de $Init_k$ vers $Fact_k(P)$ telle que pour $w \in Init_k$, $f(w) := w[1,k]$. On a alors que f est une bijection entre $Init_k$ et l'ensemble des k-mers de P.

On a alors que l'on peut assimiler chaque nœud de dBG_k^+ à un élément de $Init_k$. Maintenant, on va définir un ensemble d'arcs entre les mots de $Init_k$ qui correspondent aux arcs du dBG_k^+. On a alors besoin de la proposition suivante :

Proposition 9.5 Soient $w \in InitExact_k$ et $a \in RC(w) \cap \Sigma$. Il existe alors un unique $w' \in Init_k$ tel que $w[|w| - k + 2, |w|].a$ est un préfixe de w'.

Preuve Soient w un mot de $InitExact_k$ et a une lettre de $RC(w) \cap \Sigma$. Par définition de l'extension à droite, $w[|w|-k+2, |w|].a \in Fact(P)$. Comme $|w[|w|-k+2, |w|].a| = k$, il existe w' tel que $w[|w| - k + 2, |w|].a$ est un préfixe de w' et $|w[|w| - k + 2, |w|].a| + d(w[|w| - k + 2, |w|].a) = |w'|$. Par définition de $Init_k$, $w' \in Init_k$. ■

On utilise donc l'ensemble $Init_k$ pour représenter les nœuds du graphe de De Bruijn dBG_k^+. On va maintenant construire un ensemble des arcs isomorphes à E_k^+. Soit w un mot de $Init_k$. On note $Succ_k(w)$ l'ensemble des successeurs de $w[1,k]$ dans E_k^+ :

$$Succ_k(w) := \{x \in Init_k \mid (w[1,k], x[1,k]) \in E_k^+\}$$

On sait que pour chaque lettre a de $RC(w) \cap \Sigma$, il existe un arc de $w[1,k]$ vers $w[2,k].a$ dans dBG_k^+. On considère deux cas dépendant de la longueur de w :

Cas 1 : $|w| = k$,
D'après la proposition 9.3, $w \in InitExact_k$ et donc $w[2,k] \in SubInit_k$. Par conséquent, les arcs sortants de w dans dBG_k^+ sont des arcs de w vers w' satisfaisant la condition de la proposition 9.5. On a alors

$$Succ_k(w) = \bigcup_{a \in RC(w) \cap \Sigma} \lceil w[2,k].a \rceil$$

Cas 2 : $|w| > k$,
Comme w est plus grand que k, il contient le k-mer suivant ; en effet on a que $w[2,k].a = w[2,k+1]$ et donc il existe un unique arc sortant de w : qui va de w vers $\lceil w[2,k+1] \rceil$, car $\lceil w[2,k+1] \rceil \in Init_k$ et alors

$$Succ_k(w) = \{\lceil w[2,k+1] \rceil\}$$

On peut construire maintenant intégralement dBG_k^+ ou plus exactement un graphe isomorphe à dBG_k^+.

Théorème 9.6 Avec les ensembles $Init_k$, $InitExact_k$ et $SubInit_k$, on peut construire un graphe isomorphe à dBG_k^+ en temps linéaire en la taille de ces ensembles.

Par simplicité, par la suite, on va confondre le graphe que l'on a construit avec dBG_k^+.

On peut alors faire de même avec le graphe de De Bruijn contracté. Pour cela, on va introduire la notion d'extensibilité d'un mot de $Fact(P)$.

Soit $w \in Fact(P)$. On dit que :

— w est *extensible à droite* dans P, si et seulement si $|\{x \in RC(w) \mid |x| = 1\}| = 1$,

— w est *extensible à gauche* dans P, si et seulement si $|\{x \in LC(w) \mid |x| = 1\}| = 1$.

Si w est extensible à droite, on pose $next(w)$ tel que $\{next(w)\} = \{x \in RC(w) \mid |x| = 1\}$.

Remarque 9.7 La notion d'extensibilité d'un mot est différente de celle d'extensibilité d'un système héréditaire. L'extensibilité d'un mot indique si les arcs du dBG_k^+ vont pouvoir être contractés dans $CdBG_k^+$.

Soit w un mot de Σ^* tel que $|w| \geq k$. Le mot w est un *mot unique* de P relativement à k si et seulement si en décomposant $w = w_1 \oplus \ldots \oplus w_{|w|-k+1}$ en k-mer, on a pour tout $j \in \{1, \ldots, |w| - k\}$, w_j est extensible à droite dans P (et donc $w_j \in Fact(P)$) et w_{j+1} est extensible à gauche dans P (et donc $w_{j+1} \in Fact(P)$).

On a alors la proposition suivante :

Proposition 9.8 Soit $(V_{k,c}, E^+_{k,c})$ le graphe de De Bruijn contracté $CdBG^+_k$ de P, on a alors

- $V_{k,c} = \{w \in \Sigma^* \mid$ w un mot unique maximal par sous-chaîne de P relativement à $k\}$,
- $E^+_{k,c} = \{(u,v) \in V^2_{k,c} \mid u[|u|-k+2,|u|] = v[1,k-1]$ et $v(k) \in RC(u[|u|-k+1,|u|])\}$,
- $E^+_{k,c} = \{(u,v) \in V^2_{k,c} \mid u[|u|-k+2,|u|] = v[1,k-1]$ et $u(|u|-k+1) \in LC(v[1,k])\}$.

Pour prouver cette proposition, on a besoin des propositions 9.9 et 9.10. On a alors que l'on contracte un arc (u,v) dans dBG^+_k pour construire $CdBG^+_k$ si et seulement si u est extensible à droite et v est extensible à gauche.

On remarque qu'à la différence de V_k, l'ensemble de nœuds du graphe de De Bruijn, $V_{k,c}$ n'est pas forcément inclus dans $Fact(P)$. On a alors pour w un mot unique maximal par sous-chaîne de P relativement à k :

- $w[|w|-k+1,|w|]$ le suffixe de longueur k de w n'est pas extensible à droite OU $RC(w[|w|-k+1,|w|]) \cap \Sigma = \{a\}$ et $w[|w|-k+2,|w|].a$ n'est pas extensible à gauche.
- $w[1,k]$ le préfixe de longueur k de w n'est pas extensible à gauche OU $LC(w[1,k]) \cap \Sigma = \{a\}$ et $a.w[1,k-1]$ n'est pas extensible à droite.

Avec ces arguments, on obtient les deux propositions suivantes :

Proposition 9.9 Soit $u \in V^+_{k,c}$:

- $(u[|u|-k+1,|u|], v[1,k]) \in E^+_k$,
- il existe $w \in V_k$ tel que $(w, v[1,k]) \in E^+_k \setminus \{(u[|u|-k+1,|u|], v[1,k])\}$ ou $(u[|u|-k+1,|u|], w) \in E^+_k \setminus \{(u[|u|-k+1,|u|], v[1,k])\}$.

Proposition 9.10 Soit $u \in V^+_k$. Si u est extensible à droite et v est extensible à gauche, alors il existe $w \in V_{k,c}$ tel que $u.v(k)$ est une sous-chaîne de w. Sinon il existe $(u',v') \in E^+_{k,c}$ tel que $u = u'[|u'|-k+1,|u'|]$ et $v = v'[1,k]$.

Pour faire la même chose que le théorème 9.6 pour le graphe $CdBG^+_k$, on commencera par expliquer l'algorithme de construction et, dans un second temps, on caractérisera le concept d'extensibilité à droite et à gauche en termes de propriétés sur les mots.

Notre algorithme pour construire $CdBG^+_k$. On présente un algorithme générique pour construire de manière incrémentale $CdBG^+_k$. Cette explication utilise les mots et ne dépend d'aucune structure d'indexation. Dans les sections suivantes, on utilisera cet algorithme générique sur l'arbre des suffixes généralisé et l'arbre des suffixes généralisé tronqué.

L'algorithme principal (Algorithme 9.2) explore le dBG^+_k pour trouver les nœuds qui restent dans $CdBG^+_k$ et crée un ensemble d'arcs qui représentent les chemins non-branchant dans le dBG^+_k qui seront correctement contractés. Le point clé est de trouver tous les nœuds commençant un chemin simple et explorer ce chemin simple ; cette exploration est donnée par l'algorithme 9.1.

Algorithme 9.1 : BuildAuxCDBG(V, E, v_f, v_c).

Input : Un graphe (V, E) partiellement contracté de $CdBG_k^+$, deux nœuds v_f et v_c. v_f est le nœud initial, et v_c le nœud courant.
Output : Le graphe contracté (V', E') à jour, qui contient maintenant chaque chemin commençant en v_c.

```
1  begin
2      u := v_c;
3      On marque u;
4      // On cherche le nœud finissant la chaîne passant par v_c
5      while u est extensible à droite et next(u) est extensible à gauche do
6          if v_f = next(u) then
7              On remplace (v_f, i) par (v_c, i) pour tout (v_f, i) ∈ E;
8              return (V \ {v_f}, E)
9          u := next(u);
10         On marque u;
11     //On explore le chemin commençant par le successeur de u
12     for w ∈ Succ_k(u) do
13         if w ∈ V then
14             (V, E) := (V, E ∪ {(v_c, w)})
15         else
16             (V, E) := BuildAuxCDBG(V ∪ {w}, E ∪ {(v_c, w)}, v_f, w);
17             // On explore depuis w
18     return (V, E)
```

Algorithme 9.2 : BuildCDBG(P).

Input : A set of word S.
Output : $CdBG_k^+$ of S.

```
1  begin
2      (V, E) = (∅, ∅)
3      for v ∈ Init_k do
4          if il n'existe pas de w tel que v ∈ Succ_k(w) then
5              (V, E) := (V, E) ⋃ BuildAuxCDBG(V ∪ {v}, E, v, v)
6      // On explore CdBG_k^+ pour tout nœud que l'on a pas encore visité
7      for v_c un nœud non marqué de Init_k do
8          (V, E) := (V, E) ⋃ BuildAuxCDBG(V ∪ {v_c}, E, v_c, v_c)
9      return (V, E)
```

Une explication un peu plus détaillée. Pour commencer, on va noter que pour construire dBG_k^+, il suffit de connaître l'ensemble $Succ_k(.)$ pour chaque nœud. L'algorithme 9.2 simule une traversée de dBG_k^+ sans le construire et garde seulement les nœuds qui sont des mots uniques maximaux par sous-chaîne de P relativement à k. Pour ce genre de mot, disons m, on choisit de le représenter par un nœud v tel que $w[1,k]$ est un préfixe de m. Dans dBG_k^+, m est représenté par un chemin non-branchant et v est le premier nœud de ce chemin. En utilisant un algorithme de traversée, pour un nœud courant commençant v_c dans $Init_k$, on traverse le chemin non-branchant jusqu'à ce qu'on arrive à un nœud u qui a plusieurs successeurs ou tel que son unique successeur n'est pas extensible à gauche (*i.e.* qui a plusieurs prédécesseurs). En d'autres termes, jusqu'à ce qu'on trouve u tel que u n'est pas extensible à droite ou que $next(u)$ n'est pas extensible à gauche. Dans dBG_k^+, il existe un chemin non-branchant entre v_c et u et donc on doit créer un nœud dans $CdBG_k^+$ qui correspond à ce chemin. Pour contracter ce chemin, on choisit de garder v_c et pour tout successeur w de u, on ajoute un arc en u et w, car ces arcs ne peuvent pas être contractés. Rien ne dit que w commence nécessairement un chemin (ayant au moins un seul nœud) ; si w n'est pas déjà dans $CdBG_k^+$, on lance une nouvelle exploration commençant en w. En effet $w(0,k-1)$ est un préfixe d'un nœud de $CdBG_k^+$ et alors w est le représentant de ce chemin. Maintenant si w est déjà un nœud de $CdBG_k^+$, le cas est plus délicat. Si v_f sauvegarde le premier v_c appelé par la procédure, v_f n'est pas forcément au début d'un chemin non-branchant, mais peut être en plein milieu. Deux cas peuvent se produire. Si v_f est considéré à l'intérieur de la boucle « While », on a alors que v_f n'est pas le début d'un chemin non-branchant : on doit alors mettre à jour V en échangeant v_f par v_c et arrêter l'exploration. Dans l'autre cas, v_f est traversé pendant la boucle « For » (comme une valeur de w), c'est alors un successeur de u et le commencement d'un nouveau chemin non-branchant : on ajoute alors juste un arc entre v_c et w. Enfin, si w est déjà dans V mais $w \neq v_f$, on ajoute alors également un arc en v_c et w et on arrête.

La procédure accomplie par l'algorithme 9.1 augmente le graphe partiel de $CdBG_k^+$ restreint aux nœuds visités pendant l'exploration du chemin commençant en v_c. Il suffit maintenant de s'assurer que tous les arcs de dBG_k^+ ont été examinés, ce que l'algorithme 9.2 fait.

Plus précisément, l'algorithme 9.2 commence par visiter les chemins non-branchants commençant sur des nœuds n'ayant aucun prédécesseur (sinon ces nœuds n'auraient pas été visités). Une fois que cela est fait, on doit explorer tous les nœuds qui n'ont pas encore été marqués et continuer jusqu'à ce que tous les nœuds aient été marqués/visités.

De la discussion que l'on vient d'avoir, on obtient le théorème suivant :

Théorème 9.11 Avec les ensembles $Init_k$, $InitExact_k$ et $SubInit_k$, l'algorithme 9.2 construit un graphe isomorphe à $CdBG_k^+$ en temps linéaire en la taille de ces ensembles.

Preuve Pour la preuve, il suffit de montrer que l'on peut calculer en temps constant si u est extensible à droite et si v est extensible à gauche pour tout arc (u,v) dans E_k^+. Ce résultat sera prouvé ci-dessous grâce aux propositions 9.12 et 9.13. ■

Remarquons que l'algorithme 9.2 ne requiert pas de construire dBG_k^+ car l'ensemble des successeurs $Succ_k(u)$ de n'importe quel nœud u peut être calculé en temps constant.

Caractérisation des concepts d'extensibilité au droite et à gauche. Par construction de dBG_k^+, on obtient les deux propriétés suivantes qui vont être utiles pour

prouver la linéarité de la construction de $CdBG_k^+$, c'est-à-dire que pour un arc (u, v) de E_k^+, on peut trouver de manière constante si u est extensible à droite et si v est extensible à gauche.

Proposition 9.12 Soit w un mot de $Init_k$. Alors $w[1, k]$ est extensible à droite si et seulement si $|w| > k$ ou $|\{x \in RC(w) \mid |x| = 1\}| = 1$.

