Alignement, séquence consensus, recherche de similarités : complexité et approximabilité

François Nicolas

Académie de Montpellier

Université Montpellier II

— Sciences et Technique du Languedoc —

THÈSE

présentée à l'Université des Sciences et Techniques du Languedoc
pour obtenir le diplôme de doctorat

Spécialité	: **INFORMATIQUE**
Formation Doctorale	: **Informatique**
École Doctorale	: **Information, Structures, Systèmes**

Alignement, séquence consensus, recherche de similarités : complexité et approximabilité

par

François Nicolas

Soutenue le 13 décembre 2005 devant le jury composé de :

Irena Rusu, Professeur, Université de Nantes, LINA	Rapportrice
Christian Choffrut, Professeur, Université Paris VII, LIAFA	Rapporteur
Jean-Claude König, Professeur, Université Montpellier II, LIRMM	Examinateur
Olivier Gascuel, Directeur de Recherche CNRS, LIRMM	Directeur de thèse
Éric Rivals, Chargé de Recherche CNRS, LIRMM	Co-directeur
Gregory Kucherov, Directeur de Recherche CNRS, LIFL (Lille)	Examinateur
Julien Cassaigne, Chargé de Recherche CNRS, IML (Marseille)	Invité

Table des matières

0 Introduction **5**
- 0.1 Problèmes étudiés . . . 5
  - 0.1.1 Extraction de motifs communs . . . 5
  - 0.1.2 Recherche de similarités locales . . . 5
  - 0.1.3 Points de vue théoriques adoptés . . . 6
    - 0.1.3.1 NP-complétude . . . 6
    - 0.1.3.2 Approximabilité . . . 7
    - 0.1.3.3 Complexité paramétrique . . . 7

1 Définitions et notations **8**
- 1.1 Mathématiques générales . . . 8
  - 1.1.1 Métriques . . . 8
  - 1.1.2 Hypergraphes et graphes . . . 9
  - 1.1.3 Mots et langages . . . 9
    - 1.1.3.1 Concaténation . . . 9
    - 1.1.3.2 Préfixe, suffixe, sous-chaîne et lettres d'un mot . . . 9
    - 1.1.3.3 Distance de Hamming . . . 10
    - 1.1.3.4 Langages . . . 10
- 1.2 Complexité algorithmique . . . 10
  - 1.2.1 Algorithmes et temps de calcul . . . 10
  - 1.2.2 Problèmes de décision . . . 11
    - 1.2.2.1 Instances et algorithmes . . . 11
    - 1.2.2.2 Restriction . . . 11
    - 1.2.2.3 M-réduction . . . 11
  - 1.2.3 La théorie de la NP-complétude . . . 12
    - 1.2.3.1 Les classes P et NP . . . 12
    - 1.2.3.2 Réductions de Karp et problèmes NP-complets . . . 12
  - 1.2.4 Complexité paramétrique . . . 13
    - 1.2.4.1 Problèmes de décision paramétrés et la classe FPT . . . 13
    - 1.2.4.2 Intraitabilité des problèmes de décision paramétrés . . . 14
  - 1.2.5 Optimisation combinatoire . . . 15
    - 1.2.5.1 Problèmes d'optimisation . . . 15
    - 1.2.5.2 Résolution exacte des problèmes d'optimisation . . . 16
    - 1.2.5.3 Résolution approchée des problèmes d'optimisation . . . 17
- 1.3 Le problème de l'indépendant maximum . . . 18
  - 1.3.1 Définition . . . 18
  - 1.3.2 Complexité . . . 18
  - 1.3.3 Approximabilité . . . 18
- 1.4 Le problème de la plus longue sous-séquence commune . . . 19
  - 1.4.1 Définition . . . 19
  - 1.4.2 Algorithmes exacts . . . 20
  - 1.4.3 Complexité paramétrique . . . 20

1.4.4 Approximabilité . . . 20
1.5 Alignement . . . 20
1.5.1 Définitions . . . 21
1.5.2 Distance d'édition . . . 21
1.5.2.1 Définition . . . 22
1.5.2.2 Opérations d'édition . . . 23
1.5.3 Distance d'édition et plus longue sous-séquence commune . . . 23
1.5.4 Calcul de la distance d'édition . . . 24

2 Complexité des problèmes du centre et de la médiane pour la distance d'édition 25
2.1 Introduction . . . 25
2.2 Position des problèmes et exemples . . . 26
2.2.1 Médiane . . . 26
2.2.2 Centre . . . 29
2.2.3 Digression : médiane et centre en géométrie . . . 30
2.3 Distance de Hamming . . . 33
2.3.1 Médiane pour la distance de Hamming . . . 33
2.3.2 Centre pour la distance de Hamming . . . 33
2.3.2.1 Complexité paramétrique . . . 33
2.3.2.2 Approximabilité . . . 34
2.4 Problèmes d'alignement multiple . . . 34
2.4.1 Un algorithme général . . . 34
2.4.1.1 Position du problème . . . 34
2.4.1.2 Description de l'algorithme . . . 35
2.4.2 Somme de toutes les paires . . . 36
2.4.3 Alignement phylogénétique . . . 37
2.4.3.1 Arbre phylogénétique . . . 37
2.4.3.2 Description des problèmes . . . 37
2.4.3.3 Complexité des problèmes d'alignement phylogénétique . . . 38
2.4.4 Médiane et centre pour la distance d'édition . . . 38
2.4.4.1 Le problème de la médiane pour la distance d'édition . . . 38
2.4.4.2 Le problème du centre pour la distance d'édition . . . 39
2.5 Intraitabilité des problèmes du centre et de la médiane pour la distance de Levenshtein . . . 39
2.5.1 Intraitabilité du problème du centre pour la distance de Levenshtein . . . 40
2.5.2 Intraitabilité du problème de la médiane pour la distance de Levenshtein . . . 43
2.6 Intraitabilité du problème du centre pour une large classe de distances d'édition . . . 47
2.6.1 Le problème de la sous-séquence commune pondérée . . . 47
2.6.2 Le problème du centre pour la distance d'édition . . . 49
2.7 Un algorithme exact pour le problème du centre pour la distance d'édition . . . 52
2.7.1 Résultats préliminaires . . . 53
2.7.2 Description de l'algorithme . . . 54
2.7.2.1 Table de programmation dynamique . . . 54
2.7.2.2 Caractérisation inductive de la table de programmation dynamique . . . 55
2.7.2.3 Calcul de la table de programmation dynamique . . . 56
2.8 Conclusion . . . 57
2.8.1 Récapitulatif des résultats connus sur la complexité des problèmes d'alignement multiple 57
2.8.2 Questions ouvertes . . . 58

3 Complexité du problème de la plus longue sous-séquence commune pour les mots non orientés et/ou circulaires 59
3.1 Introduction . . . 59
3.1.1 Application des alignements de mots circulaires et/ou non orientés . . . 59
3.1.1.1 En bio-informatique . . . 59
3.1.1.2 En reconnaissance des formes . . . 59

3.1.2 Complexité du problème de l'alignement multiple pour les mots circulaires et/ou non orientés . . . 60
3.1.2.1 État de l'art sommaire . . . 60
3.1.2.2 Description sommaire de notre contribution . . . 60
3.2 Définitions et notations . . . 60
3.2.1 Autour des mots circulaires et/ou non orientés . . . 60
3.2.2 Position des problèmes . . . 62
3.3 Résultats connus et notre contribution . . . 64
3.3.1 Alignement de deux mots circulaires et/ou non orientés . . . 64
3.3.1.1 Le problème de l'alignement circulaire de score minimum . . . 64
3.3.1.2 Élimination des problèmes d'orientation . . . 65
3.3.1.3 Sous-séquences circulaires et/ou non orientées communes à deux mots . . . 65
3.3.2 Le problème de la plus courte super-séquence commune et sa variante pour les mots circulaires . . . 65
3.3.2.1 Le problème de la plus courte super-séquence commune dans sa forme classique 65
3.3.2.2 Le problème de la plus courte super-séquence commune pour les mots circulaires 66
3.3.3 Notre contribution . . . 67
3.4 Résolution exacte de LCUS, LCCS et LCUCS . . . 67
3.4.1 Résultats positifs . . . 67
3.4.2 Résultats négatifs . . . 68
3.5 Algorithmes d'approximation pour LCS, LCUS, LCCS et LCUCS . . . 71
3.6 Inapproximabilité de MISH, LCS, LCUS, LCCS et LCUCS . . . 73
3.6.1 Difficulté du problème MISH . . . 74
3.6.1.1 Gestion des "effets de bord" . . . 74
3.6.1.2 Résultats concernant MISH . . . 75
3.6.2 Inapproximabilité de LCS, LCUS, LCCS et LCUCS . . . 77
3.6.2.1 Mots croissants, mots décroissants . . . 77
3.6.2.2 Résultats concernant LCS et LCUS . . . 79
3.6.2.3 Résultats concernant LCCS et LCUCS . . . 81
3.6.2.4 Conclusion . . . 83
3.7 Questions ouvertes . . . 84
3.7.1 Algorithmes polynomiaux . . . 84
3.7.2 Complexité paramétrique . . . 84
3.7.2.1 r-MISH pour $r \geq 3$ . . . 84
3.7.2.2 LCUS, LCCS et LCUCS . . . 84
3.7.3 Approximabilité . . . 85
3.7.3.1 Existence d'un P.T.A.S. pour LCS, LCUS, LCCS et LCUCS lorsque l'alphabet d'entrée est borné . . . 85
3.7.3.2 Variations sur une conjecture de Jiang et Li . . . 85

4 Sur la difficulté de la conception de bonnes graines 86
4.1 Introduction . . . 86
4.1.1 Importance pratique et théorique de la recherche d'homologies . . . 86
4.1.1.1 Utilité pratique des homologies locales . . . 86
4.1.1.2 Importance théorique de la recherche de motifs approchés . . . 86
4.1.2 Définitions formelles et algorithmes . . . 87
4.1.2.1 Algorithmes identifiant des homologies . . . 87
4.1.2.2 Algorithmes de recherche de motifs approchés . . . 87
4.1.3 Filtration . . . 88
4.1.3.1 Principe général de la filtration . . . 88
4.1.3.2 Filtration et recherche d'homologies locales . . . 88
4.1.3.3 Sensibilité et sélectivité . . . 89
4.2 Position des problèmes . . . 90

4.2.1 Graine, similarité, détection . . . 90
4.2.1.1 Graine, similarité, détection . . . 91
4.2.1.2 Observations en vrac . . . 91
4.2.1.3 Touche à gabarit et filtration . . . 91
4.2.1.4 Rôle des similarités . . . 92
4.2.1.5 Rôle du concept de détection . . . 92
4.2.1.6 Utilité des graines à trous . . . 92
4.2.2 Description de nos résultats . . . 93
4.2.2.1 Le problème Non Detection . . . 93
4.2.2.2 Le problème Maximum Weight Lossless Seed (MWLS) . . . 94
4.2.2.3 Le problème Region Specific Optimal Seed(s) (RSOS(s)) . . . 94
4.3 Difficulté du problème Non Detection . . . 95
4.4 Inapproximabilité de MWLS . . . 102
4.4.1 Gadgets "recyclables" . . . 102
4.4.1.1 Règle de Golomb . . . 102
4.4.1.2 Encodage d'ensembles à l'aide de graines et de similarités . . . 104
4.4.2 Gadget spécifique à notre réduction de 2-MISH à MWLS . . . 106
4.5 Inapproximabilité de RSOS . . . 107
4.5.1 Le problème Maximum k-Cover . . . 108
4.5.1.1 Approximabilité de Max k-Cover . . . 108
4.5.1.2 Une formulation alternative de Max k-Cover : Max k-Hitting Set . . . 108
4.5.2 Preuve de notre résultat principal concernant RSOS . . . 110
4.5.2.1 Construction d'une instance de RSOS à partir d'une instance de Max k-HS 110
4.5.2.2 Correction de notre réduction . . . 111
4.6 Questions ouvertes . . . 113
4.6.1 Problèmes de pavage . . . 113
4.6.2 Maximisation du poids . . . 113
4.6.3 Approximation de RSOS . . . 113

A Preuve du théorème 2.6 page 56 **121**
A.1 Montrons que la condition est suffisante. . . . 122
A.1.1 Supposons qu'il existe un $j \in I$ vérifiant (E_j) et montrons que $B[\mathbf{p}][\mathbf{d}] = \texttt{oui}$. . . . 122
A.1.2 Supposons qu'il existe $a \in \Sigma$ et $J \subseteq I$ vérifiant $(F_{a,J})$ et montrons que $B[\mathbf{p}][\mathbf{d}] = \texttt{oui}$. 122
A.2 Montrons que la condition est nécessaire. . . . 123
A.2.1 Supposons qu'il existe un $j \in I$ tel que $\alpha_j = \varepsilon$ et montrons (E_j). . . . 123
A.2.2 Supposons que, pour tout $i \in I$, on ait $\alpha_i \neq \varepsilon$ et montrons qu'il existe $a \in \Sigma$ et une partie non vide $J \subseteq I$ satisfaisant $(F_{a,J})$. . . . 123

B Glossaire des problèmes **125**

À Odessos, un négociant revenu d'un voyage de plusieurs années me[1] fit cadeau d'une pierre verte, semi-transparente, substance sacrée, paraît-il, dans un immense royaume dont il avait au moins côtoyé les bords, et dont cet homme épaissement enfermé dans son profit n'avait remarqué ni les mœurs ni les dieux. Cette gemme bizarre fit sur moi le même effet qu'une pierre tombée du ciel, météore d'un autre monde. Nous connaissons encore assez mal la configuration de la terre. À cette ignorance, je ne comprends pas qu'on se résigne. J'envie ceux qui réussiront à faire le tour des deux cents cinquante mille stades grecs si bien calculés par Ératosthène, et dont le parcours nous ramènerait à notre point de départ. Je m'imaginais prenant la simple décision de continuer à aller de l'avant, sur la piste qui déjà remplaçait nos routes. Je jouais avec cette idée...Être seul, sans biens, sans prestiges, sans aucun des bénéfices d'une culture, s'exposer au milieu d'hommes neufs et parmi des hasards vierges...Il va de soi que ce n'était qu'un rêve, et le plus bref de tous. (...) Néanmoins, ce rêve monstrueux, dont eussent frémi nos ancêtres, sagement confinés dans leur terre du Latium, je l'ai fait, et de l'avoir hébergé un instant me rend à jamais différent d'eux.

Marguerite Yourcenar (*Mémoires d'Hadrien*)

[1] Le narrateur est Publius Aelius Hadrianus, empereur romain (76–138).

> Le véritable savant met vingt bonnes années en moyenne à effectuer la grande découverte, celle qui consiste à se convaincre que le délire des uns ne fait pas du tout le bonheur des autres et que chacun ici-bas se trouve indisposé par la marotte du voisin. (...) Et je ne pouvais plus l'empêcher de me donner, Parapine, cent et mille haineux détails sur le métier bouffon de chercheur auquel il était bien obligé pour avoir à bouffer de s'astreindre, haine plus précise, plus scientifique vraiment, que celles qui émanent des autres hommes placés dans des conditions similaires dans les bureaux ou magasins. (...) J'ai vu le monde devenir en moins de quelques ans une véritable pétaudière de publications universelles et saugrenues sur ce même sujet rabâché. Je me résigne pour y garder ma place et la défendre certes tant bien que mal, à produire et reproduire mon même petit article d'un congrès, d'une revue à l'autre, auquel je fais simplement subir vers la fin de chaque saison, quelques subtiles et anodines modifications, bien accessoires...(...) Non, j'aime autant vous l'avouer, je ne me sens plus de force à tracasser davantage, ce que je cherche pour achever mon existence, c'est un petit coin de recherches bien tranquilles, qui ne me vaillent plus ni ennemis, ni élèves, mais cette médiocre notoriété sans jalousie dont je me contente et dont j'ai grand besoin.

Louis-Ferdinand Céline (*Voyage au bout de la nuit*).

This is my body. And I can do whatever I want to it. I can

- push it,
- study it,
- tweak it,
- listen to it.

Everybody wants to know what I'm on. What am I on ? I'm on my bike, busting my ass, six hours a day.

Lance Armstrong.

Remerciements

Chapitre 0

Introduction

Les progrès de la biologie moléculaire, comme l'automatisation du séquençage de l'ADN, ont provoqué une explosion de la quantité de données disponibles, en particulier sous forme de séquences. Pour traiter systématiquement ces données, on doit donc recourir à des calculateurs puissants ainsi qu'à des algorithmes performants. On tombe ainsi naturellement dans le champ de compétence de l'Informatique.

0.1 Problèmes étudiés

Dans ce mémoire, nous étudions la complexité algorithmique de plusieurs problèmes de comparaison de séquences biologiques.

Ces problèmes sont de deux sortes : extraction de motifs communs et recherche de similarités.

0.1.1 Extraction de motifs communs

Dans les chapitres 2 et 3, nous considérons plusieurs formes du problème de l'extraction des motifs communs à un ensemble de séquences donné. Les motifs communs permettent, en pratique, de classifier les protéines grâce à leur structure primaire. Les problèmes que nous considérons appartiennent tous à la classe des problèmes d'*alignement multiple*. Le concept d'alignement multiple est défini à la section 1.5. Un exemple concret d'alignement multiple est présenté dans la figure 1 p. 6.

Médiane et centre. Dans le chapitre 2, nous considérons le problème de la médiane (resp. du centre) pour la distance d'édition : on recherche un mot minimisant la somme (resp. le maximum) des distances d'édition le séparant de chacun des mots d'un ensemble donné. La distance d'édition est définie dans la section 1.5.2. Lorsqu'elle est correctement pondérée, elle mesure de manière raisonnable la quantité d'évolution entre les séquences.

Nous améliorons les résultats déjà obtenus pour ces deux problèmes.

Plus longue sous-séquence commune pour les mots circulaires et/ou non orientés. Dans le chapitre 3, nous étudions les variantes pour les mots circulaires et/ou non orientés du problème classique de la *plus longue sous-séquence commune* (voir section 1.4). Nous généralisons pour chacune de ces variantes les résultats obtenus pour le problème d'origine. Cette étude est motivée, d'une part, par l'existence de séquences biologiques circulaires et/ou dont on ne sait pas déterminer l'orientation, et qu'il est nécessaire de comparer. D'autre part, dans le cadre de la reconnaissance des formes, il est souvent pertinent comparer des séquences circulaires modélisant des contours de formes planaires.