Proposition 9.13 Soit w un mot de $Init_k$ tel que $w[1, k]$ est extensible à droite. Soit a la lettre de l'unique élément de $\{x \in RC(w[1, k]) \mid |x| = 1\}$, on a alors que $w[2, k].a$ est extensible à gauche si et seulement si

$$|Occ(w[1, k], P)| = |Occ(w[2, k].a, P) \setminus \{(i, 1) \mid 1 \leq i \leq n\}|.$$

Preuve ($\Rightarrow$) Soit (i, j) un paire de $Occ(w[1, k], P)$. On a donc que $(i, j + 1) \in Occ(w[2, k], P)$. Comme $Occ(w[2, k], P) = Occ(w[2, k].a, P)$, on a alors que $(i, j + 1) \in Occ(w[2, k].a, P)$.

($\Leftarrow$) Supposons qu'il existe $(i, j) \in Occ(w[2, k], P)$ tel que $j > 0$ et $(i, j - 1) \notin Occ(w[1, k], P)$. On a alors qu'il existe une lettre $b \neq w(1)$ tel que $(i, j - 1) \notin Occ(b.w[2, k], P)$. On a alors que $(b.w[2, k], w[2, k].a)$ est aussi un arc de E_k^+ et on a alors que $w[2, k].a$ n'est pas extensible à gauche. ■

Conclusion de la section Pour résumer, dans cette section on donne une nouvelle formulation du graphe de De Bruijn en terme de mots ainsi qu'un algorithme de construction. Maintenant, supposons que les sous-chaînes des mots soient indexées par une structure de données, par exemple l'arbre des suffixes généralisé. Comment peut-on construire le graphe dBG_k^+ ou directement le graphe $CdBG_k^+$ à partir de cette structure de données ? Pour le réaliser, il suffit de calculer les trois ensembles $Init_k$, $InitExact_k$ et $SubInit_k$ ainsi que les ensembles $Occ(., P)$ et $Succ_k(.)$ pour toutes les sous-chaînes appropriées. Dans les sections suivantes, on va exposer des algorithmes pour calculer les deux graphes de De Bruijn pour deux structures de données importantes.

9.2 En utilisant l'arbre des suffixes

Commençons par montrer la construction des deux graphes de De Bruijn dBG_k^+ et $CdBG_k^+$ en utilisant l'arbre des suffixes généralisé.

9.2.1 De l'arbre des suffixes au dBG_k^+

Soit T l'arbre des suffixes généralisé de l'ensemble de mots P. On sait que pour tout nœud v de T, v est dans $Fact(P)$. Mais comme tous les éléments de $Fact(S)$ ne sont pas nécessairement des nœuds de T, on les caractérise ainsi :

Proposition 9.14 L'ensemble des nœuds de T est exactement l'ensemble des mots w de $Fact(P)$ tel que $d(w) = 0$.

On rappelle la notion de lien suffixe pour un nœud de T (feuilles incluses). On note $ls(v)$ le nœud correspondant au lien suffixe de v, *i.e.* $ls(v) := v[2, |v|]$. Par définition de

l'arbre des suffixes généralisé, pour tout nœud $w \in Fact(P)$, il existe un nœud v de T tel que w est le préfixe de v. Soit v' le nœud de longueur minimale de T tel que w est le préfixe de v', on a alors que $|v'| = |w| + d(w)$ et donc $\lceil w \rceil = v'$.

Proposition 9.15 Soit $w \in Fact(P)$, on a alors que $|\lceil w \rceil| \geq |w| > |p_T(\lceil w \rceil)|$ où $p_T(\lceil w \rceil)$ est le parent de $\lceil w \rceil$ dans T.

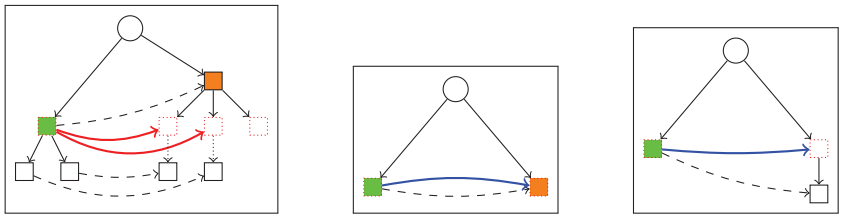
Preuve Comme $p_T(\lceil w \rceil) = \lfloor w \rfloor$, le résultat est immédiat. ■

Soit $\{x_1, \ldots, x_m\}$ l'ensemble des k-mers de P, on a alors que $Init_k = \{\lceil x_1 \rceil, \ldots, \lceil x_m \rceil\}$. En utilisant la proposition 9.15, on a alors que $Init_k = \{v \in V_T \mid |p_T(v)| < k \text{ et } |v| \geq k\}$ et $InitExact_k = \{v \in V_T \mid |v| = k\}$. On voit alors que $InitExact_k$ est un sous-ensemble de $Init_k$, car pour tout $v \in V_T$, $|p_T(v)| < |v|$.

On va reprendre alors les deux cas pour la construction de E_k^+ vue page 184, mais en l'appliquant au cas des arbres. Soit $v \in Init_k$.

Cas 1 : $|v| = k$, (Figure 9.3a)
Comme $v \in InitExact_k$, $ls(v) \in SubInit_k$. De plus chaque enfant u de $ls(v)$ est un élément de $Init_k$. Un arc sortant de v dans dBG_k^+ est alors un arc de v vers l'enfant u de $ls(v)$ tel que la première lettre de l'étiquette entre $ls(v)$ et u est l'élément du contexte droit de v. Comme l'ensemble des premières lettres des étiquettes entre v et les enfants de v est exactement $RC(v) \cap \Sigma$, le nombre d'arcs sortants de v dans dBG_k^+ est le nombre de fils de v. Pour construire les arcs sortants de v dans dBG_k^+, pour chaque fils u' de v, on associe v avec le nœud de $Init_k$ qui se trouve entre la racine et $ls(u')$, *i.e.* $\lceil ls(u')[1, k] \rceil$.

Cas 2 : $|v| > k$, (Figure 9.3b et 9.3c))
On a que $ls(v)$ est un nœud de T. Comme $|v| > k$, $|ls(v)| \geq k$. Il existe alors un élément de $Init_k$ entre la racine et $ls(v)$. On associe alors v avec ce nœud, *i.e.* $\lceil ls(v)[1, k] \rceil$.



(a) (b) (c)

FIGURE 9.3 – Les figures (a), (b) et (c) montrent les cas 1 et 2 rencontrés quand on calcule les arcs de dBG_k^+. Le nœud vert représente le nœud v, et celui en orange $ls(v)$. Les arcs en pointillés correspondent aux arcs entre un nœud et son lien suffixe. Les arcs de dBG_k^+ sont en ligne pleine et colorés en rouge pour le Cas 1 (a) et en bleu pour le cas 2 (b),(c).

On peut alors illustrer ces deux cas par la figure 9.4 :

Cas 1 : Cas où v est 6,6, $ls(v)$ est 7,7, l'unique fils u' de v est 3, et $ls(u')$ est 4, qui est dans $Init_k$.

Cas 2 : Cas où v est 1, $ls(v)$ est 2, et $\lceil sl(v)[1, k] \rceil$ est .

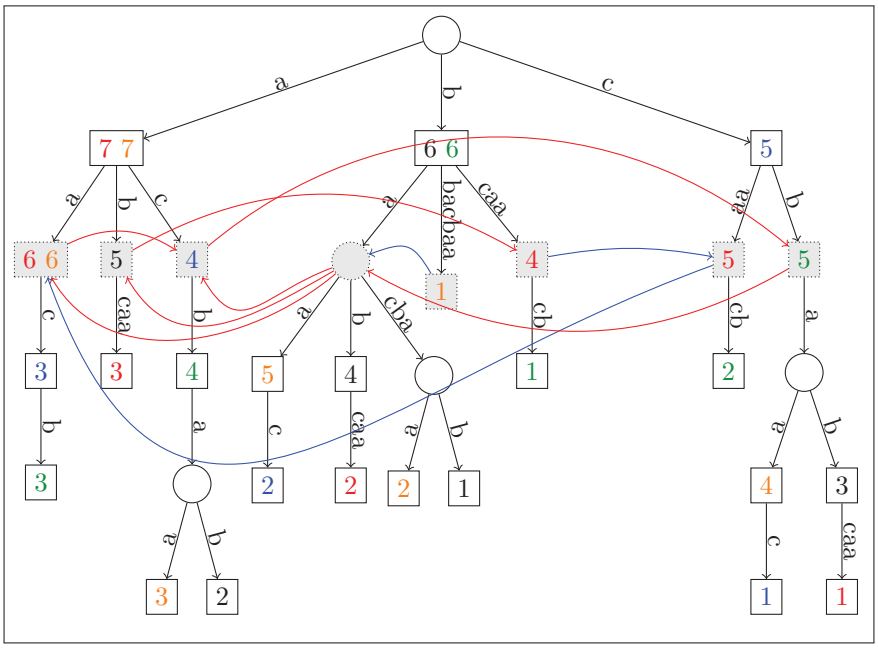


FIGURE 9.4 – Exemple d'arbre des suffixes généralisé sur $P = \{bacbab, bcaacb, cbaac, cbabcaa, bbacbaa\}$ avec le graphe de De Bruijn pour $k = 2$. Les nœuds en gris sont ceux utilisés dans le graphe de De Bruijn. Les arcs pleins courbés sont les arcs du graphe de De Bruijn. Ceux en rouge correspondent au cas 1 et ceux en bleu, au cas 2 dans la construction de E_k^+.

Dans les deux cas, la construction des arcs de E^+ réclame de suivre les liens suffixes de certains nœuds. Le nœud, disons u, pointé par un lien suffixe peut ne pas être initial. Le nœud initial représentant le préfixe de longueur k de u est alors l'unique nœud initial qui est un ancêtre de u. On équipe alors ces nœuds u d'un pointeur $q(u)$ qui pointe l'unique nœud initial du chemin entre lui-même et la racine. En d'autres mots, pour tout $u \notin Init_k$ tel que $|u| > k$, on a que $q(u) := \lceil u[1,k] \rceil$.

L'algorithme pour construire le dBG_k^+ procède comme suit. On commence par effectuer une première descente dans T qui permet de récupérer les nœuds de $Init_k$ et pour tous les nœuds dans les sous-arbres des nœuds initiaux de récupérer le pointeur $q(.)$. Enfin pour construire E^+, on passe en revue $Init_k$ et pour tout nœud v on ajoute $Succ_k(v)$ à E^+ en utilisant la formule donnée avant. Cet algorithme est alors linéaire en temps en la taille de T. De plus, le nombre d'arcs dans E^+ est linéaire en le nombre total d'enfants des nœuds initiaux. Ce qui nous donne le résultat suivant.

Théorème 9.16 Pour en ensemble de mots P, la construction du graphe de De Bruijn d'ordre k, dBG_k^+ est linéaire en temps et en espace en $|T|$, *i.e.* en $||P||$.

9.2.2 De l'arbre des suffixes au $CdBG_k^+$

Dans la section 9.1, on a vu un algorithme permettant de calculer directement $CdBG_k^+$ pour autant qu'on puisse déterminer pour un arc (u, v) si u est extensible à droite et si v est extensible à gauche. On va voir maintenant comment on peut calculer l'extensibilité à droite et à gauche dans le cas de l'arbre des suffixes.

En appliquant la proposition 9.12 dans le cas d'un arbre, pour un élément v de $Init_k$, $w[1,k]$ est extensible à droite si et seulement si $|v| > k$ ou $|Children_T(v)| = 1$. On peut alors vérifier en temps constant qu'un nœud est extensible à droite.

Pour l'extensibilité à gauche d'un unique successeur d'un nœud, on a alors uniquement besoin de la taille du support de certains nœuds (Proposition 9.13). On va montrer comment on peut calculer $|Succ_k(.)|$ sur un arbre, et comment appliquer la proposition 9.13.

Proposition 9.17 Soit v un mot de $Fact(P)$ et $V_T(\lceil v \rceil)$ l'ensemble des nœuds du sous-arbre de T enraciné en $\lceil v \rceil$.

$$Occ(v,P) = \bigcup_{v' \in V_T(\lceil v \rceil)} Position - Suff_P(v').$$

En faisant une traversée de l'arbre, on peut calculer et mémoriser $|Occ(v,P)|$ et $|Occ(v,P) \cap \{(i,1) \mid 1 \leq i \leq n\}|$ pour tout nœud v en temps linéaire en $|T|$.

Soit v un mot de $Init_k$ tel que $v[1,k]$ est extensible à droite.

Cas 1 : $|v| = k$,
on a alors que $v[1,k] = v$ et $|Children_T(v)| = 1$. En posant u l'unique fils de v dans T, on a alors $|u| > k$, $\{x \in RC(v) \mid |x| = 1\} = \{u(k+1)\}$, et $v[2,k].u(k+1) = sl(u)[1,k]$. On a alors

$$|Occ(v,P)| = |Occ(sl(u)[1,k],P) \setminus \{(i,1) \mid 1 \leq i \leq n\}|$$

et par la proposition 9.13, $sl(u)[1,k]$ est extensible à gauche.

Cas 2 : $|v| > k$,
on a alors que $\{x \in RC(v[2,k]) \mid |x| = 1\} = \{v(k+1)\}$ et $v[2,k].v(k+1) = v[2,k+1] = sl(v)[1,k]$. Par la proposition 9.13, $sl(v)[1,k]$ est extensible à gauche si et seulement si

$$|Occ(v(0,k-1),P)| = |Occ(sl(v)[1,k],P) \setminus \{(i,1) \mid 1 \leq i \leq n\}|$$

Comme $|Occ(v,P)| = |Occ(\lceil v \rceil,P)|$ et $|Occ(v,P) \setminus \{(i,1) \mid 1 \leq i \leq n\}| = |Occ(v,P)| - |Occ(v,P) \cap \{(i,1) \mid 1 \leq i \leq n\}|$, on peut déterminer en temps constant si l'unique successeur d'un nœud extensible à droite est extensible à gauche. Pour conclure, comme pour chaque nœud initial v, on peut calculer en $O(1)$ l'ensemble de ses successeurs $Succ_k(v)$, son extensibilité à droite et l'extensibilité à gauche de son unique successeur, on peut appliquer l'algorithme 9.2 pour construire $CdBG_k^+$ et on obtient une complexité linéaire en la taille de dBG_k^+, puisque chaque successeur n'est utilisé qu'une seule fois. On obtient alors le théorème 9.18.

Théorème 9.18 Pour un ensemble de mots P, construire le graphe de De Bruijn contracté d'ordre k, $CdBG_k^+$, prend un espace et un temps linéaire en $|T|$, *i.e.*, en $||P||$.

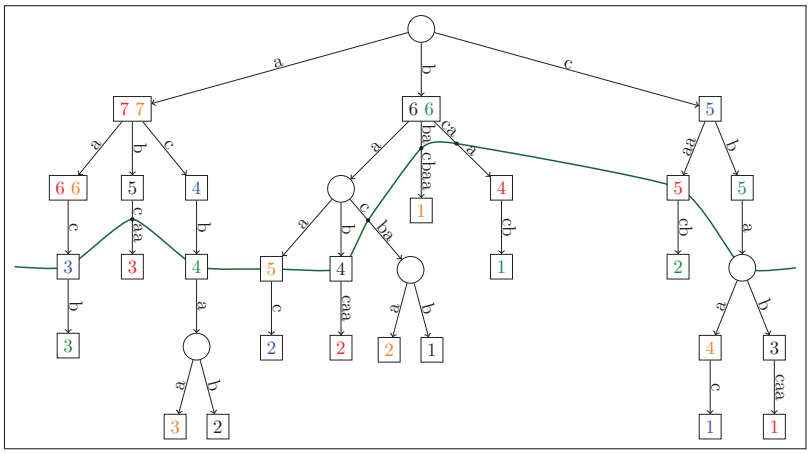
Conclusion de la section En utilisant un arbre des suffixes, on peut construire le graphe de De Bruijn d'ordre k, dBG_k^+, et le graphe de De Bruijn contracté d'ordre k, $CdBG_k^+$ d'un ensemble de mots P, en temps et en espace linéaire en la norme de P.

9.3 En utilisant l'arbre des suffixes tronqué

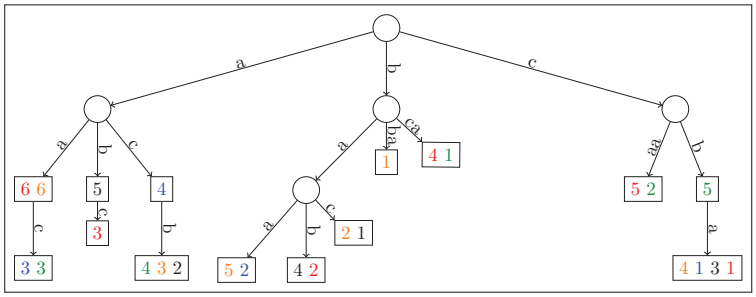
Pour construire le graphe de De Bruijn et sa version contractée d'un ensemble de mots P, on sait qu'on doit lire l'ensemble des mots au moins une fois pour avoir toute l'information disponible. On ne pourra donc pas faire mieux en temps que linéaire en $||P||$. Dans cette partie, on va montrer qu'en gardant une construction linéaire en temps en $||P||$, on peut construire dBG_k^+ et $CdBG_k^+$ en étant en espace linéaire en la taille de dBG_k^+, *i.e.*, en le nombre de sommets et d'arcs de dBG_k^+. Pour faire cela, au lieu de construire tout l'arbre des suffixes de P, on ne va en construire qu'une partie en utilisant l'arbre des suffixes tronqué que l'on a vu dans la partie 6.4.

9.3.1 De l'arbre des suffixes tronqué au dBG_k^+

La proposition 9.19 établit qu'il n'existe pas de feuille de $T(A_k)$ représentant un mot de longueur strictement plus petite que k.



(a)



(b)

FIGURE 9.5 – (a) L'arbre des suffixes généralisé de l'ensemble de mots $\{bacbab, bbacbaa, bcaacb, cbaac, cbabcaa\}$. La partie au dessus de la ligne verte correspond à l'arbre des suffixes tronqué $T(A_2)$. (b) L'arbre des suffixes tronqué $T(A_2)$ pour le même ensemble de mots.

Proposition 9.19 Soit v une feuille de $T(A_k)$. On a alors que $|v| = k$ ou $|v| = k + 1$.

Preuve Pour tout $s_i \in P$ et $j \in \{1, \dots, |s_i| - k + 1\}$, $|A_{k,i}(j)| = k$ ou $k+1$. ■

Pour un lien possible P_0, on définit l'application Q de $\mathcal{H}$ vers $\mathcal{N}$. $\mathcal{H}$, $\mathcal{N}$ et $\mathcal{L}$ ont les mêmes définitions que dans la partie 6.3, mais appliquées à $T(A_k)$. On peut voir $\mathcal{H}$ dans ce cas comme l'ensemble des feuilles de longueur $k+1$ de $T(A_k)$. On définit l'application Q comme suit :

$$\begin{array}{rccl} Q: & \mathcal{H} & \longrightarrow & \mathcal{N} \\ & v & \mapsto & \begin{cases} P_0(v) & \text{si } P_0(v) \in Init_k \\ p_T(P_0(v)) & \text{sinon} \end{cases} \end{array}$$

L'application Q peut être construite en temps linéaire en $||P||$ et permet de simuler les liens suffixes. En effet, pour chaque $v \in \mathcal{H}$, $Q(v)$ peut être construit en temps constant car dans ce cas, $P_0(v) \in Init_k \Leftrightarrow |p_T(P_0(v))| \neq k$. Il suffit donc de considérer la longueur du parent de $P_0(v)$ pour décider si $P_0(v)$ est dans $Init_k$. Comme $|\mathcal{H}| \leq ||P||$, on peut alors construire Q pour tous les éléments de $\mathcal{H}$ en $O(||P||)$.

Proposition 9.20 Soit $v \in \mathcal{L}$, $Q(v) \in Init_k$ et $Q(v) = sl(v)$ si $sl(v)$ existe.

Preuve Soit $v \in \mathcal{L}$. Si $v \in \mathcal{H}$ et $P_0(v) \notin Init_k$, $|p_T(P_0(v))| = k$ et alors $Q(v) = p_T(P_0(v)) \in Init_k$. Par la définition du lien possible Q et de A_k, pour tout nœud v de $\mathcal{L}$, $Q(v)$ est le plus petit nœud de $T(A_k)$ tel que v est un préfixe de $Q(v)$. On a alors que $Q(v) = sl(v)$. ■

Théorème 9.21 On peut construire dBG_k^+ en temps linéaire en $||P||$ et en espace linéaire en $|dBG_k^+|$.

Preuve On commence par construire $T(A_k)$. Avec $T(A_k)$, on peut construire $Init_k$, $InitExact_k$ et $SubInit_k$ comme on le fait dans l'arbre des suffixes généralisé de P. En utilisant Q comme un lien suffixe, on peut construire le graphe (V, E) satisfaisant

$$V = Init_k,$$

$$E = \left(\bigcup_{v \in Init_k, |v| = k+1} (v, Q(v)) \right) \bigcup \left(\bigcup_{v \in Init_k, |v| = k} \left(\bigcup_{u \in Children_T(v)} (v, Q(u)) \right) \right).$$

Ce graphe est alors isomorphe à dBG_k^+. Soit b l'application de $\mathcal{H}$ vers E telle que

$$b(v) = \begin{cases} (v, Q(v)) & \text{if } |p_T(v)| \neq k \\ (p_T(v), Q(v)) & \text{if } |p_T(v)| = k. \end{cases}$$

Comme on vient de montrer que (V, E) est isomorphe à dBG_k^+, l'application b est une bijection. Comme $|\mathcal{L} \setminus \mathcal{H}| \leq |P|$, le cardinal de $\mathcal{L}$ est linéaire en la taille de dBG_k^+. ■

La figure 9.6 montre un exemple de graphe de De Bruijn d'ordre 2 construit à partir de $T(A_2)$.

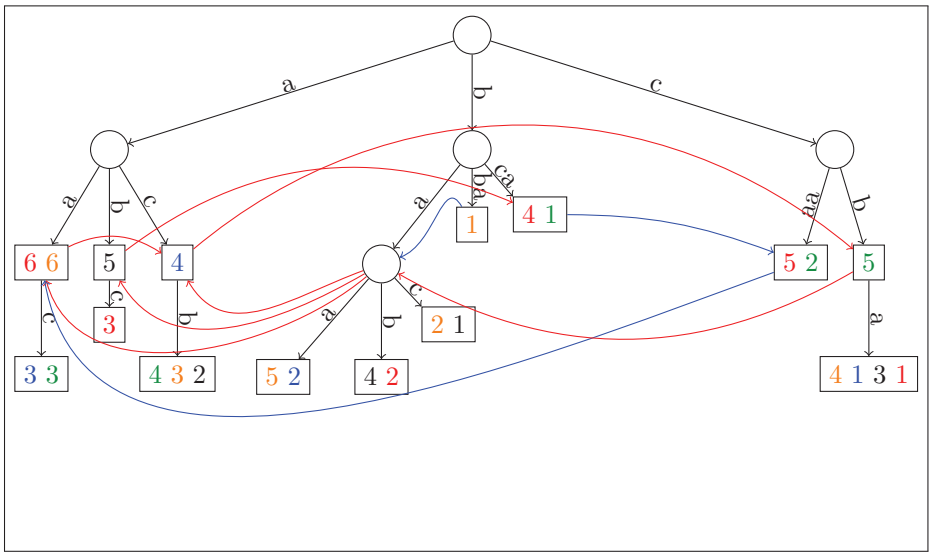


FIGURE 9.6 – Le graphe de De Bruijn d'ordre 2 construit sur $T(A_2)$. Les flèches solides incurvées rouges sont les arcs correspondant à la première partie de la définition de E_k^+, et celles en bleu correspondent à la deuxième partie de la définition de E_k^+.

9.3.2 De l'arbre des suffixes tronqué au $CdBG_k^+$

En reprenant les notations $s(.)$ et $t(.)$ de la section 6.3, on obtient la proposition suivante qui traduit ces notations pour un nœud en fonction du support de ce nœud.

Proposition 9.22 Pour chaque feuille v de $T(A_k)$, $s(v)$ est la taille du support de v dans P et $t(v)$ est la taille de l'ensemble $\big(Occ(v,P) \cap \{(i,1) \mid 1 \leq i \leq n\}\big)$.

Comme les applications $s(.)$ et $t(.)$ sont calculées pendant la construction de l'arbre des suffixes tronqué $T(A_k)$, que la proposition 9.22 nous dit que ces applications résument le support d'un nœud et que les propositions 9.12 et 9.13 nous disent que l'information sur le support des nœuds suffit pour construire de manière linéaire $CdBG_k^+$, on obtient le théorème suivant.

Théorème 9.23 On peut construire $CdBG_k^+$ en temps linéaire en $||P||$ et en espace linéaire en $|dBG_k^+|$.

Au lieu d'utiliser $T(A_k)$ pour construire dBG_k^+ ou $CdBG_k^+$, on peut prendre $T(B_{k+1})$. En effet, $T(B_{k+1})$ est l'arbre $T(A_k)$ avec des feuilles supplémentaires représentant tous les suffixes plus petits que $(k-1)$ des mots de P. Ces feuilles rendent la taille de $T(B_{k+1})$ linéaire en $||P||$ mais pas en $|dBG_k^+|$.

Conclusion de la section On a montré ici que l'on pouvait, en utilisant l'arbre des suffixes tronqué que l'on a défini dans la section 6.4, construire le graphe de De Bruijn d'ordre k, dBG_k^+, et le graphe de De Bruijn contracté d'ordre k, $CdBG_k^+$ d'un ensemble de mots P, en temps linéaire en la norme de P et en espace linéaire en la taille de dBG_k^+. Ces résultats ont été publiés dans (VII) et (II).

9.4 En utilisant l'arbre d'Aho-Corasick généralisé

On va commencer par définir la contraction d'un graphe par un ensemble de représentants.

Définition 9.24 Soient $G = (V, E)$ un graphe et V' un sous-ensemble de V qui va représenter l'ensemble des représentants. La contraction du graphe G par V' est le graphe $Contract(G, V') = (V', E')$ où

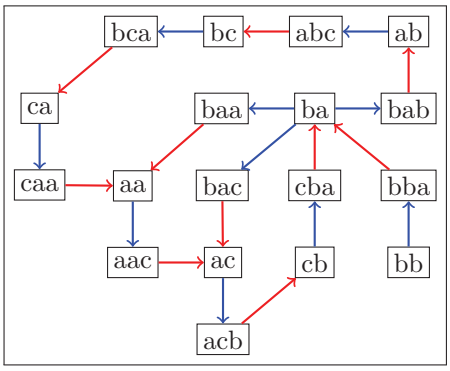
$$E' = \{(u, v) \in V' \times V' \mid \exists p = ((u, u_2), \ldots, (u_k, v)) \text{ un chemin de } G \text{ tel que } \forall i \in \{2, \ldots, k\},\ u_i \in V \setminus V'\}.$$

La contraction d'un graphe par un sous-ensemble V' de ses sommets correspond à un nouveau graphe où V' est le nouvel ensemble de sommets et il existe un sommet entre deux sommets u et v de V' s'il existe un chemin de u vers v tel que tous les nœuds différents de u et v et pris par ce chemin ne sont pas dans V'.

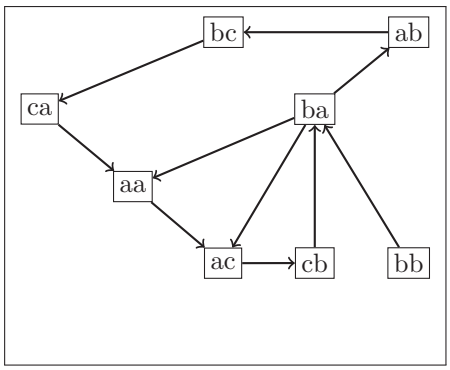
On peut alors définir les graphes dBG_k^+ et dBG_{k+1}^- en fonction d'un arbre d'Aho-Corasick généralisé, ce que l'on va faire avec le théorème suivant :

Théorème 9.25 ***voir Figure 9.7*** Soient P un ensemble de mots et k un entier.

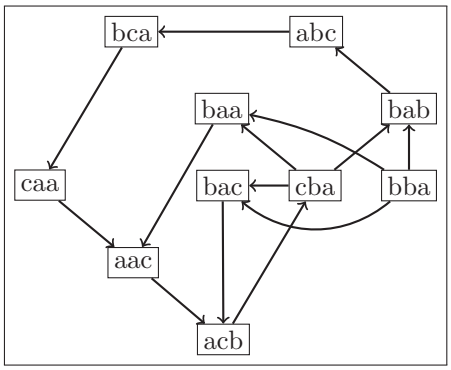
1. Le graphe $Contract(GAC(Fact_k(P) \cup Fact_{k+1}(P)), Fact_k(P))$ est isomorphe au graphe de De Bruijn $dBG_k^+(P)$,
2. Le graphe $Contract(GAC(Fact_k(P) \cup Fact_{k+1}(P)), Fact_{k+1}(P))$ est isomorphe au graphe de De Bruijn $dBG_{k+1}^-(P)$.