0.1.2 Recherche de similarités locales

Dans le chapitre 4, nous considérons le problème de la recherche des homologies locales entre deux séquences données. Cette tâche est d'importance cruciale pour les biologistes, comme en témoigne l'utilisation

```
HPVYRGVR-KRNWGKWVSEIREP--RKKSRIWLGTFPSPEMAARAHDVAALSIKGASAILNFPDLAGS
HPVYRGVR-MRSWGKWVSEIRQP--RKKTRIWLGTFVTADMAARAHDVAALTIKGSSAVLNFPELASL
HPIYRGVR-QRNSGKWVSEVREP--NKKTRIWLGTFQTAEMAARAHDVAALALRGRSACLNFADSAWR
LKPYKGIR-MRKWGKWVAEIREP--NKRSRIWLGSYATPEAAARAYDTAVFYLRGPSARLNFPELLAG
QGKYRGVR-RRPWGKYAAEIRDSR-KHGERVWLGTFDTAEDAARAYDRAAYSMRGKAAILNFPHEYNM
GKKYRGVR-RRPWGKYAAEIRDSA-RHGARVWLGTFETAEEAALAYDRAAFRMRGAKALLNFPSEIVN
EKSFRGVR-RRPWGKFAAEIRDST-RNGVRVWLGTFDSPEAAALAYDQAAFLMRGTSAILNFPVETVQ
KRHYRGVR-MRPWGKFAAEIRDPT-RRGTRVWLGTFETAIEAARAYDKEAFRLRGSKAILNFPLEVDK
-RHYRGVR-QRPWGKFAAEIRDPA-KNGARVWLGTYETAEEAAIAYDKAAYRMRGSKAHLNFPHRIGL
PKKYRGVR-QRPWGKWAAEIRDP--HKATRVWLGTFETAEAAARAYDAAALRFRGSKAKLNFPENVGT
EIRYRGVR-KRPWGRYAAEIRDP--GKKTRVWLGTFDTAEEAARAYDTAARDFRGAKAKTNFPTFLEL
TKLYRGVR-QRHWGKWVAEIRLP--RNRTRLWLGTFDTAEEAALAYDKAAYKLRGDFARLNFPNLRHN
RKKFRGVR-QRPWGRWAAEIRDP--TRGKRVWLGTYDTPEEAAVVYDKAAVKLKGPDAVTNFPVSTTA
KTKFVGVR-QRPSGKWVAEIKDT--TQKIRMWLGTFETAEEAARAYDEAACLLRGSNTRTNFANHFPN
ASIYRGVTRHHQHGRWQARIGR--VAGNKDLYLGTFGTQEEAAEAYDVAAIKFRGTNAVTNFDITRYD
SSKYKGVV-PQPNGRWGAQIYE----KHQRVWLGTFNEQEEAARSYDIAACRFRGRDAVVNFKNVLED
```

FIG. 1 – Un exemple d'alignement de 16 fragments de séquences protéiques trouvé sur le site de Pfam (numéro d'accession : `PF00847`, identificateur : `AP2-domain`). Chaque lettre majuscule représente un acide aminé et chaque `-` représente un espace inséré.

intensive des logiciels conçus pour la traiter, tels FASTA [95] ou BLAST [2] pour ne citer que les plus connus. Les homologies permettent de déduire les propriétés des molécules nouvellement séquencées à partir des propriétés des séquences déjà annotées. En pratique, les logiciels de recherche d'homologies fonctionnent sur un même principe : la *filtration*.

1. On commence par repérer, sur les séquences prises en entrée, les paires de régions qui partagent au moins un nombre fixé de motifs communs de formes fixées. Ces formes sont appelées des *graines*.
2. On vérifie de manière précise la similarité de chacune des paires de régions repérées. On retourne seulement celles qui sont suffisamment longues et semblables pour mériter d'être reportées.

Nous étudions dans ce mémoire la complexité de la détermination de bonnes graines, c'est-à-dire de graines rendant le filtre associé sensible et sélectif.

0.1.3 Points de vue théoriques adoptés

Ce mémoire à pour objectif d'évaluer les ressources en temps de calcul nécessitées par la résolution des problèmes évoqués précédemment. Nous nous plaçons du point de vue de chacune des trois principales théories de la complexité algorithmique (déterministe) :

- la NP-*complétude* (voir section 1.2.3),
- l'*approximabilité* (voir section 1.2.5) et
- la *complexité paramétrique* (voir section 1.2.4).

Introduisons sommairement ces théories.

0.1.3.1 NP-complétude

La NP-complétude est la plus ancienne et la plus rudimentaire des théories que nous considérons. Elle a été bâtie au début des années 1970 par Cook [23] et Karp [59]. Elle permet de différencier les problèmes admettant un algorithme exact et de complexité polynomiale dans le pire des cas, des problèmes dits NP-*difficiles*. Dans l'état actuel des connaissances, on peut seulement conjecturer que les problèmes NP-difficile n'admettent pas d'algorithmes polynomiaux : c'est la fameuse conjecture $P \neq NP$. Néanmoins, même si l'on ne sait pas prouver cette conjecture, tout porte à croire qu'aucun problème NP-difficile ne peut admettre d'algorithme sous-exponentiel.

Le recours à une conjecture est général en complexité algorithmique. Depuis la preuve par Turing de l'indécidabilité du problème de l'arrêt [112] à l'aide d'un argument de diagonalisation, on a pas prouvé beaucoup de résultats négatifs.

0.1.3.2 Approximabilité

La théorie de l'approximabilité concerne seulement les problèmes d'*optimisation*. Pour les problèmes d'optimisation pour lesquels il est difficile de trouver une solution optimale en temps raisonnable, il est parfois possible de trouver facilement une solution seulement légèrement sous-optimale. Tout comme une heuristique, un algorithme d'approximation est censé retourner une solution "pas trop mauvaise" en un temps "pas trop long". Mais, contrairement à une heuristique, il doit être possible de minorer la qualité de la solution retournée et de majorer le temps de calcul dans le pire des cas.

Les premiers algorithmes d'approximation polynomiaux pour des problèmes d'optimisation NP-difficiles datent du milieu des années 1970 [57]. Il est à noter que la théorie de l'approximabilité a été révolutionnée au début des années 1990 par le *"théorème PCP"* qui a permis de démontrer de nombreux résultats négatifs. On trouvera dans [28, Chapitre 2] un historique détaillé.

0.1.3.3 Complexité paramétrique

La théorie de la complexité paramétrique a été développée à partir du début des années 1990 [29]. Son but est de distinguer, parmi les problèmes NP-difficiles, ceux qui admettent de "bons" algorithmes exponentiels exacts. Précisons cette notion.

On cherche à construire un algorithme pour un certain problème NP-difficile. On considère un *paramètre* p à valeurs entières de ce problème : p est une fonction associant un entier naturel $p(x)$ à chaque instance x. On suppose que, pour les instances x rencontrées en pratique, $p(x)$ reste petit ou, du moins, que $p(x)$ croît faiblement devant la taille $|x|$ de x. Alors, un algorithme exponentiel prenant un temps $\Theta\Big(|x|^{p(x)}\Big)$ exploite moins bien les propriétés de p qu'un algorithme, également exponentiel mais de complexité $O\big(2^{p(x)}\,|x|\big)$. Grossièrement, la théorie de la complexité paramétrique a pour objectif de distinguer ces deux cas.

Chapitre 1

Définitions et notations

1.1 Mathématiques générales

La terminologie et les notations mathématiques utilisées tout au long de ce mémoire sont généralement standard. On préférera le terme *application* à celui de *fonction totale*. On note $\setminus$ la *différence ensembliste* : $X \setminus Y = \{x \in X : x \notin Y\}$ pour tous ensembles X, Y. Pour tout ensemble d'ensembles $\mathcal{C}$, la *réunion* des ensembles appartenant à $\mathcal{C}$ est noté indifféremment $\bigcup \mathcal{C}$ ou $\bigcup_{X \in \mathcal{C}} X$. Pour tout ensemble fini X, on note $\#X$ le *cardinal* de l'ensemble X.

$\mathbb{Z}$ désigne l'ensemble des *entiers relatifs* et $\mathbb{N}$ désigne l'ensemble des *entiers naturels*. Pour tous a, $b \in \mathbb{Z}$, on note $[a, b] := \{x \in \mathbb{Z} : a \leq x \leq b\}$. Un *segment* S de $\mathbb{Z}$ est une partie de $\mathbb{Z}$ telle que $S = [\min S, \max S]$. Pour tout $X \subseteq \mathbb{Z}$ et tout $t \in \mathbb{Z}$, on note $X + t := \{x + t : x \in X\}$.

Remarque 1.1 *Soient A, B, C et D des réels. Dans ce mémoire, une écriture du genre :*

$$\begin{aligned} A &\geq B && (1.1)\\ &= C && (1.2)\\ &\geq D && (1.3)\\ &> E && (1.4) \end{aligned}$$

représente la conjonction des quatre assertions

$$A \geq B,\ B = C,\ C \geq D \text{ et } D > E.$$

Elle les affecte de plus respectivement des numéros (1.1), (1.2), (1.3) *et* (1.4). *Elle* n'indique pas *que A est égal à C.*

1.1.1 Métriques

Soient un ensemble quelconque U et une application d de $U \times U$ vers l'ensemble des nombres réels.
On dit que :
- d est *symétrique* lorsque, pour tous x, $y \in U$, on a $d(x, y) = d(y, x)$;
- d satisfait l'*inégalité triangulaire* lorsque, pour tous x, y, $z \in U$, on a $d(x, z) \leq d(x, y) + d(y, z)$;
- d est *séparante* lorsque, pour tous x, $y \in U$ tels que $d(x, y) = 0$, on a $x = y$.

On dit que d est une *semi-métrique* sur U lorsque : d satisfait l'inégalité triangulaire, d est symétrique et $d(x, x) = 0$ pour tout $x \in U$. On dit que d est une *métrique* sur U lorsque d est une semi-métrique sur U qui est séparante.

Remarque 1.2 *Si d est symétrique et satisfait l'inégalité triangulaire alors d ne peut prendre que des valeurs positives ou nulles. En effet, pour tous x, $y \in U$, on a $d(x, x) \leq d(x, x) + d(x, x)$, d'où $0 \leq d(x, x) \leq d(x, y) + d(y, x) = 2 \cdot d(x, y)$, et par suite, $d(x, y)$ est positif ou nul. Ainsi, les semi-métriques et les métriques ne prennent que des valeurs positives.*

Exemple 1.1 *La distance euclidienne usuelle entre les points du plan ou de l'espace est une métrique. La* distance de Levenshtein *[70], décrite dans la section 1.5.2, est une métrique sur les mots.*

La *métrique discrète* sur U est l'application qui, à tout $(x, y) \in U \times U$, associe 1 si $x \neq y$ et 0 sinon. La *distance "tout ou rien"* sur U est l'application qui, à tout $(x, y) \in U \times U$, associe ∞ si $x \neq y$ et 0 sinon.

1.1.2 Hypergraphes et graphes

Un *hypergraphe* (*sur* V) est un couple $H = (V, \mathcal{E})$ où V est un ensemble fini et où $\mathcal{E}$ est un ensemble de sous-ensembles de V. On appelle *sommets* (resp. *hyperarêtes*) de H les éléments de V (resp. de $\mathcal{E}$). On note $|H| := \#V$ le nombre de sommets de H.

Exemple 1.2 *Soient* $E_1 := \{1, 4, 6\}$, $E_2 := \{2, 5\}$, $E_3 := \{1, 2, 3, 5\}$, $E_4 := \{1, 2, 4, 6\}$, $E_5 := \{3, 4\}$, $V := \{1, 2, 3, 4, 5, 6, 7\}$, $\mathcal{E} := \{E_1, E_2, E_3, E_4, E_5\}$ *et* $H := (V, \mathcal{E})$. *H est un hypergraphe dont les sommets sont* 1, 2, 3, 4, 5, 6, 7 *et dont les hyperarêtes sont* E_1, E_2, E_3, E_4, E_5.

Soit un entier $r \geq 2$. Un hypergraphe *r-uniforme* est un hypergraphe dont les hyperarêtes sont toutes de cardinal r. Un hypergraphe 2-uniforme est appelé un *graphe*. Les hyperarêtes d'un graphe sont appelées *arêtes*.

1.1.3 Mots et langages

Nous nous inspirons librement de la terminologie et des notations de Lothaire [76, 77].

Un *alphabet* est un ensemble (fini ou infini) de symboles ou *lettres*. On suppose que la classe des symboles de l'univers est un ensemble infini dénombrable. Dans toute la suite, la lettre Σ désignera un alphabet. Un *mot* w (sur l'alphabet Σ) est une suite de symboles (pris dans Σ) de longueur finie $|w|$. On note $\Sigma^\star$ l'ensemble des mots sur Σ et, pour tout entier $n \geq 0$, on note Σ^n l'ensemble des mots de longueur n sur Σ. La suite de longueur 0 est appelée *mot vide* et notée ε ou parfois $-$ dans les alignements (voir section 1.5).

Exemple 1.3 $w :=$ `allezlOM` *est un mot de longueur* $|w| = 8$ *sur l'alphabet* $\{$`a`, `b`, `l`, `e`, `z`, `O`, `M`$\}$.

1.1.3.1 Concaténation

Étant donnés deux mots x et y, on note xy la *concaténation* des mots x et y, c'est-à-dire le mot obtenu en écrivant x et y bout à bout dans cet ordre. L'opération de concaténation est associative, ce qui permet, par exemple, d'écrire sans ambiguïté xyz à la place de $(xy)z = x(yz)$ pour tous mots x, y, z.

Exemple 1.4 *Pour* $x :=$ `abra` *et* $y :=$ `cad`, *on a* $xyx =$ `abracadabra`.

Pour tout entier $n \geq 1$ et tout mot x, la concaténation de n copies de x est notée x^n ; comme on convient que $x^0 := \varepsilon$, on a $x^{n+1} = x(x^n)$ pour tout $n \in \mathbb{N}$.

Exemple 1.5 `bbgggaaayyerrrrrrr` $=$ `b`^{2}`é`^{0}`g`^{3}`a`^{3}`y`^{2}`e`^{1}`r`^{7}.

1.1.3.2 Préfixe, suffixe, sous-chaîne et lettres d'un mot

Soient deux mots w et x.

On dit que x est *préfixe* (resp. *suffixe*) de w lorsqu'il existe un mot u tel que $w = xu$ (resp. tel que $w = ux$). On dit que x est une *sous-chaîne* de w lorsqu'il existe deux mots u, v tels que $w = uxv$.

Pour tout $i \in [1, |w|]$, on note $w[i]$ la i-ème lettre de w, c'est-à-dire la lettre de w située à la position $i - 1$: $w = w[1]w[2]\dots w[|w|]$. Pour chaque lettre a, on note $|w|_a := \#\{i \in [1, |w|] : w[i] = a\}$ le *nombre d'occurrences* de la lettre a dans w. L'*image miroir* de w est le mot $\tilde{w} := w[|w|]\dots w[2]w[1]$. Pour tous entiers i, j vérifiant $0 \leq i \leq j \leq |w|$, on note $w(i\,;j] := w[i+1]w[i+2]\dots w[j-1]w[j]$ la *sous-chaîne* de w de longueur $j - i$ commençant à la position i. Pour tout $p \in [0, |w|]$, on note $w^{(p)} := w[1]w[2]\dots w[p]$ le *préfixe* de w de longueur p.

Exemple 1.6 *Soit* $w := \texttt{ALLEZLOM}$ *:* $w[1] = \texttt{A}$*,* $w[4] = \texttt{E}$ *et* $w[|w|] = \texttt{M}$*;* $|w|_{\texttt{L}} = 3$*;* $\tilde{w} = \texttt{MOLZELLA}$*;* $w(3\,;6] = \texttt{EZL}$*;* $w^{(5)} = \texttt{ALLEZ}$*;* $\texttt{LOM}$ *est un suffixe de* w*.*

1.1.3.3 Distance de Hamming

La distance de Hamming est un moyen simple, mais pas toujours pertinent de mesurer la distance séparant deux mots de même longueur.

Pour tous mots x et x' de même longueur, la *distance de Hamming* entre x et x', notée $d_{\mathrm{H}}(x, x')$, est le nombre des indices $i \in [1, |x|]$ pour lesquels on a $x[i] \neq x'[i]$. On vérifie facilement que, pour tout entier $l \geq 0$, $d_{\mathrm{H}}(\,\cdot\,,\,\cdot\,)$ induit une métrique sur l'ensemble des mots de longueur l.

Exemple 1.7 $d_{\mathrm{H}}(\texttt{ATTAAT}, \texttt{TATATT}) = 3$*.*

On convient que $d_{\mathrm{H}}(x, y) = \infty$ pour tous mots x, y avec $|x| \neq |y|$.

1.1.3.4 Langages

On appelle *langage* (sur l'alphabet Σ) tout ensemble de mots (sur Σ). Pour tout langage X, on note $\sigma(X)$ le cardinal du plus petit alphabet Σ tel que $X \subseteq \Sigma^\star$; pour tout mot w, on note $Xw := \{xw : x \in X\}$ et $wX := \{wx : x \in X\}$. Pour tout langage fini X, on note $|X| := \sum_{x \in X} |x|$ (à ne pas confondre avec $\#X$ qui désigne le nombre de mots appartenant à X).

Exemple 1.8 *Pour* $X := \{\texttt{pif}, \texttt{plaf}, \texttt{he}\}$*, on a :*
- $\sigma(X) = 7$ *(*$X \subseteq \{\texttt{p}, \texttt{i}, \texttt{f}, \texttt{l}, \texttt{a}, \texttt{h}, \texttt{e}\}^\star$*),*
- $X\texttt{toto} = \{\texttt{piftoto}, \texttt{plaftoto}, \texttt{hetoto}\}$ *et*
- $|X| = |\texttt{pif}| + |\texttt{plaf}| + |\texttt{he}| = 3 + 4 + 2 = 9$*.*

1.2 Complexité algorithmique

Les principaux résultats de ce mémoire sont des résultats de complexité algorithmique. Nous montrons qu'un certain nombre de problèmes liés à la bio-informatique, ne peuvent être résolus en temps raisonnable, sauf mise en défaut d'hypothèses communément admises comme la conjecture P $\subsetneq$ NP (voir section 1.2.3). Dans cette section, nous commençons par définir informellement la notion de complexité en temps pour les algorithmes (section 1.2.1), puis celle de problème de décision (section 1.2.2). Ensuite, dans les sections 1.2.3, 1.2.4 et 1.2.5, nous introduisons brièvement les trois principales théories bâties pour discriminer les problèmes combinatoires traitables (en temps raisonnable).

1.2.1 Algorithmes et temps de calcul

Objet et taille. Dans ce mémoire, on appelle *objet* tout ce qui peut être codé en machine, c'est-à-dire tout ce qui peut-être représenté naturellement à l'aide d'une suite finie de $\texttt{0}$ et de $\texttt{1}$. Par exemple, les entiers, les mots et les hypergraphes sont des objets. De plus, récursivement, tout ensemble fini et toute suite finie d'objets est un objet. Pour tout objet x, on note $|x|$ la *taille* de l'objet x, c'est-à-dire la longueur du mot sur $\{\texttt{0}, \texttt{1}\}$ qui le représente. On identifiera désormais les objets et leur codage : $\{\texttt{0}, \texttt{1}\}^\star$ peut être vu comme l'ensemble de tous les objets.

Algorithme. Un *algorithme* est une méthode pouvant être appliquée par une machine déterministe à un objet *pris en entrée*. Pour une entrée donnée, un algorithme peut s'arrêter au bout d'un nombre fini d'étapes et *retourner* un objet, ou bien boucler indéfiniement.

Soient une fonction $\varphi : \{\texttt{0}, \texttt{1}\}^\star \to \{\texttt{0}, \texttt{1}\}^\star$ et un algorithme A. On dit que A *calcule* φ lorsque, pour tout $x \in \{\texttt{0}, \texttt{1}\}^\star$:
- A *retourne* $\varphi(x)$ en temps fini lorsque x appartient au domaine de définition de φ et
- A boucle indéfiniment sinon.

La thèse de Church assure que la notion de calculabilité est indépendante du modèle de machine considéré.

Complexité. La *complexité (en temps)* d'un algorithme A est la fonction qui, à tout $x \in \{0,1\}^\star$, associe le nombre d'étapes au bout duquel l'algorithme A termine lorsqu'il prend x en entrée. Une fonction de complexité est à valeurs dans $\mathbb{N} \cup \{\infty\}$.

On dit qu'un algorithme A est *polynomial* lorsqu'il existe un polynôme f tel que, pour tout $x \in \{0,1\}^\star$ pris en entrée, A termine au bout d'au plus $f(|x|)$ étapes. On dit qu'une fonction $\varphi : \{0,1\}^\star \to \{0,1\}^\star$ est *calculable en temps polynomial* lorsqu'il existe un algorithme polynomial qui la calcule.