(a) $GAC(Fact_k(P) \cup Fact_{k+1}(P))$



(b) $dBG_k^+(P)$



(c) $dBG_{k+1}^-(P)$

FIGURE 9.7 – Exemple de liens entre l'arbre d'Aho-Corasick généralisé et les graphes de De Bruijn. Pour un ensemble $P = \{bacbab, bbacbaa, bcaacb, cbaac, cbabcaa\}$, la figure (a) représente l'arbre d'Aho-Corasick généralisé de l'ensemble de mots $Fact_k(P) \cup Fact_{k+1}(P)$ qui est l'ensemble des k-mers et des $k+1$-mers de P. Les figures (b) et (c) représentent respectivement les graphes $dBG_k^+(P)$ et $dBG_{k+1}^-(P)$.

Grâce au théorème 9.25, on a un lien entre les graphes de De Bruijn (dBG_k^+ et dBG_{k+1}^-) d'un ensemble de mots P et l'arbre d'Aho-Corasick généralisé de l'ensemble des k-mers et des $k+1$-mers de P. De plus, on a un lien fort entre les graphes $dBG_k^+(P)$ et $dBG_{k+1}^-(P)$ car ils correspondent à deux contractions d'un même graphe. Pour la suite de cette section, on va regarder le graphe $dBG_k^-(P)$, on aura des résultats similaires pour $dBG_k^+(P)$.

Pour construire le graphe $dBG_k^-(P)$, on vient d'utiliser l'arbre d'Aho-Corasick généralisé de l'ensemble de mots $Fact_k(P) \cup Fact_{k-1}(P)$. On va voir maintenant que l'on peut récupérer plus d'informations que dans le graphe de De Bruijn en allant chercher dans des chevauchements plus petits. Soient k_1 et k_2 deux entiers tels que $k_1 \leq k_2$. On pose $GAC_{k_1,k_2}(P) = GAC(\bigcup_{i=k_1}^{k_2} Fact_i(P))$ l'arbre d'Aho-Corasick généralisé de tous les facteurs de P de taille comprise entre k_1 et k_2 (voir Figure 9.8).

La taille de $GAC_{k_1,k_2}(P)$ peut être malheureusement bien plus grande que $||P||$. On va alors s'intéresser à appliquer le graphe glouton généralisé sur l'arbre d'Aho-Corasick généralisé $GAC_{k_1,k_2}(P)$. On pose $GSG_{k_1,k_2}(P) = GSG(\bigcup_{i=k_1}^{k_2} Fact_i(P), m)$ où m l'application de $\bigcup_{i=k_1}^{k_2} Fact_i(P)$ vers $\{0,1\}$ telle que pour $x \in Fact_{k_2}(P)$, $m(x) = 1$ et pour $x \in \bigcup_{i=k_1}^{k_2-1} Fact_i(P)$, $m(x) = 0$ (voir Figure 9.9).

On a vu avec le théorème 9.25 que l'on pouvait retrouver les graphes de De Bruijn en faisant une contraction de l'arbre d'Aho-Corasick généralisé. On va faire de même

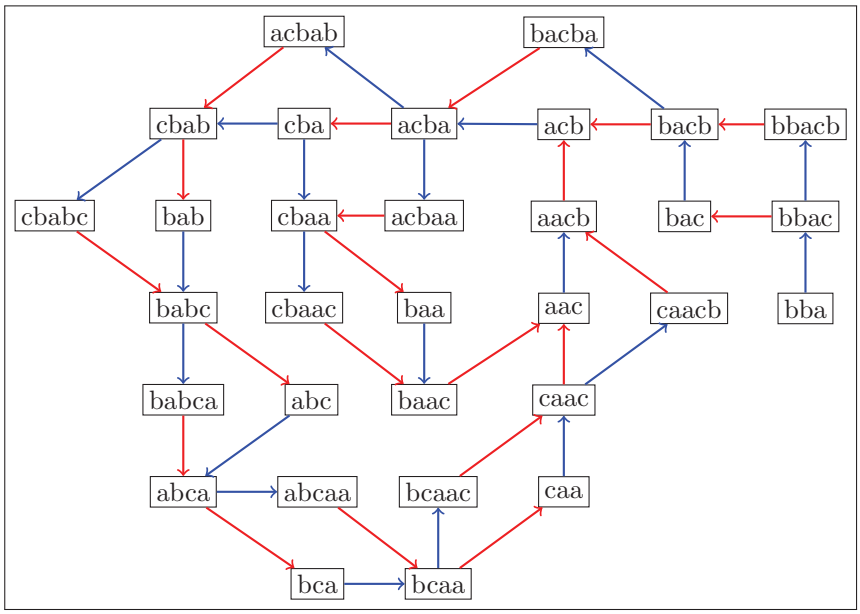


FIGURE 9.8 – Exemple de $GAC_{k_1,k_2}(P)$ pour $P = \{bacbab, bbacbaa, bcaacb, cbaac, cbabcaa\}$, $k_1 = 3$ et $k_2 = 5$.

en ajustant un peu la contraction. En effet, comme on regarde ici la construction de $dBG^-_{k_2}(P)$, on va vouloir prendre comme ensemble de représentants l'ensemble $Fact_{k_2}(P)$ et donc on ne va pas regarder n'importe quel chemin dans l'arbre d'Aho-Corasick généralisé : on va s'intéresser à l'existence d'un RB-path entre deux représentants.

Définition 9.26 Soient $G = (V, E)$ un sous-graphe d'un arbre d'Aho-Corasick généralisé et V' un sous-ensemble de V. La RB-contraction du graphe G par V est le graphe $RB - Contract(G, V') = (V', E')$ où

$$E' = \{(u, v) \in V' \times V' \mid \exists p = ((u, u_2), \dots, (u_k, v)) \text{ un RB-path de } G \text{ tel que } \forall i \in \{2, \dots, k\},\ u_i \in V \setminus V'\}$$

On va alors comparer les graphes $dBG^-_{k_2}(P)$, $RB-Contract(GSG_{k_1,k_2}(P), Fact_{k_2}(P))$, $RB-Contract(GAC_{k_1,k_2}(P), Fact_{k_2}(P))$ et $Contract(GAC_{k_1,k_2}(P), Fact_{k_2}(P))$. On peut se rendre compte rapidement que les ensembles des sommets de ces graphes sont identiques. On va alors comparer plus exactement l'ensemble des arcs de ces graphes.

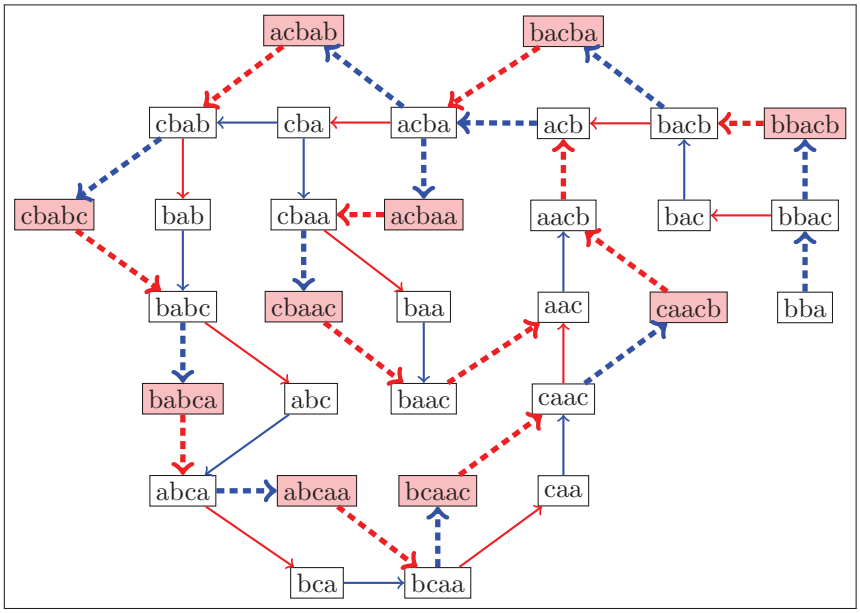
Théorème 9.27 ***voir Figure 9.10*** Soient

- E_{dBG} l'ensemble des arcs de $dBG^-_{k_2}(P)$,
- E_{GSG} l'ensemble des arcs de $RB - Contract(GSG_{k_1,k_2}(P), Fact_{k_2}(P))$,
- E_{RB-GAC} l'ensemble des arcs de $RB - Contract(GAC_{k_1,k_2}(P), Fact_{k_2}(P))$ et
- E_{GAC} l'ensemble des arcs de $Contract(GAC_{k_1,k_2}(P), Fact_{k_2}(P))$.

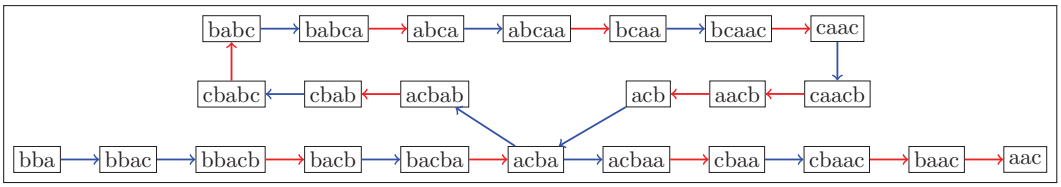
On a alors que :

$$E_{dBG} \subseteq E_{GSG} \subseteq E_{RB-GAC} \subseteq E_{GAC}$$

En calculant le graphe qui correspond à la RB-contraction du graphe glouton généralisé



(a) $GSG_{k_1,k_2}(P)$ en pointillé sur $GAC_{k_1,k_2}(P)$

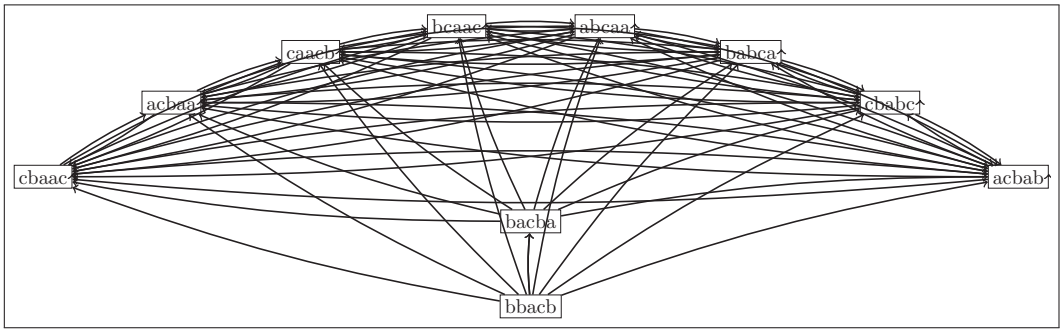


(b) $GSG_{k_1,k_2}(P)$

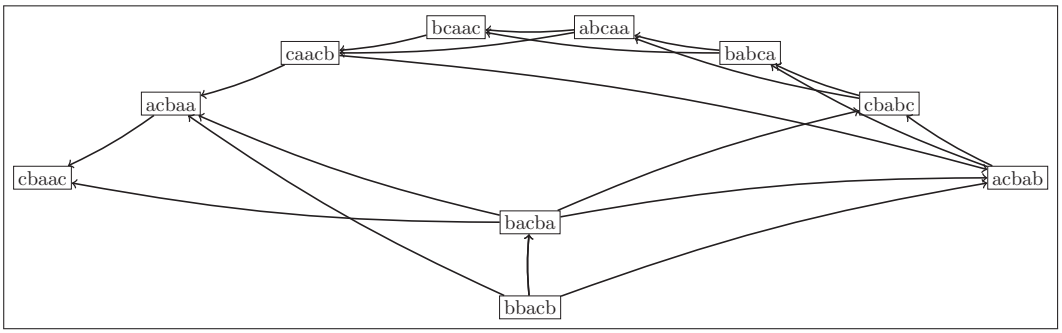
FIGURE 9.9 – Exemple de $GSG_{k_1,k_2}(P)$ pour $P = \{bacbab, bbacbaa, bcaacb, cbaac, cbabcaa\}$, $k_1 = 3$ et $k_2 = 5$. Les arcs rouges correspondent au lien entre un nœud et son plus grand suffixe et les arcs bleus au lien entre le plus grand préfixe d'un nœud et ce nœud.

de l'ensemble des sous-chaînes de P de taille comprise entre k_{min} et k_{max}, on obtient un graphe avec plus d'informations que le graphe de De Bruijn $dBG^-_{k_{max}}(P)$. On a montré dans (IV), que ce nouveau graphe a même plus d'informations que le graphe de De Bruijn multi-ordre que l'on peut retrouver par exemple dans IDBA [73] ou SPAdes [7]. De plus, l'utilisation du graphe glouton généralisé $GAC_{k_{min},k_{max}}(P)$ nous permet de savoir que les chemins couvrants de ce graphe seront des solutions optimales pour le problème Contraint-SCCS prenant en entrée l'ensemble des k_{max}-mers de P comme ensemble de mots et l'ensemble des k_{min}-mers de P comme ensemble de chevauchements interdits.

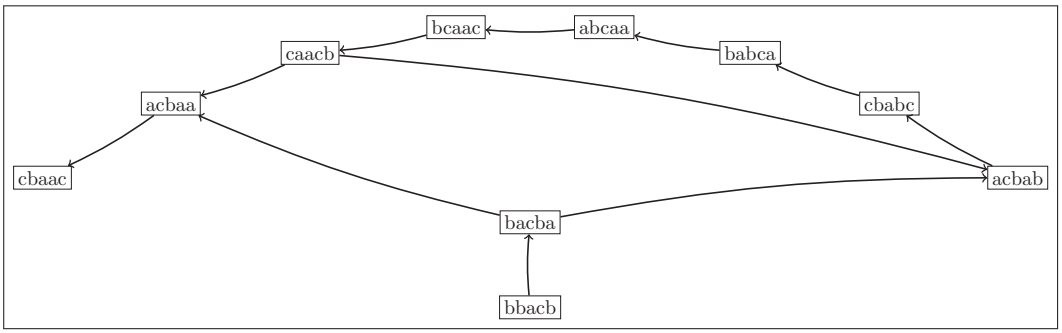
Conclusion de la section On a mis en évidence dans cette section un lien qu'il y avait entre les graphes de De Bruijn : dBG^+_k et dBG^-_{k+1} sont deux contractions du même graphe $GAC_{k,k+1}(P)$. En outre, on a montré que l'utilisation du graphe glouton généralisé appliqué à l'ensemble des k-mers permettait d'obtenir des chevauchements que les graphes de De Bruijn ne pourraient pas déceler (voir (IV)). Enfin, en utilisant les résultats sur le graphe glouton généralisé que l'on a vu dans le chapitre 8 et plus exactement le théorème 8.20, on peut donner un ratio d'approximation pour les solutions données par un parcours du graphe de De Bruijn.



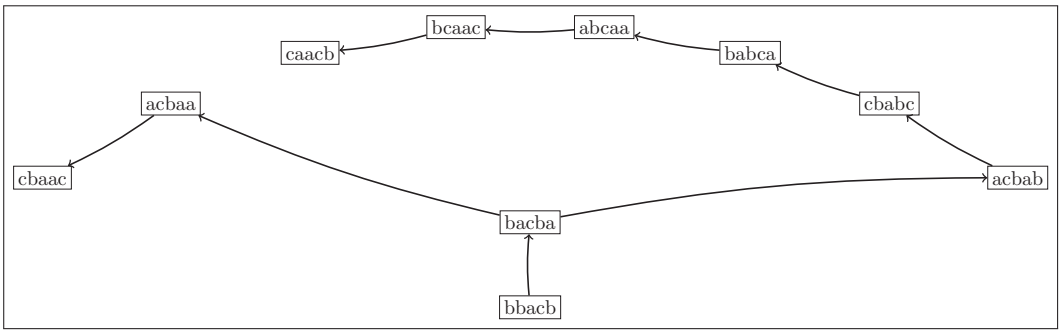
(a) $Contract(GAC_{k_1,k_2}(P), Fact_{k_2}(P))$



(b) $RB - Contract(GAC_{k_1,k_2}(P), Fact_{k_2}(P))$



(c) $RB - Contract(GSG_{k_1,k_2}(P), Fact_{k_2}(P))$



(d) $dBG^{-}_{k_2}(P)$

FIGURE 9.10 – Exemple de graphes pour $P = \{bacbab, bbacbaa, bcaacb, cbaac, cbabcaa\}$, $k_1 = 3$ et $k_2 = 5$. On peut voir, d'une part que les ensembles d'arcs sont bien compris les uns dans les autres, et d'autre part que l'utilisation du graphe glouton généralisé permet de voir plus de chevauchements que dans le cas du graphe de De Bruijn.

Conclusion du chapitre Dans ce chapitre, on a mis en évidence le lien qu'il pouvait y avoir entre les structures d'indexation et plus exactement l'arbre des suffixes, l'arbre des suffixes tronqué, l'arbre d'Aho-Corasick généralisé et le graphe glouton généralisé et les graphes de De Bruijn. Cette étude montre que l'utilisation de structures d'indexations permet, en récupérant de l'information de ces structures, d'améliorer le graphe de De Bruijn en lui faisant regarder des chevauchements qu'il n'aurait pas vu dans sa définition classique.

Conclusion générale

La question de la plus courte superchaine linéaire reste, de par sa simplicité d'énoncé et sa complexité intrinsèque, une question centrale d'algorithmique du texte. Une littérature conséquente lui est consacrée et étudie particulièrement l'approximation de ce problème. Cette thèse introduit de nombreuses variantes naturelles, pertinentes et complexes de cette question, ainsi que des angles d'approches nouveaux pour leur étude. Nos résultats permettent de mieux comprendre les propriétés de l'algorithme glouton pour toutes ces variantes. En outre, ils ont en partie comblé le gap entre les algorithmes théoriques de calcul de superchaînes et les algorithmes d'assemblage de génomes.

La figure 5.8 (page 122) montre les bornes d'approximation pour une petite trentaine de variantes naturelles de SLS. En effet, dans cette thèse, on a introduit un grand nombre de variantes du problème classique SLS. En utilisant les systèmes héréditaires, pour chacun de ces problèmes, on a fourni une méthode pour construire un algorithme glouton sur ce problème. On a de plus montré qu'en utilisant le lien intrinsèque qui existe entre un problème et son système héréditaire, on pouvait donner une borne constante sur le ratio d'approximation optimal de l'algorithme glouton pour ce problème.

La question d'une plus petite couverture circulaire de mots (SCCS) est utilisée comme première étape par la majorité des algorithmes d'approximation de SLS. Pour la résoudre, le meilleur algorithme précédemment connu est utilisé, de complexité temporelle en $O(|P|^3 + ||P||)$, donné par Papadimitriou et al. en 1982 [69]. Nous avons démontré d'une part que l'algorithme glouton pour SCCS est optimal et que sa complexité est linéaire en $||P||$: il est donc optimal pour ce problème. En améliorant cette construction de l'algorithme glouton, on a mis en évidence plusieurs graphes qui nous semblent intéressants. Le premier est l'arbre d'Aho-Corasick généralisé qui nous a permis de comprendre et de généraliser les structures d'indexation classiques telles que l'arbre des suffixes ou l'arbre d'Aho-Corasick. Le deuxième graphe est le graphe hiérarchique des chevauchements qui correspond à l'arbre d'Aho-Corasick généralisé minimum pour conserver l'ensemble des informations sur les chevauchements. On a montré que ce graphe était un bon compromis entre le graphe de De Bruijn et le graphe des chevauchements car tout en restant linéaire en la norme de l'instance, il garde toute l'information contenue dans le graphe des chevauchements. Le troisième graphe est le graphe glouton qui permet de générer les solutions de l'algorithme glouton pour le problème SCCS et peut être appliqué pour les autres variantes de SCCS.

À la question de l'apport d'une structure d'indexation pour faciliter la construction du graphe le plus utilisé pour l'assemblage de génome, le graphe de De Bruijn, nous répondons positivement. En effet, nous avons exhibé les premiers algorithmes linéaires de construction de la version compacte du graphe de De Bruijn à partir de structures d'indexation bien connues. Effectivement, on a montré qu'à partir de l'arbre des suffixes, on pouvait construire en temps et en espace le graphe de De Bruijn classique et compacté en

temps linéaire en la norme de l'instance. On a de plus montré qu'en utilisant un arbre des suffixes tronqué, on pouvait améliorer la complexité en mémoire en la bornant par la taille du graphe de De Bruijn. En appliquant le graphe glouton sur le graphe hiérarchique des chevauchements de l'ensemble des k-mers d'un ensemble de mots, on a trouvé un graphe qui a comme sous-graphe le graphe de De Bruijn et le graphe de De Bruijn multi-ordre. On a donc mis en évidence une nouvelle méthode théorique pour construire une généralisation du graphe de De Bruijn qui ne se limite pas aux chevauchements de longueur $k-1$.

Grâce à des approches nouvelles, notre travail a introduit de nombreuses pistes de recherche future. Malgré les résultats concernant l'algorithme glouton et son lien avec les structures d'indexation, de nombreuses questions demeurent ouvertes. Par exemple, quel est le lien entre le nombre de composantes connexes de l'algorithme glouton et le ratio d'approximation de l'algorithme glouton pour SLS pour cet ensemble de mots ? On pourrait alors peut-être utiliser ce nombre comme paramètre pour la complexité du problème SLS. Pour le graphe hiérarchique des chevauchements, on peut se demander si une construction en temps linéaire en la taille de l'instance ou même une construction directe sans passer par une autre structure d'indexation de celui-ci est possible. Enfin, on peut se demander, si en utilisant le graphe hiérarchique des chevauchements et le graphe glouton, on ne peut pas construire directement une superchaîne. En effet, en transformant le graphe glouton, en connectant les composantes connexes, on trouve des solutions pour SLS. Le graphe glouton a l'air de jouer un rôle prépondérant dans la compréhension de l'algorithme glouton pour SLS. Enfin, même si on a comblé une petite partie de gap entre les algorithmes théoriques de calcul de superchaînes et les algorithmes d'assemblage de génomes, il reste encore de grands vides. En effet, il n'existe pas par exemple à ce jour d'assembleur capable de donner une borne d'approximation théorique pour une solution qu'il pourrait donner. Les résultats théoriques que nous avons donnés ici, pourront peut-être dans un futur proche, aider à combler ce vide.

Index

Symboles

A

C

Liste des symboles

$(e_1 \to \ldots \to e_k)$ la notation d'une permutation circulaire de E, page 24

$(e_1, \ldots, e_{|E|})$ la notation d'une permutation de $E = \{e_1, \ldots, e_{|E|}\}$, page 24

$<$ la relation d'ordre stricte relative à un ordre $\leq$, page 20

$|E|$ le cardinal de l'ensemble E, page 19

$d^{in}(v)$ le nombre de prédécesserus du nœud v, page 55

$d^{out}(v)$ le nombre de successeurs du nœud v, page 55

$\emptyset$ l'ensemble vide, page 19

$\lceil w \rceil$ la plus longue extension de w qui a le même support que w dans P, page 182

$\lfloor w \rfloor$ la plus courte réduction de w qui a un support différent de w dans P, page 182

$\mathbb{N}$ l'ensemble des entiers positifs, page 19

$\mathbb{Z}$ l'ensemble des entiers relatifs, page 20

$\mathbb{Z}/n\mathbb{Z}$ l'ensemble des restes possibles de la division d'un nombre par n, page 22

$\mathcal{P}_c$ la version compression du problème d'optimisation $\mathcal{P}$, page 29

$\mathcal{P}_d$ le problème de décision qui correspond au problème $\mathcal{P}$, page 28

$\mathcal{P}_l$ le problème d'optimisation sur la longueur de la sortie relatif au problème $\mathcal{P}$, page 28

$\langle w \rangle$ le mot circulaire associé au mot w, page 23

$||P||$ la norme de l'ensemble P, page 21

$Ord(G)$ le maximum entre le degré entrant et le degré sortant de tous les nœuds de G, page 55

$\preceq$ l'ordre lexicographique associé à l'ordre $\leq$, page 22

$\overleftarrow{w}$ le complémentaire-renversé du mot w, page 26

$\overleftarrow{\square}^{\varphi}$ l'application de composition d'une permutation et d'un renversé, page 26

Σ^* l'ensemble des mots sur Σ, page 21

Σ^k l'ensemble des mots de longueur k sur Σ, page 21

$|w|$ la longueur du mot w, page 21

ε le mot vide, page 21

Bibliographie

À chaque référence, j'ai ajouté un petit résumé de l'information qui est importante relativement au sujet traité.

Légende

- Graphe
- Aprroximation
- Complexité
- Superchaîne
- Algorithme glouton
- Système héréditaire
- Arbre d'indexation

[1] 6.1

Alfred V. Aho and Margaret J. Corasick. Efficient string matching : An aid to bibliographic search. *Commun. ACM*, 18(6) :333–340, 1975.

Algorithme linéaire pour trouver les localisations d'un ensemble de mots dans un texte. Création et utilisation de l'arbre d'Aho-Corasick dans l'algorithme.

[2] 4.1.1

Kenneth S. Alexander. Shortest common superstrings for strings of random letters. In *Combinatorial Pattern Matching, 5th Annual Symposium, CPM 94, Asilomar, California, USA, June 5-8, 1994, Proceedings*, pages 164–172, 1994.

Analyse sur le comportement moyen de la taille de la solution optimale de SLS et expression asymptotique pour la somme des chevauchements (en $\frac{|P|\times log(|P|}{H}$ où H est l'entropie), *i.e.* relativement à SLS_c.

[3] Introduction, 2, 2.1.3, 2.3

Chris Armen and Clifford Stein. Improved length bounds for the shortest superstring problem (extended abstract). In *Algorithms and Data Structures, 4th International Workshop, WADS '95, Kingston, Ontario, Canada, August 16-18, 1995, Proceedings*, pages 494–505, 1995.

Présentation d'un algorithme avec un ratio d'approximation de $(2 + \frac{50}{69})$ pour SLS en temps $O(||P|| + |P|^3)$ (utilisation de l'algorithme de [49]).

[4] **Introduction, 2, 2.1.3, 2.3, 5**

Ⓐ **Chris Armen and Clifford Stein. Short superstrings and the structure of overlapping strings.** ***Journal of Computational Biology*****, 2(2) :307–332, 1995.**

Présentation d'un algorithme avec un ratio d'approximation de $(2 + \frac{3}{4})$ pour SLS en temps $O(||P|| + |P|^3)$.

[5] **Introduction, 2, 2.1.3, 2.3**

Ⓐ **Chris Armen and Clifford Stein. A 2 2/3-approximation algorithm for the shortest superstring problem. In** ***Combinatorial Pattern Matching, 7th Annual Symposium, CPM 96, Laguna Beach, California, USA, June 10-12, 1996, Proceedings*****, pages 87–101, 1996.**

Introduction d'un algorithme générique pour le problème SLS (rassemblant les algorithmes de [11, 18, 88]) avec un ratio d'approximation de $(2 + \frac{2}{3})$ (généralisation de l'algorithme de [4] mais analyse différente).

[6] **2.1.3**

Ⓒ **Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and hardness of approximation problems. In** ***33rd Annual Symposium on Foundations of Computer Science, Pittsburgh, Pennsylvania, USA, 24-27 October 1992*****, pages 14–23, 1992.**

Preuve qu'un problème **Max-SNP-Difficile** n'admet pas de PTAS sous l'hypothèse $\mathbf{P} \neq \mathbf{NP}$.

[7] **Introduction, 9.4**

Anton Bankevich, Sergey Nurk, Dmitry Antipov, Alexey A. Gurevich, Mikhail Dvorkin, Alexander S. Kulikov, Valery M. Lesin, Sergey I. Nikolenko, Son K. Pham, Andrey D. Prjibelski, Alex Pyshkin, Alexander Sirotkin, Nikolay Vyahhi, Glenn Tesler, Max A. Alekseyev, and Pavel A. Pevzner. Spades : A new genome assembly algorithm and its applications to single-cell sequencing. ***Journal of Computational Biology*****, 19(5) :455–477, 2012.**

Prèsentation d'un logiciel pour l'assemblage de génome utilisant le graphe de De Bruijn multi-ordre.

[8] **Introduction**

Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *J. ACM*, 9(1) :61–63, 1962.

Algorithme exact pour Max-ATSP en $O^{\#}(2^{|P|})$ en temps et en espace (où $O^{\#}()$ cache les facteurs polynomiaux en $||P||$).

[9] **Introduction**

Davide Bilò, Hans-Joachim Böckenhauer, Dennis Komm, Richard Královic, Tobias Mömke, Sebastian Seibert, and Anna Zych. Reoptimization of the shortest common superstring problem. *Algorithmica*, 61(2) :227–251, 2011.

Étude de deux problèmes de réoptimisation SLS+ et SLS- :

- SLS+ et SLS- sont **NP-Difficiles**,
- présentation de deux algorithmes avec des ratios d'approximation de $(1 + \frac{5}{6})$ (en $O(|P| \times ||P||)$) et $(1 + \frac{3}{8})$ pour SLS+,
- présentation d'un algorithme avec un ratio d'approximation de $(1 + \frac{6}{7})$ pour SLS-.

[10] **2.1.3**

Ivan Bliznets, Fedor V. Fomin, Petr A. Golovach, Nikolay Karpov, Alexander S. Kulikov, and Saket Saurabh. Parameterized complexity of superstring problems. In *Combinatorial Pattern Matching - 26th Annual Symposium, CPM 2015, Ischia Island, Italy, June 29 - July 1, 2015, Proceedings*, pages 89–99, 2015.