1.2.2 Problèmes de décision

1.2.2.1 Instances et algorithmes

Soit Δ un *problème de décision* (voir [62] pour une définition formelle). Les *instances* de Δ sont partitionnées en deux classes : les *instances positives* (appelées encore *instances à réponse* `oui`) et les *instances négatives* (appelées encore *instances à réponse* `non`). Étant donnée une instance x de Δ, *résoudre* Δ consiste à décider si x est ou non une instance positive de Δ. Un *algorithme pour* Δ est un algorithme retournant `oui` (resp. `non`) pour chaque instance positive (resp. négative) de Δ prise en entrée.

Exemple 1.9 *Considérons le problème de décision* Prime *: "*Étant donné un entier naturel x, décider si x est premier*". Les instances de* Prime *sont les entiers naturels, les instances positives de* Prime *sont les nombres premiers et les instances négatives de* Prime *sont les nombres composés.*

1.2.2.2 Restriction

Soit X une partie de l'ensemble des instances de Δ. On appelle *restriction* de Δ à X le problème de décision dont l'ensemble des instances est X et dont les instances positives (resp. négatives) sont les instances positives (resp. négatives) de Δ appartenant à X.

Exemple 1.10 *On dit qu'un graphe $G = (V, \mathcal{E})$ est* disconnexe *lorsqu'il existe une partie propre non vide $X \subsetneq V$ telle que pour tout $E \in \mathcal{E}$, on ait $E \subseteq X$ ou $E \subseteq V \setminus X$. On dit qu'un graphe est* connexe *lorsqu'il n'est pas disconnexe.*

On nomme Connected *le problème consistant à décider si un graphe donné est connexe. On vérifie facilement que la restriction de* Connected *aux graphes G possédant moins de $|G| - 1$ (resp. plus de $\frac{1}{2}(|G|-1)(|G|-2)$) arêtes n'admet que des instances négatives (resp. positives).*

Notons qu'un algorithme pour Δ est également un algorithme pour toute restriction de Δ. Ainsi, restreindre un problème ne peut pas augmenter sa complexité algorithmique.

1.2.2.3 M-réduction

Soient Δ_1 et Δ_2 deux problèmes de décision. Une *M-réduction* de Δ_1 à Δ_2 est une fonction calculable associant, à chaque instance positive (resp. négative) de Δ_1, une instance positive (resp. négative) de Δ_2.

Exemple 1.11 *On nomme* Odd *(resp.* Even*) le problème de décision :* "Étant donné un entier n, décider si n est impair (resp. pair)*".*

L'application qui, à tout entier n, associe $n+1$ est une M-réduction de Odd *à* Even *mais aussi une M-réduction de* Even *à* Odd*. Il en est de même pour l'application qui, à tout entier n, associe le reste de la division euclidienne $n+1$ par 8. Cette dernière réduction n'est pas injective.*

Expliquons l'intérêt des M-réductions en terme de construction d'algorithmes. Si l'on dispose d'une M-réduction R de Δ_1 à Δ_2 et d'un algorithme A_2 pour Δ_2, alors on peut construire un algorithme A_1 pour Δ_1 de la manière suivante. Pour chaque instance x_1 de Δ_1 prise en entrée,

1. A_1 simule l'application sur x_1 d'un algorithme calculant R, de manière à obtenir l'instance $x_2 := R(x_1)$ de Δ_2,
2. A_1 retourne le résultat obtenu en appliquant de A_2 sur x_2.

1.2.3 La théorie de la NP-complétude

Dans cette section, nous présentons une brève introduction à la théorie de la NP-complétude dont le livre de Garey et Johnson [40] constitue un guide complet. Informellement, le but de cette théorie est de classer les problèmes de décision en deux catégories. On cherche à distinguer les problèmes qui peuvent être résolus par un algorithme rapide (même dans le pire des cas) de ceux qui sont NP-*difficiles*.

- On entend ici par algorithme rapide, un algorithme dont la complexité a une croissance au plus polynomiale en fonction de la taille de la donnée.
- Un problème NP-difficile est un problème au moins aussi coûteux en temps à résoudre que n'importe lequel des problèmes d'une classe extrêmement vaste, la classe des problèmes NP-*complets*. On trouvera ci-après une définition précise des termes "NP-difficile" et "NP-complet".

Le point faible de la théorie de la NP-complétude est que l'on ne sait pas encore montrer que les deux catégories de problèmes évoquées ci-dessus sont disjointes, c'est-à-dire résoudre la fameuse conjecture P $\neq$ NP. Dans l'état actuel des connaissances, il n'est pas exclu que l'on puisse trouver un algorithme polynomial pour un problème NP-difficile mais, par expérience, toute tentative en ce sens se solde rapidement par un échec.

1.2.3.1 Les classes P et NP

On dit qu'un problème de décision est *polynomial* lorsqu'il admet un algorithme polynomial. On note P la classe des problèmes de décision polynomiaux. Notons que toute restriction d'un problème de décision polynomial est un problème de décision polynomial.

Exemple 1.12 *On vérifie facilement que le problème consistant à décider si un mot donné est un* palindrome *(c'est-à-dire s'il est égal à son image miroir) est dans* P*.*

On note NP la classe des problèmes de décision admettant un algorithme de validation polynomial. Plus formellement, soit Δ un problème de décision. On appelle *algorithme de validation polynomial* pour Δ tout algorithme polynomial C pour lequel il existe un polynôme f vérifiant : pour chaque instance x de Δ, x est une instance positive de Δ si et seulement s'il existe un objet y de taille au plus $f(|x|)$ tel que l'algorithme C retourne `oui` pour l'entrée (x, y). L'objet y est appelé *certificat polynomial* de x.

Exemple 1.13 *Soit $G = (V, \mathcal{E})$ un graphe. Un* chemin hamiltonien *de G est une bijection $h : [1, |G|] \to V$ telle que pour tout $i \in [2, |G|]$, on ait $\{h(i-1), h(i)\} \in \mathcal{E}$. Autrement dit, un chemin hamiltonien est un chemin passant une et une seule fois par chaque sommet.*

Le problème Hamilton Path *s'énonce de la manière suivante :* "Étant donné un graphe G, décider si G admet un chemin hamiltonien"*. On conjecture (voir l'exemple 1.14 p. 13) que* Hamilton Path *n'est pas dans* P*. En revanche, on peut montrer facilement que ce problème est dans* NP *[24] car la tâche consistant simplement à vérifier si un objet donné est bien un chemin hamiltonien de G est de complexité polynomiale. Soyons plus précis. D'une part, il est évident que la taille du codage d'un chemin hamiltonien de G est polynomiale en la taille de G. D'autre part, on peut construire un algorithme de validation polynomial retournant :*

- `oui` *pour chaque entrée de forme (G, h) où G est un graphe et h un chemin hamiltonien de G,*
- `non` *pour toutes les autres entrées.*

Notons que si C est un algorithme de validation polynomial pour Δ alors C est également un algorithme de validation polynomial pour toute restriction de Δ. Ainsi, NP est stable par restriction.

1.2.3.2 Réductions de Karp et problèmes NP-complets

Soient Δ_1 et Δ_2 deux problèmes de décision. Une *réduction de Karp* de Δ_1 à Δ_2 est une M-réduction de Δ_1 à Δ_2 calculable en temps polynomial.

Soit Δ un problème de décision. On dit que Δ est NP-*difficile* lorsque pour tout $\Delta' \in$ NP, il existe une réduction de Karp de Δ' à Δ. On dit que Δ est NP-*complet* lorsque Δ est NP-difficile et appartient à NP. Le théorème de Cook [23] garantit l'existence de problèmes NP-complets. On trouvera de très nombreux exemples de problèmes NP-complets dans [40].

Exemple 1.14 (Suite de l'exemple 1.13 p. 12) HAMILTON PATH *est* NP*-complet [94].*

La propriété fondamentale des problèmes NP-difficiles est la suivante : si *un* problème NP-difficile est polynomial alors il en est de même pour *tous* les problèmes NP-complets et NP = P. Il est admis (mais pas encore démontré) que l'on a P $\subsetneq$ NP.

Pour montrer qu'un problème donné Δ est NP-difficile, la méthode standard, appliquée plusieurs fois dans ce mémoire, consiste à :

– choisir judicieusement un problème Δ_0 connu comme étant NP-complet, puis à
– exhiber une réduction de Karp de Δ_0 à Δ.

1.2.4 Complexité paramétrique

Les meilleurs algorithmes connus à l'heure actuelle pour les problèmes NP-difficiles ont une complexité exponentielle dans le pire des cas. Néanmoins, certains de ces algorithmes exponentiels sont efficaces sur les instances rencontrées en pratique.

Le but de la théorie de la complexité paramétrique, introduite par Downey et Fellows [29], est de distinguer, parmi les problèmes NP-difficiles, ceux admettant des algorithmes exacts et efficaces en pratique.

Expliquons ce que nous entendons ici par "algorithme efficace en pratique". Soit Δ un problème de décision et soit p une fonction associant, à chaque instance x de Δ, un entier $p(x)$. On suppose que, pour les instances x de Δ rencontrées en pratique, $p(x)$ reste petit, disons $O(\log |x|)$. Ceci n'exclut pas, par exemple, $\max\limits_{|x|=n} p(x) = \Omega(n)$ lorsque $n \to \infty$. Un algorithme pour Δ terminant en temps $O\Big(2^{p(x)}\,|x|^{O(1)}\Big)$ pour chaque instance x de Δ prise en entrée peut être considéré comme polynomial en pratique, même s'il est exponentiel dans le pire des cas. Ce n'est pas le cas d'un algorithme de complexité $\Omega\Big(|x|^{O(p(x))}\Big)$.

1.2.4.1 Problèmes de décision paramétrés et la classe FPT

Pour tout problème de décision Δ, un *paramètre* de Δ est une fonction définie sur l'ensemble des instances de Δ. Un *problème de décision paramétré* est un couple (Δ, p) où Δ est un problème de décision et où p est un paramètre de Δ.

Soit p un *paramètre* du problème de décision Δ. On dit que (Δ, p) est *F.P.T.* (*Fixed Parameter Tractable*) lorsque :

– pour un certain polynôme f et
– pour une certaine fonction φ à valeurs positives définie sur l'image de p,

il existe un algorithme pour Δ, terminant en temps $O(\varphi(p(x))f(|x|))$ pour chaque instance x de Δ prise en entrée. Si f est de degré raisonnable et si $\varphi(p(x))$ reste assez petit pour les instances x de Δ rencontrées en pratique, alors on peut considérer que le problème Δ est traitable. Ceci reste vrai même si $\max_{|x|=n} \varphi(p(x))$ croît exponentiellement en fonction de n.

On dit également que Δ *est F.P.T. pour le paramètre* p lorsque (Δ, p) est F.P.T.

Exemple 1.15 *Soit $G = (V, \mathcal{E})$ un graphe. On appelle* transversal *de G toute partie $T \subseteq V$ telle que, pour tout $E \in \mathcal{E}$, on ait $E \cap T \neq \emptyset$. On nomme* VERTEX COVER *le problème de décision lié à la notion de transversal :* "Étant donné un graphe $G = (V, \mathcal{E})$ et un entier $k \geq 0$, décider si G admet un transversal de cardinal k".

Ce problème est NP*-complet [40] donc il n'admet vraisemblablement pas d'algorithme polynomial. En revanche, on trouve dans [29] un algorithme qui, étant donné une graphe G et un entier k, calcule un transversal de G de cardinal k en temps $O\big(2^k\,|G|\big)$, si un tel transversal existe. Notons que l'on trouve des algorithmes de complexité encore plus faible effectuant le même travail [21]. Il est donc possible trouver en temps raisonnable un transversal de petite taille, même dans un très grand graphe.*

Ainsi, VERTEX COVER *est F.P.T. pour le paramètre associant l'entier k à chaque instance (G, k).*

On note FPT la classe des problèmes de décision paramétrés (Δ, p) tels que Δ soit F.P.T. pour le paramètre p.

Remarquons que la théorie de la complexité paramétrique généralise la théorie classique de la complexité : par exemple, un problème de décision est dans P si et seulement s'il est F.P.T. pour un paramètre à valeurs entières borné.

1.2.4.2 Intraitabilité des problèmes de décision paramétrés

Dans cette section, nous expliquons comment se convaincre qu'un problème donné n'est pas F.P.T. pour un paramètre donné. Comme dans le cas de la théorie de la NP-complétude, on n'a pas d'autre choix que d'exhiber des réductions depuis des problèmes paramétrés que l'on conjecture comme n'étant pas F.P.T.

Beaucoup de problèmes paramétrés naturels (on en trouve de nombreux parmi les problèmes NP-complets) semblent impossibles à classer dans FPT.

Exemple 1.16 *Soit $H = (V, \mathcal{E})$ un hypergraphe. On dit qu'une partie $\mathcal{C} \subseteq \mathcal{E}$ est une* couverture *de H ou que H est* couvert *par $\mathcal{C}$ lorsque $V = \bigcup \mathcal{C}$, c'est-à-dire lorsque pour tout $v \in V$, il existe $E \in \mathcal{C}$ tel que $v \in E$.*

On nomme MSC_D *problème de décision suivant : "*Étant donnés un hypergraphe $H = (V, \mathcal{E})$ couvert par $\mathcal{E}$ et un entier $k \geq 0$, décider si H admet une couverture de cardinal au plus k*". MSC_D est le problème de décision associé au problème d'optimisation* Minimum Set Cover *défini dans l'exemple 1.21 p. 16.*

L'algorithme le plus simple pour MSC_D *a recours à la force brute : étant donnée une instance (H, k) de* MSC_D *avec $H = (V, \mathcal{E})$, on teste, pour chaque sous-ensemble $\mathcal{C} \subseteq \mathcal{E}$ de cardinal au plus k, si $\mathcal{C}$ est une couverture de H. Cet algorithme est de complexité $\Omega\left((\#\mathcal{E})^k\right)$ car il y a $\sum_{j=0}^{k} \binom{\#\mathcal{E}}{j}$ sous-ensembles $\mathcal{C}$ à tester. Il ne semble pas que l'on puisse faire significativement mieux et le lecteur pourra essayer de s'en convaincre. On conjecture que* MSC_D *n'est pas F.P.T. pour le paramètre k.*

Comme MSC_D est NP-complet [24], montrer que MSC_D n'est pas F.P.T. pour le paramètre k est encore plus difficile à montrer que P $\subsetneq$ NP. On est donc naturellement amené à définir le concept de réduction suivant.

Soient (Δ_1, p_1) et (Δ_2, p_2) deux problèmes de décision paramétrés. On appelle *FPT-réduction* de (Δ_1, p_1) à (Δ_2, p_2) toute M-réduction R de Δ_1 à Δ_2 pour laquelle il existe

- un polynôme f,
- une fonction φ à valeurs positives définie sur l'image de p_1 et
- une application q de l'image de p_1 vers l'image de p_2

vérifiant les deux propriétés suivantes :

1. pour toute instance x_1 de Δ_1, $p_2(R(x_1)) = q(p_1(x_1))$ (*conservation du paramètre*) et
2. il existe un algorithme calculant $R(x_1)$ en temps $O(\varphi(p_1(x_1))f(|x_1|))$ pour toute instance x_1 de Δ_1 prise en entrée (*traitabilité paramétrique*).

Notons que toutes les FPT-réductions exhibées dans ce mémoire sont également des réductions de Karp (on peut toujours supposer que φ est identiquement égale à 1).

Soit W un ensemble de problèmes de décision paramétrés et soit (Δ, p) un problème de décision paramétré. On dit que (Δ, p) est *W-difficile* lorsque pour tout $(\Delta', p') \in W$, il existe une FPT-réduction de (Δ', p') à (Δ, p). On dit que (Δ, p) est *W-complet* lorsque (Δ, p) est W-difficile et appartient à W. On dit également que Δ est *W-difficile (resp. W-complet) pour le paramètre p* lorsque (Δ, p) est W-difficile (resp. W-complet).

Comme précédemment, pour montrer que le problème de décision paramétré (Δ, p) est W-difficile, il suffit d'exhiber un problème W-complet et une FPT-réduction de ce problème à (Δ, p).

On trouve dans [29] la description d'une suite infinie W[1], W[2], W[3], ... d'ensembles de problèmes de décision paramétrés vérifiant

$$\mathrm{FPT} \subseteq \mathrm{W}[1] \subseteq \mathrm{W}[2] \subseteq \mathrm{W}[3] \subseteq \cdots$$

et telle que, pour tout entier $t \geq 1$, il existe plusieurs problèmes W[t]-complets naturels. De plus, il suffit qu'*un* problème W[t]-complet soit F.P.T. pour que $\mathrm{W}[t] = \cdots = \mathrm{W}[2] = \mathrm{W}[1] = \mathrm{FPT}$. Il est admis (mais pas encore montré) que FPT est une partie propre de W[1] et que pour tout entier $t \geq 1$, W[t] est une partie propre de W[$t+1$].

Exhibons deux problèmes classiques qui sont respectivement W[1]-complet et W[2]-complet pour un paramètre naturel.

Exemple 1.17 ([29]) *Considérons le problème de décision associé (voir section 1.2.5.2) au problème d'optimisation* Maximum Independent Set in Graphs *(voir section 1.3.2). Ce problème de décision est* W[1]*-complet pour son paramètre "seuil d'acceptation" k.*

Le problème MSC_D*, défini dans l'exemple 1.16 p. 14, est* W[2]*-complet pour le paramètre k.*

1.2.5 Optimisation combinatoire

Tous les problèmes que nous avons considérés jusqu'ici sont des problèmes de décision. Or, l'immense majorité des problèmes auxquels est confronté l'Informaticien sont des problèmes de *recherche*. À chaque instance d'un problème de recherche est associé un ensemble de solutions qui n'est pas nécessairement réduit à {`oui`} ou à {`non`}. Résoudre un problème de recherche consiste, pour chaque instance donnée, à identifier au moins une des solutions correspondantes.

Exemple 1.18 *La complexité du problème de recherche* Factoring*, énoncé ci-après, est la principale garantie de l'indéchiffrabilité du cryptosystème RSA [24] :* "Étant donné un entier naturel n, trouver un diviseur de n distinct de 1 et de n"*. Pour tout entier naturel n, l'ensemble de solutions associé à n est l'ensemble des diviseurs non triviaux de n. Le problème* Prime*, énoncé dans l'exemple 1.9 p. 11, consiste simplement à décider si cet ensemble est non vide. Le problème* Factoring *est donc encore plus difficile.*

Dans la section présente nous traitons de la notion de problème d'optimisation. Cette notion généralise naturellement celle de problème de recherche. Chaque solution de chaque instance d'un problème d'optimisation est affectée d'une mesure (voir section 1.2.5.1). Résoudre un problème d'optimisation consiste à trouver, pour chaque instance donnée, une solution correspondante de meilleure mesure possible. Dans les sections 1.2.5.2 et 1.2.5.3 suivantes, nous introduisons les notions de résolution exacte et approchée pour les problèmes d'optimisation.

1.2.5.1 Problèmes d'optimisation

Dans toute la section 1.2.5, Ω désigne un problème d'*optimisation combinatoire* (pour une définition plus formelle voir [5, Définition 1.16]) : Ω peut être un problème de *maximisation* ou de *minimisation*. À chaque *instance* x de Ω, est associé un ensemble non vide $\mathrm{Sol}(x)$ de *solutions*. À chaque couple (x, s) où x est une instance de Ω et où $s \in \mathrm{Sol}(x)$, est associée une *mesure* discrète $\mathrm{mes}(x, s) \in \mathbb{Z} \cup \{-\infty, \infty\}$: $\mathrm{mes}(x, s)$ mesure la qualité de s comme solution de x.

- Dans le cas où Ω est un problème de maximisation, on pose but := sup. Étant donnée une instance x de Ω, *résoudre* Ω consiste à trouver une solution $s \in \mathrm{Sol}(x)$ telle que la mesure $\mathrm{mes}(x, s)$ soit la plus grande possible.
- Symétriquement, si Ω est un problème de minimisation alors on pose but := inf. *Résoudre* Ω sur une instance x donnée consiste à trouver une solution $s \in \mathrm{Sol}(x)$ minimisant $\mathrm{mes}(x, s)$.