Étude de Partial-SCS$_d$ (version de SLS$_d$ où on ne demande qu'à voir un nombre fixe k de mots de l'instance dans une superchaine de taille l) :

- Partial-SCS$_d$ est FPT si paramétré par l et n'admet pas de noyau polynomial avec les paramètres $k+m$ ou $l+m$ pour des mots de taille au plus m sur un alphabet binaire tant que **NP** $\subset$ **coNP/poly**,
- SLS$_d$ admet un noyau de taille $O(r^4)$ (oú $r = ||P|| - l$),
- SLS$_d$ est **NP-complet** pour $l = ||P|| - \mu(P) - 1$ (où $\mu(P)$ est le couplage de poids maximum dans $G = (P, \mathcal{P}_2(P), w)$ et $w(\{s_1, s_2\}) = max(|ov(s_1, s_2)|, |ov(s_2, s_1)|)$.

[11] **Introduction, 2, 2.1.3, 2.1.3, 2.3, 2.34, 2.3.2, 4, 4.1, 4.1.1, 4.3.2, 5, 44, 49, 78**

Avrim Blum, Tao Jiang, Ming Li, John Tromp, and Mihalis Yannakakis. Linear approximation of shortest superstrings. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, pages 328–336, 1991.

Étude de SLS et SLS_c :

- SLS et SLS_c sont **Max-SNP-difficiles**,
- premier algorithme avec un ratio approximation en $O(1)$ pour SLS (3-approximation),
- premier algorithme avec un ratio en $O(1)$ de l'algorithme glouton pour SLS (4-approximation),
- première observation que l'algorithme glouton est optimal pour SCCS.

[12] **Introduction**

Ⓐ **Dirk Bongartz. On the approximation ratio of the group-merge algorithm for the shortest common suerstring problem. In *SOFSEM 2000 : Theory and Practice of Informatics, 27th Conference on Current Trends in Theory and Practice of Informatics, Milovy, Czech Republic, November 25 - December 2, 2000, Proceedings*, pages 298–306, 2000.**

Étude de l'algorithme Group-merge pour SLS :

- preuve que la borne inférieure de Group-merge donnée par [41] n'est pas bonne : ratio d'approximation de $(1 + \frac{1}{4})$ pour $m = 2$ et optimal pour $m \geq 3$,
- nouvelle borne inférieure du ratio d'approximation de Group-merge de $1 + \frac{1}{5}$ pour SLS
- $\frac{1}{12}$-approximation pour $k - SLS_c$,
- ratio d'approximation de Group-merge dans des cas précis de SLS.

[13] **3.4.1**

Marília D. V. Braga and Joao Meidanis. An algorithm that builds a set of strings given its overlap graph. In *LATIN 2002 : Theoretical Informatics, 5th Latin American Symposium, Cancun, Mexico, April 3-6, 2002, Proceedings*, pages 52–63, 2002.

Étude de la reconnaissance d'un graphe k-overlap (un graphe des chevauchements sans pondération où on ne garde que les arêtes avec une pondération plus grande ou égale à k) : algorithme exponentiel en le degré maximum du graphe G en entrée et polynomial en k, en la taille du graphe G et en la taille de l'alphabet.

[14] **Introduction**, **2**, **2.1.3**, **2.3**, **43**, **44**, **63**

Ⓐ **Dany Breslauer, Tao Jiang, and Zhigen Jiang. Rotations of periodic strings and short superstrings. *J. Algorithms*, 24(2) :340–353, 1997.**

Étude d'approximation pour SLS :

— présentation d'un algorithme avec un ratio d'approximation de $(2 + \frac{2}{3})$ pour SLS,

— présentation d'un algorithme avec un ratio d'approximation de $(2 + \frac{25}{42})$ pour SLS,

— preuve que si on a une α-approximation pour Max-ATSP, on peut construire un algorithme avec un ratio d'approximation de $(\frac{7}{2} - \frac{3}{2}\alpha)$ pour SLS.

[15] **Introduction**

Ⓐ Ⓒ 😑 **Raphaël Clifford, Zvi Gotthilf, Moshe Lewenstein, and Alexandru Popa. Restricted common superstring and restricted common supersequence. In *Combinatorial Pattern Matching - 22nd Annual Symposium, CPM 2011, Palermo, Italy, June 27-29, 2011. Proceedings*, pages 467–478, 2011.**

Lien entre l'AI planning et les superchaînes et étude de Restricted-SLS (pour un ensemble de mots P et un multi-ensemble t, on cherche l'ordre de t qui maximise le nombre de mots de P qui sont sous-chaînes du mot issu de ce sous-ordre) :

— Restricted-SLS est **NP-complet** et difficile à approximer avec un facteur plus petit que $n^{1-\epsilon}$ même dans le cas où :

  — les mots sont de taille 2 et t est un ensemble,

  — l'alphabet est de taille 2,

— présentation d'un algorithme avec un ratio d'approximation de $\frac{3}{4}$ pour 2-Restricted-SLS,

— présentation d'un algorithme avec un ratio d'approximation de $\frac{1}{l(\frac{l(l+1)}{2}-1)}$ pour Restricted-SLS avec des mots de taille plus petite ou égale à l.

[16] **Introduction**, **3.3**, **5.2**

Ⓒ 😑 **Maxime Crochemore, Marek Cygan, Costas S. Iliopoulos, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Algorithms for three versions of the shortest common superstring problem. In *Combinatorial Pattern Matching, 21st Annual Symposium, CPM 2010, New York, NY, USA, June 21-23, 2010. Proceedings*, pages 299–309, 2010.**

Étude de trois versions de SLS :

— Algorithme exact en temps $O(||P||^2)$ pour 2-Multi-SLS ($O(||P||)$ pour avoir la longueur de la superchaîne).

— multi-SLS peut être résolu en temps polynomial en la norme de P si $|P|$ est bornée,

— sum-SLS peut être résolu en temps $O(||P|| + |P|^3 \times log(m))$ et en espace $O(||P|| + |P|^2 \times log(m))$ (où m est le paramètre de sum-SLS).

[17] **6.2, 7.1**
Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithme du texte*. Vuibert, 2001.

Livre de référence de l'algorithmique du texte.

[18] **Introduction, 2, 2.6, 2.1.3, 2.3, 4.1.1, 5, 19**
Artur Czumaj, Leszek Gasieniec, Marek Piotrów, and Wojciech Rytter. Parallel and sequential approximations of shortest superstrings. In *Algorithm Theory - SWAT '94, 4th Scandinavian Workshop on Algorithm Theory, Aarhus, Denmark, July 6-8, 1994, Proceedings*, pages 95–106, 1994.

Études sur les problèmes des superchaînes et la parallélisation :
- présentation d'un algorithme avec un ratio d'approximation de $(2 + \frac{5}{6})$ pour SLS (affinage de la borne d'approximation de [88]),
- présentation d'un algorithme avec un ratio d'approximation de $\frac{1}{4+\epsilon}$ pour SLS_c (algorithme **NC**, *i.e.* polylogarithmique sur un nombre polynomial de machines en parallèle),
- preuve que l'algorithme glouton pour SLS est **P-complet** donc difficile à paralléliser,
- l'algorithme glouton pour SCCS à un ratio d'approximation de $\frac{1}{2}$ sans les autochevauchements,
- présentation d'un algorithme avec un ratio d'approximation de $\frac{1}{2+\epsilon}$ pour SCCS_c (algorithme **NC**)
- présentation d'un algorithme avec un ratio d'approximation de $(2 + \epsilon) \times log(|P|)$ pour SLS (algorithme **NC**),
- présentation d'un algorithme avec un ratio d'approximation de 4 pour SLS (algorithme **Randomized NC**).

[19] **3.4.1, 4.1, 4.3.2**
Artur Czumaj, Leszek Gasieniec, Marek Piotrów, and Wojciech Rytter. Sequential and parallel approximation of shortest superstrings. *J. Algorithms*, 23(1) :74–100, 1997.

Version longue de [18], avec :
- résultat sur la reconnaissance d'un graphe des chevauchements (avec poids non nuls)
- mention que l'algorithme glouton pour SCCS est optimal,
- construction d'un graphe des chevauchements en temps $O(log(|P|)$ en utilisant $O(|P|^2)$ processeurs.

[20] 7.1

Nicolaas Govert de Bruijn. A combinatorial problem. In *Koninklijke Nederlandse Akademie v. Wetenschappen*, volume 49, pages 758–764, 1946.

Introduction au graphe de De Bruijn original.

[21] 2

Patricia A. Evans and H. Todd Wareham. *Efficient Restricted-Case Algorithms for Problems in Computational Biology*, pages 27–49. John Wiley & Sons, Inc., 2011.

Application de SLS à l'assemblage de l'ADN.

[22] Introduction, 4.1.1

Gabriele Fici, Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. On the greedy algorithm for the shortest common superstring problem with reversals. *Inf. Process. Lett.*, 116(3) :245–251, 2016.

Preuve que Reverse-SLS est **NP-complet**, que l'algorithme glouton pour Reverse-SLS$_c$ a un ratio d'approximation de $\frac{1}{2}$ et implémentation de l'algorithme glouton pour Reverse-SLS$_c$ en temps linéaire en $||P||$.

[23] 4.1.1

Alan M. Frieze and Wojciech Szpankowski. Greedy algorithms for the shortest common superstring that are asmtotically optimal. In *Algorithms - ESA '96, Fourth Annual European Symposium, Barcelona, Spain, September 25-27, 1996, Proceedings*, pages 194–207, 1996.

Preuve que la moyenne de la somme des chevauchements d'une solution optimale pour SLS est asymptotiquement équivalente à la moyenne de la somme des chevauchements d'une solution de l'algorithme glouton pour SLS (et asymptotiquement équivalent à $\frac{|P|\times log(|P|)}{H}$ où H est l'entropie).

[24] 3.4.1, 4, 4.1.1

John Gallant. *String compression algorithms*. PhD thesis, Princeton University, 1982.

Preuve que SLS est **NP-complet** et introduction de l'algorithme glouton pour SLS et du graphe des chevauchements.

[25] **2.1.3, 2.2.3, 3.3, 30**

Ⓐ Ⓒ ⛀ **John Gallant, David Maier, and James A. Storer. On finding minimal length superstrings. *J. Comput. Syst. Sci.*, 20(1) :50–58, 1980.**

Ce papier donne une preuve que SLS_d est **NP-complet** :

— **(alphabet non borné)** même si pour tout $k \geq 3$, les mots de l'instance de départ sont primitifs (chaque lettre apparait au plus une fois dans chaque mot) et de longueur k. La preuve est basée sur une réduction polynomiale à partir de dHPP (*directed Hamilton Path Problem*).

— **(alphabet borné)** même si pour tout $h > 1$, les mots de l'instance de départ S sont sur l'alphabet $\{0, 1\}$ et de longueur $\lceil h.\log_2(||S||)\rceil$.

Il y a aussi une preuve qu'il existe un algorithme linéaire (en temps et en espace) pour trouver une solution optimale de SLS dans le cas où les mots de l'instance de départ sont de longueur 2 ou moins.

[26] **1.8, 4**

Ⓒ ⛀ **M. R. Garey and David S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.**

Preuve que SLS est **NP-complet**.

[27] **2.1.3**

⚇ Ⓐ Ⓒ ⛀ Ⓖ **Theodoros P. Gevezes and Leonidas S. Pitsoulis. *Optimization in Science and Engineering : In Honor of the 60th Birthday of Panos M. Pardalos*, chapter The Shortest Superstring Problem, pages 189–227. Springer New York, New York, NY, 2014.**

Revue complète du problème SLS.

[28] **3.4.1, 4.1.3.1**

⚇ **Theodoros P. Gevezes and Leonidas S. Pitsoulis. Recognition of overlap graphs. *J. Comb. Optim.*, 28(1) :25–37, 2014.**

Étude de la reconnaissance du graphe des chevauchements (sans auto-chevauchement) :

— l'ensemble des d-trees (un graphe orienté tel que son graphe sous-jacent est un arbre) est un graphe des chevauchements pour toute fonction de poids,

— détermination d'une solution représentative du fait que le graphe soit un graphe de chevauchements ou pas (prendre chaque mot de longueur $max_{in} + max_{out} + 1$, où max_{in} (resp. max_{out} est le nombre d'arêtes entrantes (resp. sortantes) du noeud représentant le mot),

— algorithme polynomial pour la reconnaissance du graphe des chevauchements (trouver une solution possible puis la vérifier).

[29] **2.1**

T. R. Gingeras, J. P. Milazzo, D. Sciaky, and R. J. Roberts. Computer programs for the assembly of dna sequences. *Nucleic Acids Research*, 7(2) :529–545, 1979.

Méthode d'assemblage issue de SLS.

[30] **Introduction, 3.3**

Alexander Golovnev, Alexander S. Kulikov, and Ivan Mihajlin. Approximating shortest superstring problem using de bruijn graphs. In *Combinatorial Pattern Matching, 24th Annual Symposium, CPM 2013, Bad Herrenalb, Germany, June 17-19, 2013. Proceedings*, pages 120–129, 2013.

Présentation d'un algorithme avec un ratio d'approximation de $(\frac{k^2+k-4}{4k-6})$ pour k-SLS en $O(|P|^3 \times ||P||)$ (l'algorithme consiste à calculer deux solutions : la première avec l'algorithme pour Max-ATSP de [68] qui a un ratio d'approximation de $\frac{2}{3}$ et l'autre avec un graphe de Bruijn de [25] et prend la meilleure des deux solutions).

[31] **Introduction**

Alexander Golovnev, Alexander S. Kulikov, and Ivan Mihajlin. Solving 3-superstring in 3 n/3 time. In *Mathematical Foundations of Computer Science 2013 - 38th International Symposium, MFCS 2013, Klosterneuburg, Austria, August 26-30, 2013. Proceedings*, pages 480–491, 2013.

Algorithme exact pour 3-SLS en temps $O^{\#}(3^{\frac{|P|}{3}})$ et en espace polynomial (où $O^{\#}()$ cache les facteurs polynomiaux en $||P||$).

[32] **Introduction, 7.2.3**

Alexander Golovnev, Alexander S. Kulikov, and Ivan Mihajlin. Solving SCS for bounded length strings in fewer than 2^n steps. *Inf. Process. Lett.*, 114(8) :421–425, 2014.

Algorithme exact randomisé pour k-SLS en temps $O^{\#}(2^{(1-\frac{1}{2k^2+1})\times|P|})$ (définition du Hierarchical Graph et caractérisation du nombre de composantes faiblement connexes des arêtes du « *hierarchical graph* » qui seront forcément prises : au plus $(1-\frac{1}{2k^2+1}) \times |P|$.