Exemple 1.19 *Soit U un ensemble quelconque et soit une application $d : U \times U \to \mathbb{Z}$.*

Le problème de la médiane *pour la distance d (d-*Median*) s'énonce de la manière suivante :* "Étant donnée une partie finie $W \subseteq U$, trouver un point $\mu \in U$ minimisant $\sum_{w \in W} d(\mu, w)$"*.*

*Si l'on prend pour Ω le problème d-*Median*, alors :* but $= \min$*; l'ensemble des instances de Ω est l'ensemble des parties finies $W \subseteq U$;* $\mathrm{Sol}(W) = U$ *quel que soit l'instance W ;* $\mathrm{mes}(W, \mu) = \sum_{w \in W} d(\mu, w)$ *quel que soit $\mu \in U$.*

Exemple 1.20 *Soit T un ensemble de couples. On appelle* couplage *de T toute partie de $M \subseteq T$ telle que deux éléments distincts de M n'aient aucune de leurs deux coordonnées en commun. Par exemple, $T := \{(\mathtt{a}, \mathtt{1}), (\mathtt{a}, \mathtt{2}), (\mathtt{b}, \mathtt{3}), (\mathtt{c}, \mathtt{2}), (\mathtt{c}, \mathtt{4})\}$ admet pour couplage $\{(\mathtt{a}, \mathtt{1}), (\mathtt{b}, \mathtt{3}), (\mathtt{c}, \mathtt{2})\}$ mais pas $\{(\mathtt{a}, \mathtt{2}), (\mathtt{b}, \mathtt{3}), (\mathtt{c}, \mathtt{2})\}$.*

Le problème du couplage maximum *(*Maximum 2-Dimensional Matching *ou, en abrégé,* Max 2-DM*) s'énonce de la manière suivante :* "Étant donné un ensemble de couples T, trouver un couplage de T de cardinal maximum"*.*

Si l'on prend pour Ω le problème du couplage maximum alors : but $= \max$*;* $\mathrm{Sol}(T)$ *est l'ensemble des couplages de T ; pour tout couplage M de T, on a* $\mathrm{mes}(T, M) = \#M$*.*

Exemple 1.21 *On nomme* Minimum Set Cover *(*MSC*) le problème de minimisation suivant :* "Étant donné un hypergraphe $H = (V, \mathcal{E})$ couvert par $\mathcal{E}$, trouver une couverture de H cardinal minimum*". La notion de couverture d'hypergraphe est définie dans l'exemple 1.16 p. 14.*

Si l'on prend pour Ω *le problème* MSC *alors :* but $=$ min, $\mathrm{Sol}(H)$ *est l'ensemble des couvertures de* H *et, pour toute couverture* $\mathcal{C} \in \mathrm{Sol}(H)$*, on a* $\mathrm{mes}(H, \mathcal{C}) = \#\mathcal{C}$*.*

Pour toute instance x de Ω, on note $\mathrm{opt}(x)$ la mesure d'une solution optimale de x :

$$\mathrm{opt}(x) = \mathrm{but}\,\{\mathrm{mes}(x, s) : s \in \mathrm{Sol}(x)\}\ .$$

1.2.5.2 Résolution exacte des problèmes d'optimisation

Un *algorithme exact* pour Ω est un algorithme retournant, pour toute instance x de Ω, une solution $s \in \mathrm{Sol}(x)$ telle que $\mathrm{mes}(x, s) = \mathrm{opt}(x)$.

Exemple 1.22 (Suite de l'exemple 1.20 p. 15) *Le problème du couplage maximum admet un algorithme exact polynomial [94].*

On appelle *problème de décision associé à* Ω le problème de décision (noté Ω_D) suivant : "*Étant donnés une instance* x *de* Ω *et un entier (noté généralement)* k*, décider si* $\mathrm{but}\{\mathrm{opt}(x), k\} = \mathrm{opt}(x)$". Ainsi, une instance de Ω_D est un couple (x, k) où x est une instance de Ω et où k est un entier. Le paramètre supplémentaire k est appelé *seuil d'acceptation* de l'instance (x, k). Notons que, si Ω est un problème de maximisation (resp. de minimisation) alors on a $\mathrm{but}\{\mathrm{opt}(x), k\} = \mathrm{opt}(x)$ si et seulement si $\mathrm{opt}(x) \geq k$ (resp. $\mathrm{opt}(x) \leq k$).

Exemple 1.23 (Suite des exemples 1.16, 1.19, 1.20 et 1.21)

- *Le problème* d-Median$_D$ *s'énonce :* "Étant donnés une partie finie $W \subseteq U$ et un entier K, décider s'il existe $\mu \in U$ tel que $\sum_{w \in W} d(\mu, w) \leq K$*"*.
- *Le problème* Max 2-DM$_D$ *s'énonce :* "Étant donnés un ensemble T de couples et un entier $k \geq 0$, décider si T admet un couplage de cardinal (au moins) k*"*.
- *Le problème* MSC$_D$ *est énoncé dans l'exemple 1.16 p. 14.*

On note NPO (O pour Optimisation [5]) l'ensemble des problèmes d'optimisation Ω possédant les propriétés $(NPO1)$, $(NPO2)$ et $(NPO3)$ suivantes.

$(NPO1)$. La fonction $\mathrm{mes}(\,\cdot\,,\,\cdot\,)$ est calculable en temps polynomial.

$(NPO2)$. Il existe un polynôme f tel que pour toute instance x de Ω et toute solution $s \in \mathrm{Sol}(x)$, on ait $|s| \leq f(|x|)$.

$(NPO3)$. Le problème de décision suivant est polynomial : "*Étant donnés une instance* x *et de* Ω *et un objet quelconque* s *décider si* $s \in \mathrm{Sol}(x)$".

Plusieurs problèmes d'optimisation rencontrés dans ce mémoire sont dans NPO.

Exemple 1.24 (Suite des exemples 1.19, 1.20 et 1.21) *Vérifions que le problème* Max 2-DM *est dans* NPO*. Supposons donnée une instance* T *de* Max 2-DM*.*

- *On peut calculer le cardinal d'un couplage donné de* T *en temps polynomial, donc* Max 2-DM *vérifie* $(NPO1)$*.*
- *De plus, tout couplage de* T *est de taille au plus égale à la taille de* T*, donc* Max 2-DM *satisfait* $(NPO2)$*.*
- *Enfin, décider si un objet donné est un couplage de* T *est réalisable en temps polynomial, donc* Max 2-DM *vérifie* $(NPO3)$*.*

On vérifie de même que MSC *est dans* NPO*.*

En revanche, lorsque U *est infini, chaque instance de* d-Median *admet une infinité de solutions. Par suite,* d-Median *n'est pas dans* NPO *car la propriété* $(NPO2)$ *est mise en défaut.*

D'une part, si $\Omega \in$ NPO, alors on a $\Omega_D \in$ NP. D'autre part, il suffit que Ω_D ne soit pas polynomial pour que Ω n'admette pas d'algorithme polynomial exact.

Exemple 1.25 (Suite des exemples 1.21 p. 16 et 1.24 p. 16) *Le problème* MSC_D *est* NP*-complet [24] donc, sous l'hypothèse* $\mathrm{P} \subsetneq \mathrm{NP}$, MSC_D *n'est pas polynomial. Par suite, toujours sous l'hypothèse* $\mathrm{P} \subsetneq \mathrm{NP}$, MSC *n'admet pas d'algorithme exact polynomial.*

1.2.5.3 Résolution approchée des problèmes d'optimisation

Beaucoup de problèmes d'optimisation n'admettent pas d'algorithmes polynomiaux exacts (ou alors seulement dans le cas improbable où NP = P). Pour de tels problèmes, il est donc naturel de se contenter d'algorithmes d'approximation c'est-à-dire d'algorithmes susceptibles de ne retourner que des solutions sous-optimales (voir [5] pour de nombreux exemples). Dans cette section, nous supposons que la fonction $\mathrm{mes}(\cdot\,,\cdot\,)$ ne prend que des valeurs positives et que, pour toute instance x de Ω, on a $\mathrm{opt}(x) < \infty$, même lorsque Ω est un problème de maximisation et que $\mathrm{Sol}(x)$ est infini.

Borne d'approximation. Un *algorithme (d'approximation)* pour Ω est un algorithme retournant, pour chaque instance x de Ω prise en entrée, un élément de $\mathrm{Sol}(x)$. Soit ρ une fonction associant, à chaque instance x de Ω, un réel $\rho(x) \geq 1$. On dit qu'un algorithme d'approximation pour Ω est de *borne* ρ lorsque, pour chaque instance x de Ω, il retourne une solution $s \in \mathrm{Sol}(x)$ vérifiant :

$$\frac{1}{\rho(x)} \times \mathrm{opt}(x) \leq \mathrm{mes}(x, s) \leq \rho(x) \times \mathrm{opt}(x)\,.$$

Notons que si Ω est un problème de maximisation (resp. de minimisation) alors la deuxième (resp. la première) inégalité de l'encadrement ci-dessus est toujours vérifiée. On appelle *ρ-approximation de* Ω tout algorithme d'approximation pour Ω de borne ρ. Un algorithme d'approximation de borne constante égale à 1 est un algorithme exact.

Exemple 1.26 (Suite de l'exemple 1.21 p. 16) *Soit ν la fonction qui, à chaque hypergraphe $H = (V, \mathcal{E})$ couvert par $\mathcal{E}$, associe $\nu(H) := \#\mathcal{E}$. Considérons l'algorithme d'approximation trivial retournant, pour chaque instance $H = (V, \mathcal{E})$ de* MSC*, l'ensemble $\mathcal{E}$ lui-même. Cet algorithme admet pour borne ν (il se peut que $V \in \mathcal{E}$).*

Soit ρ l'application qui, à chaque hypergraphe $H = (V, \mathcal{E})$ couvert par $\mathcal{E}$, associe $\rho(H) := 1 + \ln(\max_{E \in \mathcal{E}} \#E)$. Alors, il existe un algorithme d'approximation (glouton) pour MSC *admettant pour borne ρ [24].*

On appelle *P.T.A.S.* (*Polynomial-Time Approximation Scheme*) pour Ω toute famille $(A_i)_{i \in I}$ d'algorithmes indexée sur un ensemble I quelconque vérifiant : pour tout réel $r > 1$, il existe $i \in I$ tel que A_i soit un algorithme d'approximation pour Ω de borne constante r.

Exemple 1.27 *Le problème* Maximum Knapsack *s'énonce :* "Supposons donnés un ensemble fini I, deux applications s, $v : I \to \mathbb{N}$ et un entier $M \geq 0$. On cherche une partie $J \subseteq I$ vérifiant $\sum_{j \in J} s(j) \leq M$ et telle que $\sum_{j \in J} v(j)$ soit maximum". *Sous l'hypothèse* $\mathrm{P} \subsetneq \mathrm{NP}$, Maximum Knapsack *n'admet pas d'algorithme exact polynomial (son problème de décision associé est* NP*-complet). En revanche,* Maximum Knapsack *admet un P.T.A.S. [52].*

Problème d'évaluation. On appelle *problème de d'évaluation associé à* Ω le problème d'optimisation (noté Ω_E) suivant : "*Étant donnés une instance x de Ω, trouver un entier n avec* $\mathrm{but}\{\mathrm{opt}(x), n\} = \mathrm{opt}(x)$*, le plus proche possible de* $\mathrm{opt}(x)$". Un algorithme exact pour Ω_E retourne l'entier $\mathrm{opt}(x)$ pour toute instance x de Ω prise en entrée. Un algorithme d'approximation de borne ρ pour Ω_E retourne, pour chaque instance x de Ω prise en entrée, un entier compris entre $\mathrm{opt}(x)\ /\ \rho(x)$ et $\rho(x) \times \mathrm{opt}(x)$. Supposons que la fonction $\mathrm{mes}(\cdot\,,\cdot\,)$ soit calculable en temps polynomial. Alors, il suffit que Ω admette une ρ-approximation pour que Ω_E admette également un algorithme d'approximation de borne ρ : Ω est au moins aussi difficile à approximer que Ω_E.

Exemple 1.28 *On considère le problème* Minimum k-Center *:* "Supposons donnés un ensemble fini U, une métrique d sur U et un entier $k \geq 0$. On cherche une partie $C \subseteq U$ de cardinal k minimisant $\max_{x \in U} (\min_{c \in C} d(x, c))$." *Ce problème admet un algorithme d'approximation de borne constante* 2 *[50, 42]. En revanche, on peut montrer (par des méthodes élémentaires) le résultat négatif suivant : s'il existe un algorithme d'approximation de borne constante strictement inférieure à* 2 *pour* Minimum k-Center$_E$*, alors* NP = P *[50, 42].*

1.3 Le problème de l'indépendant maximum

Le problème de l'indépendant maximum est l'un des problèmes d'optimisation combinatoire les plus connus. Son problème de décision associé est l'un des tous premiers problèmes à avoir été montré NP-complet par Karp en 1972 [59].

1.3.1 Définition

Soit $H = (V, \mathcal{E})$ un hypergraphe. Un *indépendant* (ou *stable*) de H est une partie de V n'admettant comme sous-ensemble aucun élément de $\mathcal{E}$. On note $\alpha(H)$ le cardinal maximum d'un indépendant de H. Dans le cas particulier où H est un graphe, un sous-ensemble $I \subseteq V$ est un indépendant de H si et seulement si chaque arête de H a au plus une extrémité dans I.

Exemple 1.29 (Suite l'exemple 1.2 p. 9) $\{1, 3, 5, 6, 7\}$ *est un indépendant de* H *de cardinal* $\alpha(H)$*, alors que* $\{1, 3, 4\}$ *n'est pas un indépendant de* H *(car* E_5 *est inclus dans ce dernier ensemble).*

On nomme Maximum Independent Set in Hypergraphs (MISH) le problème de maximisation suivant : "*Étant donné un hypergraphe* H*, trouver un indépendant de* H *de cardinal maximum*". Ce problème est dans NPO. On pourra toujours considérer que les instances H de MISH sont des hypergraphes admettant $[1, |H|]$ pour ensemble de sommets. Pour tout entier $r \geq 2$, on nomme r-MISH la restriction de MISH aux hypergraphes r-uniformes. Le problème 2-MISH est également nommé Maximum Independent Set In Graphs ou même simplement Maximum Independent Set. Une instance générique de 2-MISH sera indifféremment désignée par G ou H.

1.3.2 Complexité

Le problème de décision MISH$_D$ associé à MISH s'énonce de la manière suivante : "*Étant donnés un hypergraphe* $H = (V, \mathcal{E})$ *et un entier* $k \geq 0$*, décider si* H *admet un indépendant de cardinal* k ". Le problème 2-MISH$_D$ est :

- NP-complet [24],
- polynomial lorsque k est fixé (car on peut alors énumérer toutes les parties de V de cardinal k en temps polynomial) et
- W[1]-complet pour le paramètre k [29].

Notons que nous démontrons dans ce mémoire que r-MISH$_D$ est W[1]-difficile pour le paramètre k, quel que soit l'entier $r \geq 3$ (théorème 3.2 p. 76).

1.3.3 Approximabilité

Le problème MISH est très difficile à approximer.

Résultats positifs. L'algorithme d'approximation trivial retournant, pour chaque hypergraphe H pris en entrée, un indépendant de cardinal 1 admet pour borne $|H|$. La méthode d'*approximation par partitionnement* permet d'obtenir une borne de $O(|H| \,/\, \log |H|)$ [46] (la meilleure connue à ce jour). Dans le cas particulier de 2-MISH, on peut faire encore légèrement mieux et obtenir une borne de $O\big(|G| \,/\, \log^2 |G|\big)$ [16]. Des résultats spécifiques à l'approximabilité de certaines restrictions de 3-MISH sont également connus [63].

Résultats négatifs. Les résultats suivants assurent que la borne donnée ci-dessus pour 2-MISH est presque optimale. Quel que soit le réel δ vérifiant $0 \leq \delta < 0.5$ (resp. $0 \leq \delta < 1$), 2-MISH n'admet pas d'algorithme d'approximation de borne $|G|^{\delta}$ sauf si NP = P (resp. NP = ZPP) [47]. On trouvera dans [94] une définition de la classe ZPP. On a P $\subseteq$ ZPP $\subseteq$ NP et on conjecture ZPP $\subsetneq$ NP.

Il est également montré dans [33] que 2-MISH n'admet pas d'algorithme d'approximation de borne $|G|^{\delta(|G|)}$ avec $\delta(n) = 1 - O\Big((\log\log n)^{-0.5}\Big)$ sauf si

$$\mathrm{NP} \subseteq \mathrm{ZPTIME}\left(2^{O\left((\log n)(\log\log n)^{1.5}\right)}\right) .$$

Notre théorème 3.3 p. 76 garantit que, quel que soit l'entier $r \geq 3$, les résultats ci-dessus restent vrai si l'on remplace 2-MISH par r-MISH.

1.4 Le problème de la plus longue sous-séquence commune

Le problème de la plus longue sous-séquence commune est un problème central en algorithmique du texte. Il constitue une version dégénérée du problème de l'alignement multiple (voir sections 1.5 et 2.4). Il trouve de nombreuses applications, par exemple dans les domaines suivants : compression de textes, conception de codes détecteurs d'erreur, comparaison de séquences biologiques, ... (voir [101] pour un survol).

Dans les chaptires 2 et 3 de ce mémoire, nous utilisons et nous généralisons un certain nombre de résultats connus sur la complexité algorithmique du problème de la plus longue sous-séquence commune.

1.4.1 Définition

Soient deux mots w et s. On dit que s est une sous-séquence de w lorsqu'on peut obtenir s à partir de w en effaçant des lettres (éventuellement toutes ou aucune).

Introduisons une notation non standard.

Définition 1.1
Soient un mot w quelconque et une partie $I \subseteq [1, |w|]$.

On note $\mathrm{sseq}(w, I)$ *la sous-séquence de w obtenue en effaçant de w tous ses lettres apparaissant à des positions n'appartenant pas à $I - 1$.*

Plus formellement, pour toute suite strictement croissante d'indices $1 \leq i_1 < i_2 < \cdots < i_k \leq |w|$, on a

$$w[i_1]w[i_2]\ldots w[i_k] = \mathrm{sseq}(w, \{i_1, i_2, \ldots, i_k\}) .$$

Notons que $\mathrm{sseq}(w, I)$ est toujours de longueur $\#I$.

Exemple 1.30 *Prenons* $w :=$ `abracadabra`*. Les mots*
- $\varepsilon = \mathrm{sseq}(w, \emptyset)$,
- $\mathtt{a}^5 = \mathrm{sseq}(w, \{1, 4, 6, 8, 11\})$,
- $\mathtt{brcdbr} = \mathrm{sseq}(w, \{2, 3, 5, 7, 9, 10\})$,
- $(\mathtt{abra})^2 = \mathrm{sseq}(w, [1, 4] \cup [8, 11])$ *et*
- $w = \mathrm{sseq}(w, [1, 11])$

sont des sous-séquences de w.

Soit X un langage non vide. Une *sous-séquence commune* de X est un mot s tel que, pour tout $x \in X$, s est une sous-séquence de x. On note $\mathrm{lcs}(X)$ la longueur maximale d'une sous-séquence commune de X.

Exemple 1.31 *Soient* $x :=$ `ATCTGAT` *et* $y :=$ `TGCATA`*. Alors,* `TCTA` *et* `TGAT` *sont deux plus longues sous-séquences communes de $\{x, y\}$, et donc* $\mathrm{lcs}(\{x, y\}) = 4$.