[33] **Introduction**

Ⓒ ☰ **Zvi Gotthilf, Moshe Lewenstein, and Alexandru Popa. On shortest common superstring and swap permutations. In *String Processing and Information Retrieval - 17th International Symposium, SPIRE 2010, Los Cabos, Mexico, October 11-13, 2010. Proceedings*, pages 270–278, 2010.**

Étude de Restricted-SLS et de Repetion-RSLS (problème relatif à Restricted-SLS où on maximise le nombre de fois que l'on voit un mot de P et pas le nombre de mots de P) :

— Restricted-SLS est **NP-complet**,

— algorithme polynomial en $O(|P| \times ||P|| \times l)$ pour Repetion-RSLS où tous les mots de P ont une taille inférieure ou égale à l.

[34] **Introduction, 2.3.1, 3.4.1**

Ⓐ ☰ **Dan Gusfield. Faster implementation of a shortest superstring approximation. *Inf. Process. Lett.*, 51(5) :271–274, 1994.**

Amélioration de la méthode d'approximation de [88] en temps de calcul : $O(||P|| + |P|^3)$ au lieu de $O(|P| \times ||P|| + |P|^3)$ (algorithme pour Canonical-assignement (problème qui pour une couverture circulaire de mots d'un ensemble de mots associe chaque mot à sa couverture circulaire de mots) en $O(||P||$ au lieu de $O(|P| \times ||P||))$.

[35] **3.4.1, 6.2**

⚇ Ⓐ **Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology.* Cambridge University Press, 1997.**

Ce livre présente plusieurs algorithmes intéressants sur l'arbre des suffixes comme par exemple comment construire en $O(||P||)$ l'ensemble des chevauchements maximaux de P et en $O(|P|^2 + ||P||)$ le graphe des chevauchements de P.

[36] **Introduction**

☰ **Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. In *Proceedings of the 1961 16th ACM National Meeting*, ACM '61, pages 71.201–71.204, New York, NY, USA, 1961. ACM.**

Algorithme exact pour SLS en $O^{\#}(2^{|P|})$ en temps et en espace (algorithme de programmation dynamique et où $O^{\#}()$ cache les facteurs polynomiaux en $||P||$).

[37] **Introduction, 2**

Ⓣ **Lucian Ilie and Cristian Popescu. The shortest common superstring problem and viral genome compression. *Fundam. Inform.*, 73(1-2) :153–164, 2006.**

Utilisation de SLS pour le modèle du *viral genome compression*.

[38] **5**

Ⓒ Ⓖ Ⓢh **T. A. Jenkyns. The greedy travelling salesman's problem. *Networks*, 9(4) :363–373, 1979.**

Introduction de l'algorithme glouton pour Min-ATSP et Max-ATSP avec les systèmes héréditaires.

[39] **Introduction, 4.1.1**

Ⓒ Ⓣ **Tao Jiang and Ming Li. On the complexity of learning strings and sequences. *Theor. Comput. Sci.*, 119(2) :363–371, 1993.**

Introduction et étude de Consistent-SLS (version de SLS où on a en plus un ensemble de mots (dit négatif) que l'on ne veut pas voir dans la superchaîne finale) :

— Consistent-SLS est **NP-complet**,

— algorithme polynomial pour Consistent-SLS dans le cas où on borne le nombre de mots positifs.

[40] **Introduction**

Ⓐ Ⓣ Ⓖ **Tao Jiang and Ming Li. Approximating shortest superstrings with constraints. *Theor. Comput. Sci.*, 134(2) :473–491, 1994.**

Algorithme d'approximation pour des cas spéciaux de Consistent-SLS et borne supérieure de $O(n^{\frac{4}{3}})$ pour l'algorithme glouton pour Consistent-SLS.

[41] **Introduction, 12**

Ⓐ Ⓣ **Tao Jiang and Ming Li. DNA sequencing and string learning. *Mathematical Systems Theory*, 29(4) :387–405, 1996.**

Preuve d'une borne inférieure (exemple) pour l'algorithme Group-Merge en $|P| \times \log(|P|)$ pour SLS, pour k-SACS et pour Consistent-SLS.

[42] **Introduction**

Ⓐ Ⓒ Ⓣ Ⓖ **Tao Jiang, Ming Li, and Ding-Zhu Du. A note on shortest superstrings with flipping. *Inf. Process. Lett.*, 44(4) :195–199, 1992.**

Introduction de Reverse-SLS (version de SLS avec possibilité de changer le sens des mots), observation que Reverse-SLS est **NP-Difficile** (sans preuve) et 4-approximation pour l'algorithme glouton pour Reverse-SLS.

[43] **Introduction, 3.4, 3.4.3**

Haim Kaplan, Moshe Lewenstein, Nira Shafrir, and Maxim Sviridenko. Approximation algorithms for asymmetric TSP by decomposing directed regular multigraphs. *J. ACM*, 52(4) :602–626, 2005.

— Présentation d'un algorithme avec un ratio d'approxiation de $\frac{2}{3}$ pour Max-ATSP et pour SLS_c,

— Présentation d'un algorithme avec un ratio d'approxiation de $(2+\frac{1}{2})$ pour SLS (en utilisant le résultat de [14]).

[44] **Introduction, 2, 2.1.3, 2.3, 3.3, 4.1.1**

Haim Kaplan and Nira Shafrir. The greedy algorithm for shortest superstrings. *Inf. Process. Lett.*, 93(1) :13–17, 2005.

Preuve que l'algorithme glouton pour SLS est une $\frac{7}{2}$-approximation (reprise de la preuve de [11] que l'algorithme glouton pour SLS est une 4-approximation et en y ajoutant un résultat de [14]).

[45] **Introduction**

Richard M. Karp. Dynamic programming meets the principle of inclusion and exclusion. *Operations Research Letters*, 1(2) :49 – 51, 1982.

— TSP (Hamiltonian Cycle Problem = Traveling Salesman Problems) et HPP (Hamiltonian Path Problem) sont **NP-complets**,

— Algorithme exact pour SLS et Max-ATSP en $O^{\#}(2^{|P|})$ en temps et en espace (algorithme inclusion-exclusion et où $O^{\#}()$ cache les facteurs polynomiaux en $||P||$).

[46] **2.1.3**

Marek Karpinski and Richard Schmied. Improved lower bounds for the shortest superstring and related problems. *CoRR*, abs/1111.5442, 2011.

Étude de bornes inférieures pour les ratios d'approximation : $1+\frac{1}{332}$ pour SLS et $1-\frac{1}{203}$ pour SLS_c.

[47] **2.1**

John D. Kececioglu and Eugene W. Myers. Combinatiorial algorithms for DNA sequence assembly. *Algorithmica*, 13(1/2) :7–51, 1995.

Étude théorique et pratique pour l'assemblage de génomes de manière générale avec une couverture incomplète, des erreurs de séquences et l'absence d'information sur la localisation des fragments et sur leur orientation.

[48] **Introduction**

Samuel Kohn, Allan Gottlieb, and Meryle Kohn. A generating function approach to the traveling salesman problem. In *Proceedings of the 1977 Annual Conference*, ACM '77, pages 294–300, New York, NY, USA, 1977. ACM.

Algorithme exact pour SLS et Max-ATSP en $O^{\#}(2^{|P|})$ en temps et en espace (algorithme inclusion-exclusion et où $O^{\#}()$ cache les facteurs polynomiaux en $||P||$).

[49] **Introduction, 3**

S. Rao Kosaraju, James K. Park, and Clifford Stein. Long tours and short superstrings (preliminary version). In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 166–177, 1994.

Étude de SLS et SLS_c :

- présentation d'un algorithme avec un ratio d'approxiation de $\frac{38}{63}$ pour Max-ATSP et SLS_c,
- présentation d'un algorithme avec un ratio d'approxiation de $(2+\frac{50}{63})$ pour SLS (en utilisant résultat de [11]).

[50] **3.31, 4.1.1**

Alexander S. Kulikov, Sergey Savinov, and Evgeniy Sluzhaev. Greedy conjecture for strings of length 4. In *Combinatorial Pattern Matching - 26th Annual Symposium, CPM 2015, Ischia Island, Italy, June 29 - July 1, 2015, Proceedings*, pages 307–315, 2015.

Preuve que l'algorithme glouton pour 4-SLS a un ratio d'approximation de 2.

[51] **4.1.3**

Uli Laube and Maik Weinard. Conditional inequalities and the shortest common superstring problem. *Int. J. Found. Comput. Sci.*, 16(6) :1219–1230, 2005.

Amélioration et extension des résultats de [95] : preuve de la conjecture gloutonne pour SLS pour certaines instances et preuve que la triple inégalité n'est pas suffisante pour prouver la conjecture gloutonne.

[52] **Introduction, 2**

Arthur M. Lesk. *Computational Molecular Biology : Sources and Methods for Sequence Analysis*. Oxford University Press, 1989.

Application de SLS en biologie computationnelle : la plus petite superchaîne d'un ensemble de mots préserve les structures biologiques importantes.

[53] **Introduction, 2**

Ming Li. Towards a DNA sequencing theory (learning a string) (preliminary version). In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume I*, pages 125–134, 1990.

Étude de SLS et application de SLS en biologie computationnelle : première présentation de l'algorithme Group-Merge, ratio d'approximation conjecturé à 2 et prouvé en $O(|P| \times log(|P|))$.

[54] **4.1.1**

Xuan Liu and Ondrej Sýkora. Sequential and parallel algorithms for the shortest common superstring problem. In M. Vajtersic, R. Trobec, P. Zinterhof, and A. Uhl, editors, *Parallel Numerics 05*, chapter 5, pages 97–107. International Workshop on Parallel Numerics, 2005.

Implémentation d'algorithmes séquentiels pour SLS tels que l'algorithme glouton, GA (Genetic Algo) et SA (Simulated Annealing) ainsi que des algorithmes parallèles tels que parallele island GA et parallel SA et comparaison des résultats.

[55] **1.11**

Roger C. Lyndon. On Burnside's problem. *Trans. Amer. Math. Soc.*, 77 :202–215, 1954.

Dans cet article, on a la première définition des mots de Lyndon (appelé dans l'article *standart lexicographic sequences*).

[56] **Introduction, 4.1.1**

Bin Ma. Why greed works for shortest common superstring problem. *Theor. Comput. Sci.*, 410(51) :5374–5381, 2009.

Analyse lisse d'algorithme (étude du ratio entre une solution d'une instance et une solution de l'instance où on a créé une petite perturbation) pour SLS et Wildcards-SLS (version de SLS où on ajoute un caractère joker) :

— analyse lisse en $1 + o(1)$ pour l'algorithme glouton pour SLS,

— analyse lisse en $1 + o(1)$ pour l'algorithme glouton pour Wildcards-SLS.

Première fois qu'on peut démontrer que la borne inférieure d'approximation d'un problème peut être différente en analyse du pire cas et en analyse lisse.

[57] 2.1.3

Ⓒ ㊿ **David Maier and James A. Storer. A note on the complexity of the superstring problem. Technical Report 233, Dept. of Electrical Engineering and Computer Science, Princeton University, Princeton, 1977.**

Ce papier donne une preuve que SLS_d est **NP-complet** :

— **(alphabet non borné)** même si pour tout $k \geq 8$, les mots de l'instance de départ sont de longueur k. La preuve est basée sur une réduction polynomiale à partir de *node cover problem*.

[58] 6, 6.2, 6.3, 6.3

Ⓜ **Edward M. McCreight. A space-economical suffix tree construction algorithm. *J. ACM*, 23(2) :262–272, 1976.**

Cet article présente un algorithme pour la construction de l'arbre des suffixes en temps linéaire en la taille de l'instance qui est plus économique en espace que [96].

[59] Introduction, 4.2, 4.15, 4.16, 4.17, 4.3.2, 26

Ⓐ Ⓖ Ⓢh **Julián Mestre. Greedy in approximation algorithms. In *Algorithms - ESA 2006, 14th Annual European Symposium, Zurich, Switzerland, September 11-13, 2006, Proceedings*, pages 528–539, 2006.**

Étude des systèmes héréditaires et de la k-extensibilité : preuve qu'un système héréditaire k-extensible donne un algorithme glouton avec un ratio d'approximation de $\frac{1}{k}$ et preuve que l'intersection de k matroide est k-extensible.

[60] 2.1.3

㊿ Ⓒ ㊿ **Martin Middendorf. More on the complexity of common superstring and supersequence problems. *Theor. Comput. Sci.*, 125(2) :205–228, 1994.**

Cet article traite de la complexité de deux problèmes SLS_d et Max-CCP. Il regarde SLS_d sous la contrainte de la taille maximale de l'orbite, c'est à dire le nombre de fois maximum qu'une lettre est vue dans tous les mots de l'instance de départ. On a alors que :

— **(alphabet non borné)** SLS_d est NP-complet même si tous les mots de l'instance de départ sont de longueur 3 et la taille maximale de l'orbite est de 8 ou que tous les mots de l'instance de départ sont de longueur 4 et la taille maximale de l'orbite est de 6.

— **(alphabet borné)** Sur l'alphabet $\{0,1\}$, SLS_d est **NP-complet** si chaque mot a exactement trois 1 (plus précisement de la forme $0^p1O^q10^r1$ ou $10^p1O^q10^r$) et SLS peut être résolu en temps polynomial si chaque mot contient au plus un 1 (i.e. de la forme 0^p10^q) ou si chaque mot est de la forme 10^p1.

Pour Max-CCP, on a que :

— **(alphabet non borné)** Max-CCP est **NP-complet** même si tous les mots de l'instance de départ sont de longueur 3 et la taille maximale de l'orbite est de 8 et il existe un algorithme polynomial qui résout Max-CCP pour les mots de taille 2.

— **(alphabet borné)** Sur l'alphabet $\{0,1\}$, Max-CCP est **NP-complet**.

[61] **2.1.3**

Ⓒ 🏛 **Martin Middendorf. Shortest common superstrings and scheduling with coordinated starting times. *Theor. Comput. Sci.*, 191(1-2) :205–214, 1998.**

Dans cet article, on a une preuve que le problème SLS_d est **NP-complet** même si chaque mot de l'instance de départ est de la forme 10^p10^q.

[62] **2.5.3**

Gaspard Monge. Mémoire sur la théorie des déblais et des remblais. In *Mémoires de l'Académie Royale des Sciences*, pages 666–704, 1781.

Inégalité entre 4 mots a, b, c et d :

$$|ov(a,c)| + |ov(b,a)| \leq |a| + |ov(b,c)|$$

et si $|ov(a,b)| \geq |ov(a,d)|$ et $|ov(a,b)| \geq |ov(c,b)|$:

$$|ov(a,d)| + |ov(c,b)| \leq |ov(a,b)| + |ov(c,d)|$$

[63] **Introduction, 1.11, 2, 2.1.3, 2.3**

Ⓐ 🏛 **Marcin Mucha. Lyndon words and short superstrings. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 958–972, 2013.**

Extension du résultat de [14] en utilisant les mots de Lyndon pour une approximation de SLS grâce à une approximation pour Max-ATSP de $\frac{7-3\alpha}{2}$ à $2 + \frac{11(1-\alpha)}{9-2\alpha}$ (pour une α-approximation pour Max-ATSP). Application de ce résultat à une $\frac{2}{3}$-approximation pour Max-ATSP : $2 + \frac{11}{23}$-approximation pour SLS.