On nomme Longest Common Subsequence (LCS) le problème de maximisation suivant : "*Étant donné un langage fini et non vide X, trouver une sous-séquence commune de X de longueur maximale*". Ce problème est dans NPO.

1.4.2 Algorithmes exacts

Un algorithme de programmation dynamique connu depuis longtemps permet de calculer, pour deux mots x, y donnés quelconques, une plus longue sous-séquence commune à x et y en temps quadratique $O(|x|\,|y|)$ [24]. Il se généralise directement en un algorithme permettant de calculer, pour tout langage fini X pris en entrée, une plus longue sous-séquence commune de X en temps $O\big((\#X)\prod_{x\in X}|x|\big)$ [53] (voir aussi section 2.4.1). Ainsi, lorsque $\#X$ est fixé, le problème LCS est polynomial.

Notons qu'il existe des algorithmes avec des complexités nettement meilleures pour de nombreuses restrictions intéressantes du problème (voir [4, chapitre 4] et [108, chapitre 3] pour un survol). Par exemple, le cas où on se restreint à rechercher une plus longue sous-séquence commune à *deux* mots a été très largement étudié. Nous y reviendrons dans la section 1.5 dans le cadre plus général du problème de l'*alignement* de deux mots.

1.4.3 Complexité paramétrique

Le problème de décision LCS_D associé à LCS s'énonce de la manière suivante : "*Étant donnés un langage fini, non vide X et un entier $k \geq 0$, décider si X admet une sous-séquence commune de longueur k*".

La complexité paramétrique de LCS_D a été étudiée pour toutes les combinaisons des 3 paramètres $\#X$, $\sigma(X)$ et k [14, 13, 97] :

- $\#X$ est le nombre de mots pris en entrée,
- $\sigma(X)$ est le cardinal de l'alphabet d'entrée et
- k est la longueur minimum acceptable pour une plus longue sous-séquence commune de X.

Le tableau suivant résume les divers résultat connus :

$\#X$	$\sigma(X)$	k	LCS_D
borné	–	–	polynomial [53]
–	paramètre	paramètre	F.P.T. (remarque 3.7 p. 68)
–	borné par 2	–	NP-complet [81]
paramètre	borné par 2	–	W[1]-difficile [97]
–	–	paramètre	W[2]-difficile [14]
paramètre	–	paramètre	W[1]-complet [14]
paramètre	paramètre	–	W[t]-difficile $\forall t \geq 1$ [13]

.

1.4.4 Approximabilité

On trouve dans [55] (voir aussi notre théorème 3.4 p. 80) une réduction de Maximum Clique (qui est une formulation alternative de 2-MISH) vers LCS garantissant le résultat suivant. Quelle que soit l'application b de $\mathbb{N}\times\mathbb{N}\times\mathbb{N}$ vers l'ensemble des réels supérieurs ou égaux à 1, on a : si LCS admet un algorithme d'approximation de borne $b\big(\min_{x\in X}|x|, \sigma(X), \#X\big)$ alors 2-MISH admet un algorithme d'approximation de borne $b(|G|, |G|, 2\,|G|)$. Les résultats donnés dans la section 1.3 garantissent donc que LCS est très difficile à approximer.

Néanmoins, LCS admet des algorithmes d'approximation de bornes

- $\sigma(X)$ [55] (voir aussi la proposition 3.4 p. 71) et
- $(\min_{x\in X}|x|) \,/\, \log(\min_{x\in X}|x|)$ [46] (voir aussi la proposition 3.5 p. 73).

Notons qu'en toute rigueur, le rapport $(\min_{x\in X}|x|)/\log(\min_{x\in X}|x|)$ n'est pas défini lorsqu'on a $\min_{x\in X}|x| \leq 1$ c'est-à-dire lorsque X contient un mot α, vide ou réduit à une lettre. Or, dans ce cas, α ou ε est l'unique plus longue sous-séquence commune de X. Ainsi, ces instances "problématiques" se traitent trivialement et, par conséquent, peuvent être écartées.

1.5 Alignement

L'alignement est un moyen largement utilisé pour visualiser la similarité de deux ou plusieurs mots. Les mots alignés sont des séquences biologiques dans la plupart des applications [45], mais pas seulement [101].

1.5.1 Définitions

Un *alignement* est une matrice A vérifiant :
- chaque entrée de A est un mot de longueur au plus 1 (c'est-à-dire soit une lettre, soit le mot vide) et
- dans chaque colonne de A, il apparaît au moins une lettre.

Soient un entier $m \geq 1$ et m mots w_1, w_2, …, w_m. On dit que A est un alignement *de* $(w_1, w_2, \ldots, w_m)$ lorsque A vérifie :
- A possède exactement m lignes et
- pour chaque $i \in [1, m]$, on obtient w_i en concaténant les entrées de A situées sur sa i-ème ligne (dans l'ordre naturel gauche-droite).

Autrement dit, la matrice à m lignes et n colonnes

$$\begin{bmatrix} \alpha_{1,1} & \alpha_{1,2} & \cdots & \alpha_{1,n} \\ \alpha_{2,1} & \alpha_{2,2} & \cdots & \alpha_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{m,1} & \alpha_{m,2} & \cdots & \alpha_{m,n} \end{bmatrix}$$

est un alignement de $(w_1, w_2, \ldots, w_m)$ si et seulement si, pour tout $i \in [1, m]$ et tout $j \in [1, n]$,
- $\alpha_{i,j}$ est une lettre ou le mot vide,
- $w_i = \alpha_{i,1}\alpha_{i,2}\ldots\alpha_{i,n}$ et
- $\alpha_{1,j}\alpha_{2,j}\ldots\alpha_{m,j} \neq \varepsilon$.

Notons que la matrice à m lignes et 0 colonne est l'unique alignement de m exemplaires du mot vide. Lorsqu'on dessine un alignement, on convient de représenter le mot vide par $-$ pour plus de clarté visuelle.

Exemple 1.32 *La matrice*

$$\begin{bmatrix} \texttt{o} & \texttt{c} & \texttt{t} & \texttt{o} & \texttt{b} & \texttt{r} & - & \texttt{e} & - \\ \texttt{o} & - & \texttt{t} & \texttt{a} & - & \texttt{r} & \texttt{i} & \texttt{e} & \texttt{s} \\ \texttt{o} & \texttt{c} & \texttt{t} & \texttt{a} & - & - & \texttt{v} & \texttt{e} & - \end{bmatrix}$$

est un alignement de $(\texttt{octobre}, \texttt{otaries}, \texttt{octave})$.

Le nombre de colonnes d'un alignement de $(w_1, w_2, \ldots, w_m)$ est compris entre $\max\{|w_1|, |w_2|, \ldots, |w_m|\}$ et $|w_1| + |w_2| + \cdots + |w_m|$.

Exemple 1.33 *Soient* $w_1 := \texttt{pif}$, $w_2 := \texttt{plaf}$ *et* $w_3 = \texttt{he}$.
Alors

$$\begin{bmatrix} \texttt{p} & \texttt{i} & \texttt{f} & - & - & - & - & - & - \\ - & - & - & \texttt{p} & \texttt{l} & \texttt{a} & \texttt{f} & - & - \\ - & - & - & - & - & - & - & \texttt{h} & \texttt{e} \end{bmatrix} \quad \textit{et} \quad \begin{bmatrix} \texttt{p} & \texttt{i} & \texttt{f} & - \\ \texttt{p} & \texttt{l} & \texttt{a} & \texttt{f} \\ \texttt{h} & \texttt{e} & - & - \end{bmatrix}$$

sont deux alignements de (w_1, w_2, w_3) *possédant respectivement* $|w_1| + |w_2| + |w_3| = 9$ *et* $\max\{|w_1|, |w_2|, |w_3|\} = 4$ *colonnes.*

Dans le reste de ce chapitre, nous ne considérons plus que des alignements de deux mots. Nous reviendrons au problème de l'alignement multiple dans la section 2.4.

1.5.2 Distance d'édition

La distance d'édition est un moyen couramment utilisé pour mesurer la quantité d'évolution séparant deux séquences biologiques. Soient Σ un alphabet (éventuellement infini) et $\pi : (\Sigma \cup \{\varepsilon\}) \times (\Sigma \cup \{\varepsilon\}) \to \mathbb{Z}$.

1.5.2.1 Définition

La *fonction de pondération* π permet d'associer un score à chaque alignement d'un couple de mots sur Σ. Considérons un alignement à deux lignes

$$A = \begin{bmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_n \\ \beta_1 & \beta_2 & \cdots & \beta_n \end{bmatrix}$$

où $n \in \mathbb{N}$ et où les α_i et les β_i sont des éléments de $\Sigma \cup \{\varepsilon\}$ vérifiant $\alpha_i\beta_i \neq \varepsilon$. On définit le π-*score* de l'alignement A comme étant la quantité $\pi(\alpha_1, \beta_1) + \pi(\alpha_2, \beta_2) + \cdots + \pi(\alpha_n, \beta_n)$. Notons que la valeur de $\pi(\varepsilon, \varepsilon)$ ne joue aucun rôle dans cette définition.

Exemple 1.34 *Soient* $\Sigma := \{\mathtt{A}, \mathtt{T}, \mathtt{G}, \mathtt{C}\}$, $x := \mathtt{ATCGAT}$, $y := \mathtt{GATGCAT}$ *et* $\pi : \{\mathtt{A}, \mathtt{T}, \mathtt{G}, \mathtt{C}, \varepsilon\} \times \{\mathtt{A}, \mathtt{T}, \mathtt{G}, \mathtt{C}, \varepsilon\} \to \mathbb{Z}$ *donnée par le tableau :*

π	A	T	G	C	ε
A	-5	3	2	2	4
T	3	-5	2	2	4
G	2	2	-5	3	4
C	2	2	3	-5	4
ε	4	4	4	4	$*$

.

Alors,

$$\begin{bmatrix} - & \mathtt{A} & \mathtt{T} & \mathtt{C} & \mathtt{G} & \mathtt{A} & \mathtt{T} \\ \mathtt{G} & \mathtt{A} & \mathtt{T} & \mathtt{G} & \mathtt{C} & \mathtt{A} & \mathtt{T} \end{bmatrix} \quad \textit{et} \quad \begin{bmatrix} - & \mathtt{A} & \mathtt{T} & \mathtt{C} & \mathtt{G} & - & \mathtt{A} & \mathtt{T} \\ \mathtt{G} & \mathtt{A} & \mathtt{T} & - & \mathtt{G} & \mathtt{C} & \mathtt{A} & \mathtt{T} \end{bmatrix}$$

sont deux alignements de (x, y) *de* π*-scores respectifs* $4-5-5+3+3-5-5 = -10$ *et* $4-5-5+4-5+4-5-5 = -13$.

Pour tous mots x, $y \in \Sigma^\star$, la *distance d'édition pondérée par* π entre x et y (notée $d_{\mathrm{E}}^{\pi}(x, y)$) est le π-score minimum pour un alignement de (x, y). On vérifie facilement que, si π est symétrique (resp. satisfait l'inégalité triangulaire, resp. est positive et séparante, resp. est une semi-métrique, resp. est une métrique) alors il en est de même pour $d_{\mathrm{E}}^{\pi}(\,\cdot\,,\,\cdot\,)$.

Remarque 1.3 *Pour tout* $w \in \Sigma^\star$, *on a :*

$$d_{\mathrm{E}}^{\pi}(w, \varepsilon) = \sum_{i=1}^{|w|} \pi(w[i], \varepsilon) \quad \left(\textit{resp. } d_{\mathrm{E}}^{\pi}(\varepsilon, w) = \sum_{i=1}^{|w|} \pi(\varepsilon, w[i]) \right)$$

car

$$\begin{bmatrix} w[1] & w[2] & \cdots & w[|w|] \\ - & - & \cdots & - \end{bmatrix} \quad \left(\textit{resp. } \begin{bmatrix} - & - & \cdots & - \\ w[1] & w[2] & \cdots & w[|w|] \end{bmatrix} \right)$$

est l'unique alignement de (w, ε) *(resp.* (ε, w)*).*

Soit δ est la *métrique discrète* sur l'ensemble des symboles de l'univers augmenté de ε. La distance d'édition d_{E}^{δ} est une métrique sur l'ensemble des mots appelée *distance de Levenshtein* [70]. On la notera désormais d_{L}. Pour tous mots x, y, on peut dire informellement que $d_{\mathrm{L}}(x, y)$ est le nombre minimum d'insertions, délétions et substitutions nécessaires pour transformer x en y. Cette dernière affirmation sera formalisée à fin de la section 1.5.2.2 suivante.

Exemple 1.35 *Soient un entier* $N \geq 1$, $x := (\mathtt{01})^N$ *et* $y := (\mathtt{10})^N$. *On a* $d_{\mathrm{L}}(x, y) = 2$ *car on peut aligner optimalement* (x, y) *de la manière suivante :*

$$\begin{bmatrix} \mathtt{0} & \mathtt{1} & \mathtt{0} & \mathtt{1} & \cdots & \mathtt{0} & \mathtt{1} & - \\ - & \mathtt{1} & \mathtt{0} & \mathtt{1} & \cdots & \mathtt{0} & \mathtt{1} & \mathtt{0} \end{bmatrix}.$$

Notons que, par ailleurs, on a $d_{\mathrm{H}}(x, y) = 2N$.

1.5.2.2 Opérations d'édition

On considère les trois opérations élémentaires suivantes sur les mots : l'*insertion* d'une lettre, la *délétion* (ou *effacement*) d'une lettre et la *substitution* d'une lettre par une autre. Ces opérations sont appelées *opérations d'édition élémentaires* (*OÉÉ* en abrégé).

Formellement, une OÉÉ est un couple de la forme $(u\alpha v, u\beta v)$ avec u, $v \in \Sigma^\star$ et α, $\beta \in \Sigma \cup \{\varepsilon\}$ vérifiant $\alpha \neq \beta$.

Soient a, $b \in \Sigma$ avec $a \neq b$.

1. Une *insertion* de b est une OÉÉ de la forme (uv, ubv) avec u, $v \in \Sigma^\star$.
2. Une *délétion* (ou *effacement*) de a est une OÉÉ de la forme (uav, uv) avec u, $v \in \Sigma^\star$.
3. Une *substitution* de a en b est une OÉÉ de la forme (uav, ubv) avec u, $v \in \Sigma^\star$.

La fonction π permet d'associer un *coût* à chaque OÉÉ : pour tous u, $v \in \Sigma^\star$ et tous α, $\beta \in \Sigma \cup \{\varepsilon\}$ avec $\alpha \neq \beta$, l'OÉÉ $(u\alpha v, u\beta v)$ est affectée du coût $\pi(\alpha, \beta)$.

Une *suite d'éditions élémentaires* (*SÉÉ* en abrégé) est une suite finie non vide $(w_0, w_1, \ldots, w_l)$ de mots sur Σ telle que pour tout $i \in [1, l]$, (w_{i-1}, w_i) est une OÉÉ. Soient x, $y \in \Sigma^\star$. On dit qu'une SÉÉ transforme x en y lorsqu'elle admet x et y respectivement pour premier et dernier terme ($w_0 = x$ et $w_l = y$).

Remarque 1.4 *Quels que soient les mots x et y, il existe toujours une SÉÉ transformant x en y. Il suffit par exemple d'effacer toutes les lettres de x puis d'insérer toutes les lettres de y :*

$$\left(x, x^{(|x|-1)}, \ldots, x^{(2)}, x^{(1)}, \varepsilon, y^{(1)}, y^{(2)}, \ldots, y^{(|y|-1)}, y\right).$$

On définit le *coût pondéré par π* de la SÉÉ $(w_0, w_1, \ldots, w_l)$ comme étant la quantité $c_1 + c_2 + \cdots + c_l$ où, pour chaque $i \in [1, l]$, c_i est le coût pour π affecté à l'OÉÉ (w_{i-1}, w_i). Notons que, dans cette définition, les valeurs des $\pi(\alpha, \alpha)$ ($\alpha \in \Sigma \cup \{\varepsilon\}$) ne jouent aucun rôle.

Exemple 1.36 *Soient $\Sigma := \{\mathtt{A}, \mathtt{T}\}$ et π donnée par le tableau*

π	A	T	ε
A	$*$	1	2
T	3	$*$	4
ε	5	6	$*$

.

Le coût pondéré par π de la SÉÉ

$$(\mathtt{TAATA}, \mathtt{TATTA}, \mathtt{TTTA}, \mathtt{ATTA}, \mathtt{ATA}, \mathtt{AATA}, \mathtt{ATATA})$$

est égal à $1 + 2 + 3 + 4 + 5 + 6 = 21$.

On suppose que π satisfait l'inégalité triangulaire. Alors, pour tous x, $y \in \Sigma^\star$, $d_{\mathrm{E}}^{\pi}(x, y)$ est le coût pondéré par π minimum sur toutes les SÉÉ transformant x en y. En particulier, la distance de Levenshtein de x à y est égale à la longueur minimale pour une SÉÉ transformant x en y, diminuée de 1.

1.5.3 Distance d'édition et plus longue sous-séquence commune

Dans cette section, nous mettons en évidence les deux faits suivants :

1. Le problème de l'*évaluation* de la distance d'édition entre deux mots donnés est une généralisation du problème de l'*évaluation* de la longueur d'une plus longue sous-séquence commune à deux mots donnés.
2. Le problème de la *recherche* d'un alignement de score minimum entre deux mots donnés est une généralisation du problème de la *recherche* d'une plus longue sous-séquence commune à deux mots donnés.

On définit de la manière suivante une métrique λ sur l'ensemble des symboles de l'univers augmenté de ε. Pour toutes lettres a, b avec $a \neq b$, λ est donnée par :

– $\lambda(a, a) = 0$ (l'appariement exact ne coûte rien),
– $\lambda(a, \varepsilon) = \lambda(\varepsilon, a) = 1$ (toute insertion/délétion coûte 1) et
– $\lambda(a, b) = 2$ (toute substitution coûte 2).

Pour tous mots x, y, $d_{\mathrm{E}}^{\lambda}(x, y)$ est le nombre minimum d'insertions/délétions nécessaires pour transformer x en y. En effet, pour toutes lettres distinctes a et b on a $\lambda(a, \varepsilon) + \lambda(\varepsilon, b) = 2 = \lambda(a, b)$ donc toute substitution de a en b dans une SÉÉ peut être remplacée par une délétion de a suivie d'une insertion de b sans augmentation de coût.

On a vérifie facilement que $d_{\mathrm{E}}^{\lambda}(x, y) = |x| + |y| - 2 \times \mathrm{lcs}(\{x, y\})$ [4]. Plus précisément, soit

$$\begin{bmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_n \\ \beta_1 & \beta_2 & \cdots & \beta_n \end{bmatrix}$$

un alignement de (x, y) de λ-score minimum. Pour tout $i \in [1, n]$ posons

$$\varsigma_i := \begin{cases} \alpha_i & \text{si } \alpha_i = \beta_i \\ \varepsilon & \text{sinon} \end{cases} .$$

Alors $\varsigma_1 \varsigma_2 \dots \varsigma_n$ est une plus longue sous-séquence commune de $\{x, y\}$.

Exemple 1.37 (Suite de l'exemple 1.31 p. 19) *Pour* $x :=$ `ATCTGAT` *et* $y :=$ `TGCATA`*, on a* $d_{\mathrm{E}}^{\lambda}(x, y) = |x| + |y| - 2 \times \mathrm{lcs}(\{x, y\}) = 7 + 6 - 2 \times 4 = 5$ *donc*

$$\begin{bmatrix} \mathtt{A} & \mathtt{T} & - & \mathtt{C} & - & \mathtt{T} & \mathtt{G} & \mathtt{A} & \mathtt{T} \\ - & \mathtt{T} & \mathtt{G} & \mathtt{C} & \mathtt{A} & \mathtt{T} & - & \mathtt{A} & - \end{bmatrix} \quad \text{et} \quad \begin{bmatrix} \mathtt{A} & \mathtt{T} & \mathtt{C} & \mathtt{T} & \mathtt{G} & - & \mathtt{A} & \mathtt{T} & - \\ - & \mathtt{T} & - & - & \mathtt{G} & \mathtt{C} & \mathtt{A} & \mathtt{T} & \mathtt{A} \end{bmatrix}$$

sont des alignements de (x, y) *de* λ*-score minimal. On extrait respectivement de ces alignements les plus longues sous-séquences communes* `TCTA` *et* `TGAT` *de* $\{x, y\}$.