[64] **6.3, 6.3, 6.3**

Ⓐ **Joong Chae Na, Alberto Apostolico, Costas S. Iliopoulos, and Kunsoo Park. Truncated suffix trees and their application to data compression.** ***Theor. Comput. Sci.*****, 1-3(304) :87–101, 2003.**

Cet article présente deux algorithmes de construction d'un arbre des suffixes tronqué.

[65] **Introduction, 2.1.3, 2.1.3**

Ⓐ Ⓒ **Sascha Ott.** ***Graph-Theoretic Concepts in Computer Science : 25th International Workshop, WG'99 Ascona, Switzerland, June 17–19, 1999 Proceedings*****, chapter Lower Bounds for Approximating Shortest Superstrings over an Alphabet of Size 2, pages 55–64. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.**

Étude de SLS et SLS_c :

- borne inférieure d'approximation de $1 + \frac{1}{17245}$ pour SLS sur un alphabet binaire,
- borne inférieure d'approximation de $1+\frac{1}{11216}-\epsilon$ pour SLS_c sur un alphabet binaire,
- SLS est **APX-Difficile** sur un alphabet binaire.

[66] **4.2**

Ⓖ Ⓢh **James G. Oxley.** ***Matroid theory.*** **Oxford University Press, 1992.**

Livre de référence sur les matroides.

[67] **Introduction, 2, 2.1.3, 2.1.3**

Ⓐ **Katarzyna E. Paluch. Better approximation algorithms for maximum asymmetric traveling salesman and shortest superstring.** ***CoRR*****, abs/1401.3670, 2014.**

Étude de SLS et SLS_c :

- présentation d'un algorithme avec un ratio d'approxiation de $\frac{3}{4}$ pour Max-ATSP et SLS_c,
- présentation d'un algorithme avec un ratio d'approxiation de $(2+\frac{11}{30})$ pour SLS.

[68] **Introduction, 2, 2.1.3, 2.3, 30**

Ⓐ **Katarzyna E. Paluch, Khaled M. Elbassioni, and Anke van Zuylen. Simpler approximation of the maximum asymmetric traveling salesman problem. In** ***STACS*****, pages 501–506, 2012.**

Étude de SLS et SLS_c :

— présentation d'un algorithme avec un ratio d'approxiation de $\frac{2}{3}$ pour Max-ATSP et SLS_c,

— présentation d'un algorithme avec un ratio d'approxiation de $(2+\frac{1}{2})$ pour SLS.

[69] **Introduction, 2.3, 2.4.2, 2.5.2, 2.5.2, Conclusion**

Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization : Algorithms and Complexity.* Prentice-Hall, 1982.

Algorithme polynomial pour Cyclic Cover Problem (algorithme hongrois en $0(|V_G|^3)$).

[70] **1.8.2**

Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *J. Comput. Syst. Sci.*, 43(3) :425–440, 1991.

Définition de la classe **Max-SNP**.

[71] **2**

Hannu Peltola, Hans Söderlund, Jorma Tarhio, and Esko Ukkonen. Algorithms for some string matching problems arising in molecular genetics. In *IFIP Congress*, pages 59–64, 1983.

Application de SLS en biologie computationnelle et algorithme en $O(|P|^3)$ pour SCCS.

[72] **2.1**

Hannu Peltola, Hans Söderlund, and Esko Ukkonen. SEQAID : a DNA sequence assembling program based on a mathematical model. *Nucleic Acids Research*, 12(1) :307–321, 1984.

Méthode d'assemblage issue de SLS.

[73] **Introduction, 9.4**

Yu Peng, Henry C. M. Leung, Siu-Ming Yiu, and Francis Y. L. Chin. IDBA - A practical iterative de bruijn graph de novo assembler. In *Research in Computational Molecular Biology, 14th Annual International Conference, RECOMB 2010, Lisbon, Portugal, April 25-28, 2010. Proceedings*, pages 426–440, 2010.

Présentation d'un logiciel pour l'assemblage de génome utilisant le graphe de De Bruijn multi-ordre.

[74] **Introduction**

Pavel A. Pevzner, Haixu Tang, and Michael S. Waterman. A new approach to fragment assembly in DNA sequencing. In *RECOMB*, pages 256–267, 2001.

Application pratique de l'utilisation du graphe de De Bruijn dans l'assemblage de génome.

[75] **2**

Pavel A. Pevzner and Michael S. Waterman. Open combinatorial problems in computational molecular biology. In *ISTCS*, pages 158–173, 1995.

Application de SLS en biologie computationnelle.

[76] **4.1.1**, **4.1.3.1**

Kai Plociennik. A probabilistic PTAS for shortest common superstring. *Theor. Comput. Sci.*, 522 :44–53, 2014.

Analyse en moyenne en $1 + o(1)$ de l'algorithme glouton pour SLS et preuve que si l'algorithme glouton ne devait pas satisfaire la conjecture à 2, cela ne serait que pour un nombre exponentiellement petit d'instance.

[77] **Introduction**

A.S. Rebai and M. Elloumi. Approximation algorithm for the shortest approximate common superstring problem. In *Proc. 12th Word Academy of Science*, pages 302–307, 2006.

Étude de SACS (*Shortest Approximate Common Superstring*) (version de SLS où on cherche une superchaîne approximée) : présentation d'un algorithme avec un ratio d'approxiation de $\frac{1}{2}$ pour SACS en temps $O(|P|^2 \times (l^2 + log(|P|)))$ où l est la longueur de la plus grande chaîne de l'instance.

[78] **4.1.1**

Heidi J. Romero, Carlos A. Brizuela, and Andrei Tchernykh. An experimental comparison of approximation algorithms for the shortest common superstring problem. In *5th Mexican International Conference on Computer Science (ENC 2004), 20-24 September 2004, Colima, Mexico*, pages 27–34, 2004.

Comparaison en pratique de 2 algorithmes de SLS, un algorithme avec un ratio d'approximation de 4 (Mgreedy de [11]) et un algorithme avec un ratio d'approximation de 3 (Tgreedy de [11]) : quotient de $\frac{4}{3}$ dans le pire des cas et de 1 dans le cas moyen indépendamment de la taille de l'instance (dans la pratique une approximation moyenne de 1,014 pour les deux algorithmes).

[79] **2.5.2**

Sartaj Sahni and Teofilo F. Gonzalez. P-complete approximation problems. *J. ACM*, 23(3) :555–565, 1976.

Preuve que Min-ATSP est difficilement approximable sous l'hypothèse **P** $\neq$ **NP**.

[80] **6.3, 6.3**

Marcel H. Schulz, Sebastian Bauer, and Peter N. Robinson. The generalised k-truncated suffix tree for time-and space-efficient searches in multiple DNA or protein sequences. *IJBRA*, 4(1) :81–95, 2008.

On a dans cet article une définition et une construction linéaire en la taille de l'instance de l'arbre des suffixes généralisé et tronqué à k pour un entier k.

[81] **Introduction, 2**

Marvin B. Shapiro. An algorithm for reconstructing protein and rna sequences. *J. ACM*, 14(4) :720–731, October 1967.

Méthode d'assemblage issue de SLS.

[82] **2**

R. Staden. Automation of the computer handling of gel reading data produced by the shotgun method of dna sequencing. *Nucleic Acids Res.*, 10(15) :4731–4751, 1982.

Application de SLS en biologie computationnelle.

[83] **Introduction, 2**

James A. Storer. Np-completeness results concerning data compression. Technical Report 234, Dept. of Electrical Engineering and Computer Science, Princeton University, Princeton, 1977.

Application de SLS en compression de données.

[84] **Introduction, 2, 2.1.3, 4.1.1**

Ⓐ Ⓑ Ⓖ **James A. Storer. *Data Compression : Methods and Theory.* Computer Science Press, 1988.**

Application de SLS en compression de données et conjecture que le ratio d'approximation de l'algorithme glouton pour SLS est de 2.

[85] **Introduction, 2**

Ⓑ **James A. Storer and Thomas G. Szymanski. The macro model for data compression (extended abstract). In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA*, pages 30–39, 1978.**

Application de SLS en compression de données.

[86] **Introduction, 2, 2.1.3, 2.3**

Ⓐ Ⓑ **Z. Sweedyk. A $2\frac{1}{2}$-approximation algorithm for shortest superstring. *SIAM J. Comput.*, 29(3) :954–986, 1999.**

Présentation d'un algorithme avec un ratio d'approxiation de $(2+\frac{1}{2})$ pour SLS.

[87] **Introduction, 2.1.3, 3.3, 3.4.2, 4.1.1, 4.3.2**

Ⓒ Ⓐ Ⓑ Ⓖ **Jorma Tarhio and Esko Ukkonen. A greedy approximation algorithm for constructing shortest common superstrings. *Theor. Comput. Sci.*, 57 :131–145, 1988.**

Étude de SLS et SLS_c :

— présentation d'un algorithme avec un ratio d'approxiation de $\frac{1}{2}$ pour SLS_c,

— conjecture que le ratio d'approximation de l'algorithme glouton pour SLS est de 2,

— lien entre SLS et Max-ATSP sur le graphe des chevauchements,

— implémentation de l'algorithme glouton pour SLS en $O(||P|| \times |P|)$.

[88] **Introduction, 2, 2.1.3, 2.3, 2.3.1, 5, 18, 34**

Ⓐ Ⓑ **Shang-Hua Teng and F. Frances Yao. Approximating shortest superstrings. In *34th Annual Symposium on Foundations of Computer Science*, pages 158–165, 1993.**

Présentation d'un algorithme avec un ratio d'approxiation de $(2+\frac{8}{9})$ pour SLS en temps $O(|P| \times ||P|| + |P|^3)$ (algorithme non issu de l'algorithme glouton mais construit à l'aide d'une solution optimale de SCCS).

[89] **Introduction, 2.1.3**

Ⓒ ㊂ **Vadim G. Timkovskii. Complexity of common subsequence and supersequence problems and related problems. *Cybernetics*, 25(5) :565–580, 1990.**

Le problème SLS_d est résolvable en temps polynomial si la taille maximale de l'orbite est de 2.

[90] **Introduction, 2.5.3, 2.5.4, 3, 3, 3.4, 4.1.1, 4.3.2**

㉾ Ⓐ ㊂ Ⓖ **Jonathan S. Turner. Approximation algorithms for the shortest common superstring problem. *Inf. Comput.*, 83(1) :1–20, 1989.**

Présentation d'un algorithme avec un ratio d'approxiation de $\frac{1}{2}$ pour SLS_c et implémentation de l'algorithme glouton pour SLS en $O(||P|| + |P| \times log(|P|))$ pour un alphabet petit et en $O(||P|| \times log(||P||))$ pour un alphabet quelconque. Il définit, de plus, le graphe des préfixes.

[91] **Introduction, 2.1, 2.1.1, 4.1.1**

㊂ Ⓖ **Esko Ukkonen. A linear-time algorithm for finding approximate shortest common superstrings. *Algorithmica*, 5(3) :313–323, 1990.**

Implémentation de l'algorithme glouton pour SLS en $O(||P||)$ pour les petits alphabets et en $O(||P|| * min(log(|P|), log(|\Sigma|)))$ pour les alphabets de taille arbitraire (où Σ est l'alphabet de l'instance P).

[92] **6.2**

Ⓐ **Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3) :249–260, 1995.**

Cet article présente un algorithme online pour la construction de l'arbre des suffixes en temps linéaire en la taille de l'instance.

[93] **2.1, 2.16, 2.1.3**

Ⓐ Ⓒ ㊂ **Virginia Vassilevska. Explicit inapproximability bounds for the shortest superstring problem. In *Mathematical Foundations of Computer Science 2005, 30th International Symposium, MFCS 2005, Gdansk, Poland, August 29 - September 2, 2005, Proceedings*, pages 793–800, 2005.**

Étude de SLS et SLS_c :

— preuve que si on a une α-approximation pour un alphabet binaire sur SLS, on a une α-approximation pour un alphabet quelconque sur SLS,

— borne inférieure d'approximation de $1 + \frac{1}{1216}$ pour SLS,

— borne inférieure d'approximation de $1 + \frac{1}{1071}$ pour SLS_c.

[94] **2**

M.S. Waterman. *Introduction to Computational Biology : Maps, Sequences, and Genoms (Interdisciplinary Statistics)*. Chapman and Hall, 1995.

Application de SLS en biologie computationnelle.

[95] **4.1.3.2, 4.12, 51**

Maik Weinard and Georg Schnitger. On the greedy superstring conjecture. *SIAM J. Discrete Math.*, 20(2) :502–522, 2006.

Étude du cas où greedy détermine un "linear greedy order", c'est-à-dire que chaque étape de greedy étend à gauche ou à droite la même chaine (à chaque étape de greedy, il n'y a qu'une unique chaîne issue de la fusion) :

— soient C_1 et C_2 deux cyclic cover,

$$|Greedy_{\text{cyclic cover}}| \leq ||C_1|| + ||C_2||$$

— dans le cas du "linear greedy order", $|Greedy_{\text{linéaire}}| \leq |opt_{\text{linéaire}}| + |opt_{\text{cyclic cover}}|$

— triple inégalité (amélioration de l'inégalité de Monge).

[96] **6.2, 58**

Peter Weiner. Linear pattern matching algorithms. In *14th Annual Symposium on Switching and Automata Theory, Iowa City, Iowa, USA, October 15-17, 1973*, pages 1–11, 1973.

Cet article présente l'arbre des suffixes d'un mot et donne un algorithme linéaire en la taille de l'instance pour le construire.

[97] **4.2**

Hassler Whitney. On the abstract properties of linear dependence. In *The Johns Hopkins University Press*, volume 57, pages 509–533, 1935.

Introduction des matroides.

[98] **Introduction, 2**

Robert Wilensky. *Planning and Understanding : A Computational Approach to Human Reasoning*. Addison Wesley Longman Publishing Co, 1983.

Application de SLS au AI planning.

[99] 4.1.1

En-Hui Yang and Zhen Zhang. The shortest common superstring problem : Average case analysis for both exact and approximate matching. *IEEE Transactions on Information Theory*, 45(6) :1867–1886, 1999.

Étude des différents algorithmes gloutons pour SLS et SACS asymptotiquement.

[100] 4.1.1

Assaf Zaritsky and Moshe Sipper. The preservation of favored building blocks in the struggle for fitness : the puzzle algorithm. *IEEE Trans. Evolutionary Computation*, 8(5) :443–455, 2004.

Cet article présente deux algorithmes de coévolution, "puzzle" et "co-puzzle" qui sont comparés expérimentalement à l'algorithme glouton pour SLS et à un algorithme génétique standard.