1.5.4 Calcul de la distance d'édition

On suppose que, pour tous α, $\beta \in \Sigma \cup \{\varepsilon\}$ donnés, $\pi(\alpha, \beta)$ est un entier calculable en temps constant. Alors, étant donnés deux mots x, $y \in \Sigma^{\star}$, on peut calculer un alignement de (x, y) de π-score minimum en temps $O(|x|\,|y|)$ et en espace linéaire [45] grâce à la méthode de Hirschberg [48].

Notons que dans le cas où l'alphabet-réservoir Σ est fini, ce calcul peut être effectué en temps légèrement sous-quadratique [26], mais en espace super-linéaire.

Notons que la complexité algorithmique de nombreuses variantes de la distance d'édition, plus pertinentes biologiquement, a été étudiée. Typiquement, on considère, en plus des OÉÉ, de nouvelles opérations d'édition affectées de coûts spécifiques. Par exemple, on peut autoriser des insertions/délétions de sous-chaînes non réduites à une seule lettre. On attribue à ces opérations un coût fonction des longueurs des mots insérés/effacés [25, 43, 86]. Un autre exemple, consiste à autoriser des *amplifications/contractions* de lettres [12]. Ces opérations sont affectées de coûts inférieurs à ceux des insertions/délétions des lettres correspondantes. Étant donnés une lettre a et un entier $k \geq 0$, on appelle *amplification* (resp. *contraction*) de a d'ordre $k \geq 0$, l'opération consistant à remplacer dans un mot une occurrence de a (resp. de a^k) par a (resp. par a^k).

Chapitre 2

Complexité des problèmes du centre et de la médiane pour la distance d'édition

2.1 Introduction

Pairwise alignment whispers...
... multiple alignment shouts out loud.
Arthur Lesk [51]

Extraction de motifs communs. Un principe fondamental de la bio-informatique est que deux séquences biologiques similaires partagent en général des propriétés communes (nous reviendrons sur ce point à la section 4.1.1.1). Malheureusement, la réciproque est fausse, ce qui est assez contre intuitif. En particulier, il est vérifié empiriquement que des protéines semblables du point de vue de la structure secondaire, de la structure tertiaire ou de la fonction peuvent avoir des structures primaires très différentes : les séquences correspondantes peuvent ne partager que peu de motifs communs, voire pas du tout. Notons que ceci laisse à penser que la nature s'encombre de redondances, inutiles au premier abord. Le problème de l'extraction des motifs communs caractérisant une famille de protéines donnée est une tâche importante en bio-informatique.

Signature. Les motifs communs extraits d'une famille de protéines (comme celle des hémoglobines ou celle des anticorps) servent à lui fabriquer une *signature*. Typiquement, cette signature prend la forme d'une expression régulière. Étant donnée une nouvelle séquence d'acides aminés, on peut déterminer la famille à laquelle elle appartient en la comparant aux signatures des diverses familles de protéines connues. Il existe pour cela des banques de signatures en ligne comme PROSITE (`http://www.expasy.org/prosite/`) ou PFam (`http://www.sanger.ac.uk/Software/Pfam/`). On pourra aussi consulter [8] pour d'autres exemples.

De la nécessité de comparer plusieurs séquences. Pour extraire les motifs communs les plus caractéristiques possible d'une famille de séquences, il faut comparer un nombre important de séquences appartenant à la famille considérée. En effet, les motifs recherchés sont souvent petits et peu nombreux devant la longueur des séquences. Donc, sur un nombre de séquences trop faible, on risque de ne pas les distinguer ou de les confondre avec des motifs communs apparaissant par hasard.

Alignement multiple. L'alignement (voir définition page 21) est la formalisation la plus courante de la comparaison de séquences. Il existe de nombreux logiciels dédiés à la construction d'alignements multiples, comme par exemple, MSA [75], MACAW [103], ClustalW [110], MUSCLE [30], *etc.* Hormis MSA, ces logiciels sont de nature heuristique.

Difficulté du problème de l'extraction des motifs communs. D'une part, rechercher des motifs communs est rendu difficile par la complexité de la forme des motifs caractérisant les familles de séquences étudiées en pratique. Les méthodes à disposition sont parfois trop rudimentaires pour extraire une quelconque information commune de séquences correspondant pourtant à des protéines possédant des propriétés biologiques semblables.

D'autre part, la complexité du calcul d'un alignement optimal explose en fonction du nombre de séquences traitées, pour la plupart des fonctions objectives envisagées. Ces fonctions attribuent un score à chaque alignement. De leur nature dépendent les motifs communs que l'on cherche à découvrir dans les alignements de meilleur score. Les fonctions de score les plus intéressantes tiennent compte de la distance évolutive entre les séquences étudiées. On trouvera dans la section 2.4 un survol des divers résultats connus sur les problèmes d'alignement correspondants. De plus, dans la section 2.5.2, nous montrons que l'un des plus connus de ces problèmes d'optimisation d'alignement est difficile à résoudre exactement, même si on se restreint à un alphabet d'entrée binaire.

Médiane, centre et consensus. Supposons donné un ensemble fini W de mots. Dans ce chapitre, nous étudions la complexité de deux problèmes d'optimisation.

1. Le problème de la recherche d'un mot μ minimisant la somme des distances d'édition aux mots de W.
2. Le problème de la recherche d'un mot g minimisant le maximum des distances d'édition aux mots de W.

Le mot μ est une *médiane* et le mot g est un *centre* de W. Chacun de ces mots peut être considéré comme un consensus approché de W dans le sens où il contient des motifs communs aux mots de W. Notons que le problème de la médiane peut être vu comme un problème d'alignement multiple (voir section 2.4.4.1).

2.2 Position des problèmes et exemples

Dans cette section, nous introduisons les problèmes du centre et de la médiane sous leur forme la plus générale. Nous donnons plusieurs exemples et démontrons quelques résultats élémentaires. Dans toute la suite, U désigne un ensemble quelconque et d désigne une application $U \times U \to \mathbb{Z} \cup \{\infty\}$, appelée *distance*. Dans la suite, la distance d sera souvent une semi-métrique ou une métrique (voir section 1.1.1), mais pas toujours.

2.2.1 Médiane

Définition 2.1 (Médiane)
Pour toute partie finie $W \subseteq U$, on pose :

$$\mathrm{S}_d(W) := \inf_{\mu \in U} \left(\sum_{w \in W} d(\mu, w) \right)$$

et on appelle médiane *de W pour la distance d, tout élément $\mu \in U$ tel que $\sum_{w \in W} d(\mu, w) = \mathrm{S}_d(W)$.*

Comme d ne prend que des valeurs entières, il suffit que la fonction d soit minorée pour que toute partie finie de U admette une médiane pour d. En effet, la borne inférieure d'un ensemble d'entiers minoré est son plus petit élément. Par exemple, toute semi-métrique est minorée par 0 (remarque 1.2 p. 8).

Exemple 2.1 (Un exemple facile) *Plaçons-nous dans le cas où l'on a $U = \mathbb{Z}$ et $d(x, y) = |x - y|$ pour tous x, $y \in \mathbb{Z}$. Soient x, μ, $y \in \mathbb{Z}$ avec $x < \mu < y$, $W_1 := \{x, y\}$ et $W_2 := \{x, \mu, y\}$.*

Alors, on a $\mathrm{S}_d(W_1) = \mathrm{S}_d(W_2) = y - x$. Pour la distance d, $[x, y]$ est l'ensemble des médianes de W_1 alors que μ est l'unique médiane de W_2.

Étudions brièvement ce que peuvent être les médianes d'un ensemble à deux éléments.

Proposition 2.1
On suppose que d est une semi-métrique sur U. Soient x, $y \in U$.

Alors, x et y sont des médianes de $\{x, y\}$ pour la distance d et $\mathrm{S}_d(\{x, y\}) = d(x, y)$.

Preuve. En utilisant la symétrie de d et l'inégalité triangulaire, on obtient, pour tout $\mu \in U$:

$$\sum_{w \in \{x,y\}} d(\mu, w) = d(\mu, x) + d(\mu, y) = d(x, \mu) + d(\mu, y) \geq d(x, y)$$

d'où $\mathrm{S}_d(\{x, y\}) \geq d(x, y)$. D'autre part, toujours en utilisant les propriétés de semi-métrique de d, il vient aussi :

$$\mathrm{S}_d(\{x, y\}) \leq \sum_{w \in \{x,y\}} d(x, w) = d(x, x) + d(x, y) = d(x, y)\,.$$

On en déduit que $\mathrm{S}_d(\{x, y\}) = d(x, y)$ et que x est bien une médiane de $\{x, y\}$ pour d (le cas de y est symétrique). **Q.E.D.**

Exemple 2.2 (Médianes d'une paire de mots) *Soient Σ un alphabet quelconque, et π une semi-métrique sur $\Sigma \cup \{\varepsilon\}$. Soient x, $y \in \Sigma^\star$ et $W := \{x, y\}$. Alors, les médianes de W pour la distance d'édition d_{E}^{π} sont les mots sur Σ apparaissant dans une SÉÉ transformant x en y, de coût pondéré par π minimum.*

Soient un entier $N \geq 1$ et $W_3 := \{\mathtt{0}^N, \mathtt{1}^N\}$. Caractérisons les médianes de W_3 pour d_{L}. Les seules OÉÉs apparaissant dans une SÉÉ de longueur minimale transformant $\mathtt{0}^N$ en $\mathtt{1}^N$ sont des substitutions de $\mathtt{0}$ en $\mathtt{1}$: chacune des N lettres de $\mathtt{0}^N$ est substituée par un $\mathtt{1}$, ces opérations pouvant s'effectuer dans un ordre quelconque. Ainsi, on a $\mathrm{S}_{d_{\mathrm{L}}}(W_3) = d_{\mathrm{L}}(\mathtt{0}^N, \mathtt{1}^N) = N$ et les médianes de W_3 pour d sont les mots sur $\{\mathtt{0}, \mathtt{1}\}$ de longueur N.

Étant donné un langage fini W dont tous les mots ont même longueur, on peut facilement calculer toutes les médianes de W pour la distance de Hamming.

Proposition 2.2 (Médianes pour la distance de Hamming)
Soient un entier $L \geq 1$, un ensemble fini W de mots de longueur L et un mot μ de longueur L.

Alors, μ est une médiane de W pour la distance de Hamming si et seulement si pour tout $i \in [1, L]$, la i-ème lettre $\mu[i]$ de μ maximise le nombre des $w \in W$ tels que $w[i] = \mu[i]$.

Preuve. Notons δ la métrique discrète sur l'ensemble de tous les symboles de l'univers : pour toutes lettres a, b avec $a \neq b$, on a $\delta(a, a) = 0$ et $\delta(a, b) = 1$.

Pour tous mots x, y de longueur L, on a

$$d_{\mathrm{H}}(x, y) = \sum_{i=1}^{L} \delta(x[i], y[i])$$

et, pour toute lettre a et tout indice $i \in [1, L]$, on a

$$\sum_{w \in W} \delta(a, w[i]) = \#\{w \in W : w[i] \neq a\} = m - \#\{w \in W : w[i] = a\}$$

où m désigne le cardinal de W. Pour tout mot μ de longueur L, on peut alors écrire :

$$\begin{aligned}\sum_{w \in W} d_{\mathrm{H}}(\mu, w) &= \sum_{w \in W} \left(\sum_{i=1}^{L} \delta(\mu[i], w[i]) \right) = \sum_{i=1}^{L} \left(\sum_{w \in W} \delta(\mu[i], w[i]) \right) \\ &= Lm - \sum_{i=1}^{L} \#\{w \in W : w[i] = \mu[i]\}\,.\end{aligned}$$

Ainsi, le mot μ minimise $\sum_{w \in W} d_{\mathrm{H}}(\mu, w)$ si et seulement si, pour chaque $i \in [1, L]$, $\mu[i]$ maximise $\#\{w \in W : w[i] = \mu[i]\}$. **Q.E.D.**

Exemple 2.3 *Soient un entier $N \geq 0$, un entier $\sigma \geq 2$ et σ lettres a_1, a_2, ..., a_σ distinctes de $\mathtt{0}$. On pose :*

$$W_4 := \left\{ \mathtt{0}^{N+1}, a_1 \mathtt{1}^N, a_2 \mathtt{1}^N, \ldots, a_\sigma \mathtt{1}^N \right\} .$$

Alors, les médianes de W_4 pour la distance de Hamming sont les $a\mathtt{1}^N$ ($a \in \{\mathtt{0}, a_1, a_2, \ldots, a_\sigma\}$) et $\mathrm{S}_{d_\mathrm{H}}(W_4) = N + \sigma$.

Reprenons maintenant la définition du problème de la médiane donnée dans l'exemple 1.19 p. 15 :

Définition 2.2 (Le problème de la médiane)
Le problème de la médiane *pour la distance d (d-*Median*) s'énonce de la manière suivante :* "Étant donnée une partie finie $W \subseteq U$, trouver un point $\mu \in U$ minimisant $\sum_{w \in W} d(\mu, w)$".

Le seul résultat général connu sur ce problème est un résultat d'approximabilité :

Proposition 2.3 ([45])
*Supposons que d soit une semi-métrique sur U calculable en temps polynomial. Alors, l'algorithme retournant, pour chaque partie finie $W \subseteq U$, un élément $w_* \in W$ minimisant $\sum_{w \in W} d(w_*, w)$, est un algorithme d'approximation pour d-*Median *de borne $2 - \frac{2}{\#W}$.*

Preuve. Comme d est calculable en temps polynomial, l'algorithme décrit est polynomial. Il reste à montrer la validité de la borne annoncée.

Soit $\mu \in U$ tel que $\mathrm{S}_d(W) = \sum_{w \in W} d(\mu, w)$. Soit $w_\mu \in W$ minimisant $d(\mu, w_\mu)$:

$$d(\mu, w_\mu) \leq \frac{\mathrm{S}_d(W)}{\#W} . \tag{2.1}$$

Les propriétés de semi-métrique de d permettent d'écrire :

$$\begin{aligned}
\sum_{w \in W} d(w_*, w) &\leq \sum_{w \in W} d(w_\mu, w) \\
&= \sum_{w \in W \setminus \{w_\mu\}} d(w_\mu, w) \\
&\leq \sum_{w \in W \setminus \{w_\mu\}} \big(d(w, \mu) + d(\mu, w_\mu) \big) \\
&= \sum_{w \in W \setminus \{w_\mu\}} \big(d(\mu, w) + d(\mu, w_\mu) \big)
\end{aligned}$$

et comme :

$$\begin{aligned}
\sum_{w \in W \setminus \{w_\mu\}} \big(d(\mu, w) + d(\mu, w_\mu) \big) &= \sum_{w \in W \setminus \{w_\mu\}} d(\mu, w) + \sum_{w \in W \setminus \{w_\mu\}} d(\mu, w_\mu) \\
&= \big(\mathrm{S}_d(W) - d(\mu, w_\mu) \big) + (\#W - 1) d(\mu, w_\mu) \\
&= \mathrm{S}_d(W) + (\#W - 2) d(\mu, w_\mu)
\end{aligned}$$

on obtient :

$$\sum_{w \in W} d(w_*, w) \leq \mathrm{S}_d(W) + (\#W - 2) d(\mu, w_\mu) .$$

Par suite, l'inégalité (2.1) permet la majoration :

$$\sum_{w \in W} d(w_*, w) \leq \mathrm{S}_d(W) + (\#W - 2) \frac{\mathrm{S}_d(W)}{\#W} = \left(2 - \frac{2}{\#W} \right) \mathrm{S}_d(W)$$

c'est-à-dire exactement la borne annoncée. **Q.E.D.**

2.2.2 Centre

Définition 2.3 (Centre)
Pour toute partie finie $W \subseteq U$, on appelle rayon *de W pour la distance d, la quantité*

$$\mathrm{R}_d(W) := \inf_{g \in U} \left(\max_{w \in W} d(g, w) \right) .$$

On appelle centre *de W pour la distance d, tout élément $g \in U$ tel que $\max_{w \in W} d(g, w) = \mathrm{R}_d(W)$.*

Notons que, si d est minorée alors toute partie finie $W \subseteq U$ admet un centre pour d et, si la distance d est positive alors on a $\mathrm{R}_d(W) \leq \mathrm{S}_d(W)$.

Exemple 2.4 *Prenons pour d la distance usuelle sur $\mathbb{Z}$ comme dans l'exemple 2.1 p. 26. Alors, toute partie finie $W \subseteq \mathbb{Z}$ est de rayon*

$$\mathrm{R}_d(W) = \left\lceil \frac{\max W - \min W}{2} \right\rceil$$

et ses centres pour d sont

$$\left\lceil \frac{\max W + \min W}{2} \right\rceil \qquad \textit{et} \qquad \left\lfloor \frac{\max W + \min W}{2} \right\rfloor .$$

Notons que ces deux entiers sont égaux ou non suivant que $\max W + \min W$ est pair ou impair.

Exemple 2.5 (Centre d'une paire de mots, suite de l'exemple 2.2 p. 27) *La paire de mots $W = \{x, y\}$ est de rayon $\lceil d_{\mathrm{L}}(x, y) \,/\, 2 \rceil$ pour la distance de Levenshtein. De plus, les centres de W pour d_{L} sont les mots g apparaissant en position centrale $\lceil d_{\mathrm{L}}(x, y) \,/\, 2 \rceil$ ou $\lfloor d_{\mathrm{L}}(x, y) \,/\, 2 \rfloor$ dans une SÉÉ de longueur minimum $d_{\mathrm{L}}(x, y) + 1$ transformant x en y.*

Pour la distance de Levenshtein, $W_3 = \left\{ \mathtt{0}^N, \mathtt{1}^N \right\}$ est de rayon $\lceil N \,/\, 2 \rceil$ et, ses centres sont les mots $g \in \{\mathtt{0}, \mathtt{1}\}^\star$ tels que $\left\{ |g|_{\mathtt{0}} , |g|_{\mathtt{1}} \right\} = \left\{ \lceil N \,/\, 2 \rceil , \lfloor N \,/\, 2 \rfloor \right\}$.

Exemple 2.6 (Suite de l'exemple 2.3 p. 28) *Pour fixer les idées, supposons que N soit un entier pair au moins égal à 4. Alors, on a $\mathrm{R}_{d_{\mathrm{H}}}(W_4) = N/2+1$. De plus, les centres de W_4 pour la distance de Hamming, sont :*

1. *les mots de la forme ag avec $a \in \{\mathtt{0}, a_1, a_2, \ldots, a_\sigma\}$ et $g \in \{\mathtt{0}, \mathtt{1}\}^\star$ vérifiant $|g|_{\mathtt{0}} = |g|_{\mathtt{1}} = N \,/\, 2$ et,*
2. *les mots de la forme $\mathtt{0}g$ avec $g \in \{\mathtt{0}, \mathtt{1}\}^\star$ vérifiant $|g|_{\mathtt{0}} = N \,/\, 2 - 1$ et $|g|_{\mathtt{1}} = N \,/\, 2 + 1$.*

Notons que, pour la distance de Hamming, l'ensemble des centres et l'ensemble des médianes de W_4 sont disjoints.

Définition 2.4 (Le problème du centre)
Le problème du centre *pour la distance d (d-*Center*) s'énonce de la manière suivante :* "Étant donnée une partie finie $W \subseteq U$, trouver un point $g \in U$ minimisant $\max_{w \in W} d(g, w)$".

Notons que, sous les hypothèses de la proposition 2.4 ci-dessous, la distance d est positive (remarque 1.2 p. 8) donc toute partie finie de U admet un centre pour d.

Proposition 2.4 ([10])
*On suppose que d est finie, symétrique et satisfait l'inégalité triangulaire. Alors, l'algorithme retournant, pour chaque partie finie $W \subseteq U$, un élément w_1 quelconque de W est un algorithme d'approximation pour d-*Center *de borne constante 2.*

Preuve. L'algorithme décrit est trivialement polynomial. Il reste à montrer la validité de la borne annoncée.

Soit g tel que $\max_{w \in W} d(g, w) = \mathrm{R}_d(W)$. Pour tout $w \in W$, on a

$$d(w_1, w) \leq d(w_1, g) + d(g, w) = d(g, w_1) + d(g, w) \leq \mathrm{R}_d(W) + \mathrm{R}_d(W)$$

d'où

$$\max_{w \in W} d(w_1, w) \leq 2 \cdot \mathrm{R}_d(W)$$

c'est-à-dire exactement la borne annoncée. **Q.E.D.**

2.2.3 Digression : médiane et centre en géométrie

On étudie ici le problème de l'existence et de l'unicité de la médiane et du centre pour les métriques associées à des *normes* et, plus particulièrement, à des normes *euclidiennes*.

On note $\mathbb{R}$ le corps des réels. Dans toute cette section, U désigne un $\mathbb{R}$-espace vectoriel de dimension finie et $\|\cdot\|$ une norme sur U. On prend pour d la métrique sur U associée à $\|\cdot\|$. On autorise ainsi exceptionnellement d à prendre des valeurs réelles non entières. Pour tous $\vec{x}$, $\vec{y} \in U$ et tout $\lambda \in \mathbb{R}$, on a :
- $\|\vec{x}\| = 0$ si et seulement si $\vec{x} = \vec{0}$,
- $\|\lambda\vec{x}\| = |\lambda| \cdot \|\vec{x}\|$ (homogénéité),
- $\|\vec{x} + \vec{y}\| \leq \|\vec{x}\| + \|\vec{y}\|$ (inégalité triangulaire) et
- $d(\vec{x}, \vec{y}) = \|\vec{x} - \vec{y}\|$.

Proposition 2.5 (Existence)
Toute partie finie d'un espace vectoriel normé U de dimension finie admet au moins un centre et au moins une médiane pour la métrique associée à la norme de U.

Preuve. Cette preuve est fondée sur un argument de compacité.

Soit une partie finie $W \subseteq U$. On note f l'application de U vers $\mathbb{R}$ qui, à tout $\vec{x} \in U$, associe

$$f(\vec{x}) := \sum_{\vec{w} \in W} \|\vec{x} - \vec{w}\| \quad \left(\text{resp. } f(\vec{x}) := \max_{\vec{w} \in W} \|\vec{x} - \vec{w}\|\right).$$

L'ensemble des médianes (resp. des centres) de W pour d est l'ensemble des points en lesquels f atteint sa borne inférieure $\inf_{\vec{x} \in U} f(\vec{x})$. Il s'agit de montrer que cet ensemble est non vide.

Soient $\vec{w}_1 \in W$ et $B := \left\{\vec{x} \in U : \|\vec{x}\| \leq f(\vec{0}) + \|\vec{w}_1\|\right\}$. La fonction f est continue sur la boule fermée B qui est compacte donc il existe $\vec{a} \in B$ tel que :

$$f(\vec{a}) = \inf_{\vec{x} \in B} f(\vec{x}). \tag{2.2}$$

Or, pour tout $\vec{x} \in U \setminus B$, on a :

$$f(\vec{x}) \geq \|\vec{x} - \vec{w}_1\| \geq \|\vec{x}\| - \|\vec{w}_1\| > \left(f(\vec{0}) + \|\vec{w}_1\|\right) - \|\vec{w}_1\| = f(\vec{0})$$

et, comme $\vec{0}$ est un élément de B, on a également $f(\vec{a}) \leq f(\vec{0})$. Il en résulte :

$$\forall \vec{x} \in U \setminus B \qquad f(\vec{x}) > f(\vec{a}). \tag{2.3}$$

On déduit des assertions (2.2) et (2.3) que f atteint sa borne inférieure au point $\vec{a}$: $\vec{a}$ est une médiane (resp. un centre) de W pour la métrique associée à la norme de U. **Q.E.D.**

L'exemple suivant montre que, dans un espace normé quelconque, on n'a pas forcément unicité du centre ni de la médiane.

Exemple 2.7 *On suppose que $U := \mathbb{R} \times \mathbb{R}$ et que, pour tout $(x_1, x_2) \in \mathbb{R} \times \mathbb{R}$, on a $\|(x_1, x_2)\| = |x_1| + |x_2|$. On pose $W := \{(-1, -1), (1, 1)\}$.*

1. *On a $\mathrm{R}_d(W) = 2$ et les centres de W pour d sont les couples de la forme $(-x, x)$ avec $-1 \leq x \leq 1$.*
2. *On a de plus $\mathrm{S}_d(W) = 4$ et les médianes de W pour d sont les couples $(x, y) \in \mathbb{R} \times \mathbb{R}$ avec x et y compris entre -1 et $+1$.*

Supposons désormais que la norme $\|\cdot\|$ soit euclidienne. Elle vérifie alors la propriété suivante :

Proposition 2.6 (Cas d'égalité dans l'inégalité de Minkowski [109])
Si $\|\cdot\|$ est une norme euclidienne sur un $\mathbb{R}$-espace vectoriel U alors :

$$\forall \vec{x}, \vec{y} \in U \qquad \|\vec{x} + \vec{y}\| = \|\vec{x}\| + \|\vec{y}\| \iff \|\vec{y}\|\,\vec{x} = \|\vec{x}\|\,\vec{y}.$$

Autrement dit, $\|\vec{x}+\vec{y}\| = \|\vec{x}\| + \|\vec{y}\|$ si et seulement si $\vec{x}$ et $\vec{y}$ sont colinéaires et de même sens. À l'aide de la proposition 2.6 ci-dessus on montre que, dans un espace euclidien :

– tout ensemble fini de points admet un unique centre (proposition 2.8 p. 32), et que
– tout ensemble fini de points non alignés admet une unique médiane (proposition 2.7 p. 31).

Proposition 2.7 (Unicité de la médiane pour la distance euclidienne)
Soient U un espace euclidien et soit une partie finie $W \subseteq U$ telle que les points de U appartenant à W ne soient pas tous alignés.

Alors, W admet une unique médiane pour la métrique associée à la norme de U.

Preuve. L'existence d'une médiane de W pour la métrique d associée à la norme $\|\cdot\|$ de U est garantie par la proposition 2.5 p. 30. Il reste à vérifier l'unicité.

Supposons que W admette deux médianes distinctes $\vec{\mu}_1$ et $\vec{\mu}_2$ pour la métrique d. On va montrer que tous les point de U appartenant à W sont sur la droite passant par $\vec{\mu}_1$ et $\vec{\mu}_2$.

D'une part, on a :

$$\sum_{\vec{w}\in W} \|\vec{\mu}_1 - \vec{w}\| = \sum_{\vec{w}\in W} \|\vec{\mu}_2 - \vec{w}\| = \mathrm{S}_d(W)$$

et

$$\forall \vec{w} \in U \qquad \|(\vec{\mu}_1 - \vec{w}) + (\vec{\mu}_2 - \vec{w})\| \leq \|\vec{\mu}_1 - \vec{w}\| + \|\vec{\mu}_2 - \vec{w}\| \tag{2.4}$$

d'où :

$$\sum_{\vec{w}\in W} \|(\vec{\mu}_1 - \vec{w}) + (\vec{\mu}_2 - \vec{w})\| \leq \sum_{\vec{w}\in W} (\|\vec{\mu}_1 - \vec{w}\| + \|\vec{\mu}_2 - \vec{w}\|) = 2 \cdot \mathrm{S}_d(W)\,. \tag{2.5}$$

D'autre part, posant $\vec{\mu} := \frac{1}{2}(\vec{\mu}_1 + \vec{\mu}_2)$, il vient :

$$\forall \vec{w} \in U \qquad (\vec{\mu}_1 - \vec{w}) + (\vec{\mu}_2 - \vec{w}) = 2(\vec{\mu} - \vec{w})$$

d'où :

$$\sum_{\vec{w}\in W} \|(\vec{\mu}_1 - \vec{w}) + (\vec{\mu}_2 - \vec{w})\| = 2 \sum_{\vec{w}\in W} \|\vec{\mu} - \vec{w}\| \geq 2 \cdot \mathrm{S}_d(W)\,. \tag{2.6}$$

En combinant (2.5) et (2.6), on obtient :

$$\sum_{\vec{w}\in W} \|(\vec{\mu}_1 - \vec{w}) + (\vec{\mu}_2 - \vec{w})\| = \sum_{\vec{w}\in W} (\|\vec{\mu}_1 - \vec{w}\| + \|\vec{\mu}_2 - \vec{w}\|) \tag{2.7}$$

(on obtient également que les deux membres de (2.7) sont égaux à $2 \cdot \mathrm{S}_d(W)$ et à $2\sum_{\vec{w}\in W} \|\vec{\mu} - \vec{w}\|$ mais cela ne sera pas utile). On déduit de (2.4) et (2.7) que, pour tout $\vec{w} \in W$, on a :

$$\|(\vec{\mu}_1 - \vec{w}) + (\vec{\mu}_2 - \vec{w})\| = \|\vec{\mu}_1 - \vec{w}\| + \|\vec{\mu}_2 - \vec{w}\|\ .$$

Il en résulte que les vecteurs $\vec{\mu}_1 - \vec{w}$ et $\vec{\mu}_2 - \vec{w}$ sont liés (proposition 2.6 p. 30) et, par suite, $\vec{w}$ est sur la droite passant par $\vec{\mu}_1$ et $\vec{\mu}_2$. **Q.E.D.**

Examinons rapidement le cas où tous les points de U appartenant à W sont alignés. Alors, il existe $\vec{a}$, $\vec{b} \in U$ avec $\vec{a} \neq \vec{0}$ tels que W soit inclus dans la droite (affine) $D := \left\{\lambda\vec{a} + \vec{b} : \lambda \in \mathbb{R}\right\}$. Posant $m := \#W$, on peut trouver λ_1, λ_2, …, $\lambda_m \in \mathbb{R}$ tels que

$$W = \{w_1, w_2, \ldots, w_m\} \text{ avec } w_i := \lambda_i\vec{a} + \vec{b} \text{ pour tout } i \in [1, m].$$

Orientons alors D dans le sens (par exemple) de $\vec{a}$ et supposons que les points de W sont numérotés dans leur ordre d'apparition sur D. On a ainsi :

$$\lambda_1 < \lambda_2 < \cdots < \lambda_m\,.$$

– Si m est impair alors $w_{(m+1)/2}$ est l'unique médiane de W pour d.

– Si m est pair alors l'ensemble des médianes de W est le segment d'extrémités $w_{\lfloor (m+1)/2 \rfloor}$ et $w_{\lceil (m+1)/2 \rceil}$. Il n'y a pas unicité dans ce cas.

Notons de plus que $\frac{1}{2}(w_1 + w_m)$ est l'unique centre de W. Les preuves des assertions précédentes sont laissées au lecteur.

Proposition 2.8 (Unicité du centre pour la distance euclidienne)
Toute partie finie d'un espace euclidien U admet un unique centre pour la métrique associée à la norme de U.

Preuve. Soit une partie finie $W \subseteq U$. L'existence d'un centre de W pour la métrique d associée à la norme $\|\cdot\|$ de U est garantie par la proposition 2.5 p. 30. Il reste à vérifier l'unicité.

Soient deux centres $\vec{g}_1$ et $\vec{g}_2$ de W. Si $\mathrm{R}_d(W) = 0$ alors W est réduit à un point donc on a immédiatement $W = \{\vec{g}_1\} = \{\vec{g}_2\}$ et $\vec{g}_1 = \vec{g}_2$. Supposons désormais $\mathrm{R}_d(W) > 0$.

Posons $\vec{g} := \frac{1}{2}(\vec{g}_1 + \vec{g}_2)$ et considérons $\vec{v} \in W$ tel que :

$$\|\vec{g} - \vec{v}\| = \max_{\vec{w} \in W} \|\vec{g} - \vec{w}\| \ .$$

On a

$$\|\vec{g} - \vec{v}\| \geq \mathrm{R}_d(W) \tag{2.8}$$

et, comme $\vec{g}_i$ est un centre de W, on a aussi

$$\|\vec{g}_i - \vec{v}\| \leq \mathrm{R}_d(W) \tag{2.9}$$

pour tout $i \in \{1, 2\}$. Il vient alors :

$$\begin{aligned}
2 \cdot \mathrm{R}_d(W) &\leq 2\,\|\vec{g} - \vec{v}\| && \text{par (2.8)} \\
&= \|(\vec{g}_1 - \vec{v}) + (\vec{g}_2 - \vec{v})\| && \text{car } 2(\vec{g} - \vec{v}) = (\vec{g}_1 - \vec{v}) + (\vec{g}_2 - \vec{v}) \\
&\leq \|\vec{g}_1 - \vec{v}\| + \|\vec{g}_2 - \vec{v}\| && \text{par l'inégalité triangulaire} \\
&\leq 2 \cdot \mathrm{R}_d(W) && \text{par (2.9).}
\end{aligned}$$

Ainsi, toutes les inégalités apparaissant ci-dessus sont en fait des égalités. En particulier, on a

$$\|\vec{g}_1 - \vec{v}\| + \|\vec{g}_2 - \vec{v}\| = 2 \cdot \mathrm{R}_d(W) \tag{2.10}$$

et

$$\|(\vec{g}_1 - \vec{v}) + (\vec{g}_2 - \vec{v})\| = \|\vec{g}_1 - \vec{v}\| + \|\vec{g}_2 - \vec{v}\| \ . \tag{2.11}$$

En combinant (2.10) et (2.9) on obtient $\|\vec{g}_1 - \vec{v}\| = \|\vec{g}_2 - \vec{v}\| = \mathrm{R}_d(W)$. Or, l'égalité (2.11) permet d'appliquer la proposition 2.6 p. 30 :

$$\|\vec{g}_1 - \vec{v}\|\,(\vec{g}_2 - \vec{v}) = \|\vec{g}_2 - \vec{v}\|\,(\vec{g}_1 - \vec{v})\,.$$

Il en résulte $\mathrm{R}_d(W)(\vec{g}_1 - \vec{v}) = \mathrm{R}_d(W)(\vec{g}_2 - \vec{v})$ avec $\mathrm{R}_d(W) > 0$ d'où $\vec{g}_1 = \vec{g}_2$. **Q.E.D.**

Exemple 2.8 (Le cas du triangle) *Plaçons-nous dans le cas où U est le plan euclidien usuel et où W est l'ensemble des trois sommets d'un triangle non aplati T.*

La médiane et le centre de W pour la métrique associée à la norme de U sont constructibles géométriquement [107]. On distingue plusieurs configurations suivant la mesure des angles de T.

- *Lorsqu'un angle de T est de mesure supérieure ou égale à* 120 *degrés alors le sommet correspondant est la médiane de W.*
- *Lorsque tous les angles T sont de mesures inférieures à* 120 *degrés, la médiane de W est le* point de Torricelli-Fermat *de T. En se plaçant sur ce point, on voit chacun des trois côtés de T sous un angle de* 120 *degrés.*
- *Si les trois angles du triangle T sont aigus alors le centre de W est le centre du cercle circonscrit à T.*
- *Si le triangle T admet un angle obtus alors le centre de W est le milieu du côté de T opposé à l'angle obtus.*

2.3 Distance de Hamming

Dans cette section, nous donnons les principaux résultats de nature algorithmique connus sur les problèmes du centre et de la médiane pour la distance de Hamming, ainsi que sur leurs généralisations les plus étudiées.

Pour tous mots c, w avec $|c| \leq |w|$, on note

$$d_{\mathrm{S}}(c, w) := \min_{i \in [0, |w|-|c|]} d_{\mathrm{H}}(c, w(i\,;\, i + |c|])$$

la distance de Hamming minimale entre c et une sous-chaîne de w de même longueur que c.

2.3.1 Médiane pour la distance de Hamming

Considérons le problème k-Consensus Pattern : "*Supposons donnés deux entiers k, $L \geq 0$ et un langage fini W ne contenant que des mots de longueur au moins L. On cherche un ensemble C de k mots de longueur L minimisant* $\sum_{w \in W} \min_{c \in C} d_{\mathrm{S}}(c, w)$."

- La restriction de k-Consensus Pattern aux instances (k, L, W) pour lesquelles $k = 1$, est appelée simplement Consensus Pattern.
- La restriction de k-Consensus Pattern aux instances (k, L, W) pour lesquelles W ne contient que des mots de longueur L, est appelée Hamming Median k-Clustering.

La restriction de k-Consensus Pattern aux instances (k, L, W) pour lesquelles on a à la fois $k = 1$ et $|w| = L$ pour tout $w \in W$, correspond au problème de la médiane pour la distance de Hamming[1]. Cette restriction admet un algorithme exact polynomial, comme on peut le déduire de la caractérisation donnée dans la proposition 2.2 p. 27.

Le problème Consensus Pattern$_D$ est NP-complet [73] et W[1]-difficile pour le paramètre $\#W$. Cela reste vrai même si on se restreint aux instances pour lesquelles $\sigma(W) = 2$ [39]. Sa complexité paramétrique est également connue pour d'autres paramètres (par exemple L) [39].

Notons que l'approximabilité de k-Consensus Pattern (resp. Hamming Median k-Clustering) a également fait l'objet de travaux [56] (resp. [93]). En particulier, dès que $\sigma(W)$ et k sont bornés, k-Consensus Pattern admet un P.T.A.S. [56].

2.3.2 Centre pour la distance de Hamming

Considérons le problème k-Closest Substring : "*Supposons donnés deux entiers k, $L \geq 0$ et un langage fini W ne contenant que des mots de longueur au moins L. On cherche un ensemble C de k mots de longueur L minimisant* $\max_{w \in W} \min_{c \in C} d_{\mathrm{S}}(c, w)$."

- La restriction de k-Closest Substring aux instances (k, L, W) pour lesquelles on a $k = 1$ et $|w| = L$ pour tout $w \in W$, est appelée Closest String. Ce problème correspond au problème du centre pour la distance de Hamming.
- La restriction de k-Closest Substring aux instances (k, L, W) pour lesquelles $k = 1$ est appelée simplement Closest Substring.
- La restriction de k-Closest Substring aux instances (k, L, W) pour lesquelles W ne contient que des mots de longueur L est appelée Hamming Radius k-Clustering.

Le problème de décision Closest String$_D$ est NP-complet même si on le restreint aux instances telles que $\sigma(W) = 2$ [67]. Il en est donc de même pour Closest Substring$_D$, k-Closest Substring$_D$ et Hamming Radius k-Clustering$_D$.

2.3.2.1 Complexité paramétrique

La complexité paramétrique des problèmes (k-)Closest Substring$_D$ et Closest String$_D$ est connue pour la plupart des paramètres intéressants [35, 36, 39, 44].

[1] Cette restriction de k-Consensus Pattern correspond en fait à la restriction de d_{H}-Median aux instances W qui sont des langages dont tous les mots sont de même longueur.

En particulier, même si $\sigma(W)$ n'est pas borné, Closest String est F.P.T. pour le paramètre $\#W$ [44]. Ceci contraste à la fois avec le problème du centre pour la distance d'édition (voir théorèmes 2.2 p. 42 et 2.5 p. 51) et avec Closest Substring [39]. En effet, ces deux derniers problèmes sont W[1]-difficiles pour le paramètre $\#W$ même si on les restreint aux instances pour lesquelles $\sigma(W) = 2$.

Par ailleurs, étant donnés un ensemble fini W de mots de longueur L et un entier $d \geq 0$, on peut trouver un mot g de longueur L vérifiant $\max_{w \in W} d_{\mathrm{H}}(g, w) \leq d$ en temps $O\big(|W| + \#W \times d^{d+1}\big)$ si un tel mot g existe [44].

2.3.2.2 Approximabilité

L'approximabilité de (k-)Closest Substring, Hamming Radius k-Clustering et Closest String a également été étudiée [74, 72, 56, 41, 67, 34]. Notamment, dès que $\sigma(W)$ et k sont bornés, k-Closest Substring admet un P.T.A.S. [56]. En revanche, dans le cas où k est non borné, il n'existe pas d'algorithme d'approximation pour Hamming Radius k-Clustering de borne constante strictement plus petite que 2 sauf si NP = P ; de plus, ce dernier résultat reste vrai même si on se restreint aux instances pour lesquelles $\sigma(W) = 2$ [41, 56].

2.4 Problèmes d'alignement multiple

Dans cette section, nous passons en revue les principaux résultats connus sur les problèmes d'optimisation combinatoire liés à l'alignement multiple.

2.4.1 Un algorithme général

Nous montrons ici que, pour toute fonction de score prise dans une large classe, il existe un algorithme calculant un alignement optimal de tout m-uplet de mots pris en entrée, en un temps polynomial à m fixé.

2.4.1.1 Position du problème

Soit Π une application associant un coût entier à chaque alignement-colonne (un *alignement-colonne* est un alignement possédant une unique colonne). Dans certaines applications (voir section 2.4.2 par exemple), seul les images par Π des alignements-colonnes à entrées dans un certain alphabet fini étendu $\Sigma \cup \{\varepsilon\}$ présenteront un intérêt.

Pour tout alignement

$$A = \begin{bmatrix} \alpha_{1,1} & \alpha_{1,2} & \cdots & \alpha_{1,n} \\ \alpha_{2,1} & \alpha_{2,2} & \cdots & \alpha_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{m,1} & \alpha_{m,2} & \cdots & \alpha_{m,n} \end{bmatrix},$$

on appelle Π-*score* de A la quantité

$$\sum_{j=1}^{n} \Pi \left(\begin{bmatrix} \alpha_{1,j} \\ \alpha_{2,j} \\ \vdots \\ \alpha_{m,j} \end{bmatrix} \right).$$

Le Π-score de A est donc la somme des Π-scores de ses colonnes. On dit que A est Π-*optimal* lorsque A est de Π-score minimum parmi tous les alignements de $(w_1, w_2, \ldots, w_m)$ (où $w_i := \alpha_{i,1}\alpha_{i,2}\ldots\alpha_{i,n}$ pour tout $i \in [1, m]$). Dans cette section, nous décrivons un algorithme exact pour le problème d'optimisation suivant : "*Étant donnés m mots w_1, w_2, …, w_m, trouver un alignement de $(w_1, w_2, \ldots, w_m)$ de Π-score minimum.*" Il peut sembler incongru de nommer "score" une quantité que l'on cherche à minimiser. Cet usage est néanmoins répandu chez les Informaticiens travaillant sur la complexité des problèmes d'alignement multiple. De toute façon, on peut se ramener à un problème de maximisation de score en changeant Π en $-\Pi$. On fait l'hypothèse suivante sur la complexité de la fonction Π. On suppose qu'il existe une fonction $\varphi : \mathbb{N} \to \mathbb{N}$ telle que : pour

chaque alignement-colonne C à m entrées, on peut calculer en temps $O(\varphi(m))$ l'image de C par Π. Ainsi, on peut borner la complexité d'un algorithme calculant Π uniquement en fonction du nombre de lignes de l'alignement-colonne pris en argument, indépendamment de la nature de ses entrées. Cette hypothèse est raisonnable car, en pratique, une lettre quelconque est encodable dans un mot-machine donc la taille d'un alignement-colonne à m lignes est $O(m)$.

Il est également raisonnable de supposer que φ est à croissance au moins linéaire ($\varphi(m) = \Omega(m)$). Alors, la complexité de notre algorithme est

$$O\left(\varphi(m)2^m \prod_{i=1}^{m}(|w_i|+1)\right), \tag{2.12}$$

donc polynomiale pour m fixé.

Exemple 2.9 (Plus longue sous-séquence commune) *On suppose que* Π *associe*
- -1 *à chaque alignement-colonne dont toutes les entrées sont identiques et*
- 0 *à tout autre argument.*

Alors, le Π*-score d'un alignement* Π*-optimal de* $(w_1, w_2, \ldots, w_m)$ *est égal à* $-\mathrm{lcs}(\{w_1, w_2, \ldots, w_m\})$*. De plus, si* A *est un alignement* Π*-optimal de* $(w_1, w_2, \ldots, w_m)$ *alors, en effaçant les colonnes de* A *de* Π*-score nul, on obtient un alignement de* m *exemplaires d'une plus longue sous-séquence commune de* $\{w_1, w_2, \ldots, w_m\}$*. On retrouve ainsi le fait que l'on peut calculer une plus longue sous-séquence commune à un langage de cardinal* m *en un temps polynomial à* m *fixé.*

2.4.1.2 Description de l'algorithme

L'algorithme que nous décrivons ici généralise à un nombre quelconque m de mots, l'algorithme quadratique classique (voir section 1.5.4) calculant la distance d'édition entre 2 mots donnés. Il est fondé sur l'observation :

Proposition 2.9 (Principe d'optimalité)
Soient A *un alignement* Π*-optimal et* A' *l'alignement obtenu à partir de* A *en effaçant sa colonne la plus à droite. Alors,* A' *est également* Π*-optimal.*

La proposition 2.9 p. 35 va nous permettre d'utiliser le principe de *programmation dynamique*. Rappelons que pour tout mot w et tout $p \in [0, |w|]$, $w^{(p)}$ désigne le préfixe de w de longueur p. Posons $\mathbf{w} := (w_1, w_2, \ldots, w_m)$ et

$$P := [0, |w_1|] \times [0, |w_2|] \times \cdots \times [0, |w_m|] \ .$$

Pour tout vecteur $\mathbf{p} = (p_1, p_2, \ldots, p_m) \in P$, notons

$$\mathbf{w}^{(\mathbf{p})} := \left(w_1^{(p_1)}, w_2^{(p_2)}, \ldots, w_m^{(p_m)}\right)$$

et $T[\mathbf{p}]$ le Π-score d'un alignement Π-optimal de $\mathbf{w}^{(\mathbf{p})}$. Par définition, $T[(|w_1|, |w_2|, \ldots, |w_m|)]$ est le Π-score d'un alignement Π-optimal de $\mathbf{w}$. Notre algorithme calcule dynamiquement *toutes* les entrées de la table m-dimensionnelle $T[\,\cdot\,]$ à l'aide de la relation de récurrence (2.13) établie ci-après.

Relation de récurrence. Soit $\mathbf{p} \in P$ et soit C un alignement-colonne à m entrées. Alors, C est la colonne la plus à droite d'un alignement de $\mathbf{w}^{(\mathbf{p})}$ si et seulement si pour tout $i \in [1, m]$, l'entrée située sur la i-ème ligne de C est, soit $w_i[p_i]$, soit un $-$. Notons que, si $p_i = 0$ alors $w_i^{(p_i)}$ est le mot vide ; dans ce cas, l'entrée située sur la i-ème ligne de la colonne la plus à droite d'un alignement de $\mathbf{w}^{(\mathbf{p})}$ est nécessairement un $-$.

Ceci nous incite à considérer, pour tout vecteur binaire $\mathbf{e} = (e_1, e_2, \ldots, e_m) \in \{0,1\}^m$, la matrice colonne $C_{\mathbf{p},\mathbf{e}}$ à m entrées donnée par : pour tout $i \in [1, m]$, l'entrée de $C_{\mathbf{p},\mathbf{e}}$ située sur sa i-ème ligne est égale à
- $w_i[p_i]$ si $e_i = 1$ et $p_i \neq 0$,
- $-$ sinon.

Notons également $E_{\mathbf{p}}$ l'ensemble des $\mathbf{e} = (e_1, e_2, \ldots, e_m) \in \{0,1\}^m$ pour lesquels il existe un $i \in [1,m]$ vérifiant $e_i = 1$ et $p_i \neq 0$. Notons que $C_{\mathbf{p},\mathbf{e}}$ admet une entrée distincte de $-$ si et seulement si $\mathbf{e} \in E_{\mathbf{p}}$.

Ainsi, C est la colonne la plus à droite d'un alignement de $\mathbf{w}^{(\mathbf{p})}$ si et seulement s'il existe $\mathbf{e} \in E_{\mathbf{p}}$ tel que $C = C_{\mathbf{p},\mathbf{e}}$. Par ailleurs, supposons que $C_{\mathbf{p},\mathbf{e}}$ soit la colonne la plus à droite d'un certain alignement de $\mathbf{w}^{(\mathbf{p})}$. Alors, en effaçant cette colonne, on obtient un alignement de $\mathbf{w}^{(\mathbf{p}-\mathbf{e})}$ qui, par la proposition 2.9 p. 35, est Π-optimal si l'alignement de départ est Π-optimal.

On en déduit que pour tout $\mathbf{p} \in P$ avec $\mathbf{p} \neq (0,0,\ldots,0)$ on a :

$$T[\mathbf{p}] = \min_{\mathbf{e} \in E_{\mathbf{p}}} \left(T[\mathbf{p} - \mathbf{e}] + \Pi(C_{\mathbf{p},\mathbf{e}}) \right) . \tag{2.13}$$

Algorithme et complexité. Notre algorithme procède de la manière suivante.

1. On initialise $T[(0,0,\ldots,0)]$ à 0 car l'unique alignement de $\mathbf{w}^{((0,0,\ldots,0))} = (\varepsilon, \varepsilon, \ldots, \varepsilon)$ est la matrice à m lignes et 0 colonne dont le Π-score est nul par définition.
2. On énumère les $\mathbf{p} \in P \setminus \{(0,0,\ldots,0)\}$ dans l'ordre (par exemple) lexicographique et on calcule $T[\mathbf{p}]$ à l'aide de la formule (2.13) à partir d'entrées de $T[\,\cdot\,]$ déjà calculées.
3. Par la méthode standard du "*backtracking*" (ou *retour arrière*) on récupère un alignement Π-optimal de $\mathbf{w}$.

Il y a $\#P = \prod_{i=1}^{m} (|w_i| + 1)$ entrées de T à calculer et pour chaque $\mathbf{p} \in P$, le calcul de $T[\mathbf{p}]$ prend un temps $O(\varphi(m) \times \#E_{\mathbf{p}}) = O(\varphi(m) 2^m)$. Notre algorithme a donc bien la complexité indiquée en (2.12) (le temps nécessaire au "backtracking" final est $O(m \sum_{i=1}^{m} |w_i|)$, donc asymptotiquement négligeable).

2.4.2 Somme de toutes les paires

Jusqu'à la fin de la section 2.4, Σ désigne un alphabet fini et $\pi : (\Sigma \cup \{\varepsilon\}) \times (\Sigma \cup \{\varepsilon\}) \to \mathbb{Z}$ une fonction de pondération.

Soit $A = \left[\alpha_{i,j}\right]_{(i,j) \in [1,m] \times [1,n]}$ un alignement à entrées dans Σ possédant m lignes et n colonnes. On appelle *SP-score de A pondéré par π* (SP pour Sum of all Pairs) la quantité

$$\sum_{h=1}^{n} \sum_{i=1}^{m} \sum_{j=1}^{m} \pi(\alpha_{i,h}, \alpha_{j,h}) .$$

On nomme SP^π-MSA (MSA pour Multiple Sequence Alignement) le problème : "*Étant donnés m mots w_1, w_2, …, $w_m \in \Sigma^\star$ trouver un alignement de $(w_1, w_2, \ldots, w_m)$ de SP-score pondéré par π minimum.*" La complexité de SP^π-MSA a été étudiée [114, 15, 58, 32], ainsi que son approximabilité [7, 58, 82].

Algorithme exact. SP^π-MSA admet un algorithme exact dont la complexité est polynomiale à m fixé. En effet, il suffit de prendre l'algorithme de la section 2.4.1 correspondant à une fonction de score Π vérifiant :

$$\Pi\left(\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_m \end{bmatrix}\right) := \sum_{i=1}^{m} \sum_{j=1}^{m} \pi(\alpha_i, \alpha_j)$$

pour tout entier $m \geq 1$ et tous α_1, α_2, …, $\alpha_m \in \Sigma \cup \{\varepsilon\}$ avec $\alpha_1 \alpha_2 \ldots \alpha_m \neq \varepsilon$. Notons que les valeurs de Π pour les alignements-colonnes dont les entrées ne sont pas toutes dans $\Sigma \cup \{\varepsilon\}$ ne jouent aucun rôle.

Complexité. Pour que le problème de décision associé à SP^π-MSA soit NP-complet, il suffit que Σ soit de cardinal 2 et que π soit une métrique sur $\Sigma \cup \{\varepsilon\}$ [32]. À notre connaissance, il n'existe aucun résultat concernant la complexité paramétrique de SP^π-MSA.

Approximabilité. Si π est une semi-métrique, alors il existe, pour chaque entier $l \geq 0$, un algorithme d'approximation pour SP$^\pi$-MSA de borne $2 - l \,/\, m$ [7]. En revanche, lorsque Σ est de cardinal 3, il existe une semi-métrique π_3 sur $\Sigma \cup \{\varepsilon\}$ telle que : si SP$^{\pi_3}$-MSA admet un P.T.A.S. alors NP = P [58].

2.4.3 Alignement phylogénétique

2.4.3.1 Arbre phylogénétique

La *phylogénie* est la science qui a pour but de reconstruire l'histoire évolutive des espèces. Nous définissons ici le concept d'arbre phylogénétique enraciné et expliquons sa signification biologique.

Graphe orienté. Un *graphe orienté* est un couple $G = (V, A)$ où V est un ensemble fini quelconque et où A est une partie de $V \times V$. Les éléments de V (resp. de A) sont appelés *sommets* (resp. *arcs*) de G. Soit $r \in V$: r est une *racine* de G lorsque pour tout $v \in V$, il existe un entier $n \geq 0$ et des sommets v_0, v_1, …, $v_n \in V$ vérifiant $r = v_0$, $v_n = v$ et $(v_{i-1}, v_i) \in A$ pour tout $i \in [1, n]$. Pour tout $v \in V$, le *degré sortant* de v dans G est le nombre des $w \in V$ tels que $(v, w) \in A$. Le *degré sortant maximum* de G est l'entier $\Delta^+(G) := \max_{v \in V} \#\{w \in V : (v, w) \in A\}$.

Arbre enraciné. Un *arbre enraciné* non vide est un graphe orienté $T = (V, A)$ admettant une racine et tel que $\#A = \#V - 1$. Une *feuille* (resp. un *nœud interne*) de T est un sommet dont le degré sortant dans T est nul (resp. non nul). On dit que T est un *arbre phylogénétique* lorsque T n'admet pas de sommet de degré sortant égal à 1. On dit que T est un *arbre-étoile* lorsque T n'admet qu'un seul sommet interne.

Signification biologique. Un arbre phylogénétique a pour fonction de représenter l'histoire évolutive d'un ensemble d'espèce. Dans un arbre phylogénétique T, les feuilles de T représentent les espèces actuelles étudiées. Les sommets internes de T représentent des espèces éteintes. La topologie de T représente l'histoire des événements de spéciation qui ont permis d'engendrer les espèces représentées par les sommets de T à partir de leur ancêtre commun représenté par la racine de T.

2.4.3.2 Description des problèmes

On nomme Tree$^\pi$-MSA le problème : "*On suppose donnés un arbre phylogénétique $T = (V, A)$ et une application ℓ_0 associant un mot sur Σ à chaque feuille de T. On cherche un étiquetage $\ell : V \to \Sigma^\star$ prolongeant ℓ_0 tel que $\sum_{(u,v) \in A} d_{\mathrm{E}}^\pi\big(\ell(u), \ell(v)\big)$ soit minimum.*"

Signification biologique. Expliquons la signification biologique de Tree$^\pi$-MSA. Dans ce problème, on suppose connue l'histoire T des événements de spéciation qui ont permis d'engendrer les séquences biologiques $\ell_0(v)$ pour v feuille de T. On cherche à reconstruire les séquences ancestrales $\ell(v)$ pour tout sommet interne v de T. Si l'on se réfère au principe du *maximum de parcimonie* ces séquences ancestrales sont telles que la quantité totale d'évolution $\sum_{(u,v) \in A} d_{\mathrm{E}}^\pi\big(\ell(u), \ell(v)\big)$ est minimale. Le principe du maximum de parcimonie, souvent attribué au moine franciscain et philosophe Guillaume d'Ockham, peut être énoncé grossièrement de la manière suivante : "*L'explication la plus simple est généralement la meilleure*".

Par ailleurs, notons qu'étant données une instance (T, ℓ_0) de Tree$^\pi$-MSA et une solution ℓ de cette instance, il existe une manière canonique de construire un alignement des mots $\ell_0(v)$ pour v feuille de T [101, 115]. Cet alignement est d'autant plus pertinent biologiquement que ℓ est une solution proche de l'optimum.

La variante avec goulot d'étranglement. On considère également la variante de Tree$^\pi$-MSA nommée BTree$^\pi$-MSA (B pour Bottleneck), où l'on cherche un étiquetage $\ell : V \to \Sigma^\star$ minimisant $\max_{(u,v) \in A} d_{\mathrm{E}}^\pi\big(\ell(u), \ell(v)\big)$ au lieu de $\sum_{(u,v) \in A} d_{\mathrm{E}}^\pi\big(\ell(u), \ell(v)\big)$. BTree$^\pi$-MSA a été introduit pour pallier une faiblesse de Tree$^\pi$-MSA [99] : dans les instances rencontrées en pratique, certaines séquences biologiques parmi les étiquettes $\ell_0(v)$ (v feuille de T) peuvent être artificiellement sur-représentées et "attirer" vers elles l'ensemble des étiquettes $\ell(v)$ pour v sommet de T.

2.4.3.3 Complexité des problèmes d'alignement phylogénétique

La complexité [100, 114, 32] et l'approximabilité [54, 113, 115] de TREE$^{\pi}$-MSA ont été étudiées (voir aussi la section 2.4.4.1).

Algorithmes exacts. TREE$^{\pi}$-MSA admet un algorithme exact, polynomial à $\#V$ fixé. En effet, on peut se ramener à l'algorithme décrit dans la section 2.4.1 [101].

Complexité. Si Σ est de cardinal 2 et si π est une métrique sur $\Sigma \cup \{\varepsilon\}$, alors le problème de décision associé à TREE$^{\pi}$-MSA est NP-complet, même si on le restreint aux instances (T, ℓ_0) telles que T soit un arbre binaire (c'est-à-dire tel que $\Delta^+(T) = 2$) [32].

Approximabilité. Si π satisfait l'inégalité triangulaire alors, pour chaque constante $deg \geq 2$, le problème TREE$^{\pi}$-MSA admet un P.T.A.S. dès qu'on le restreint aux instances (T, ℓ_0) vérifiant $\Delta^+(T) = deg$ [113]. En revanche, lorsque Σ est de cardinal 7, on peut trouver une fonction de pondération π_7 telle que : la restriction de TREE$^{\pi_7}$-MSA aux instances (T, ℓ_0) telles que T est un arbre-étoile n'admet pas de P.T.A.S., sauf si NP = P [114].

Alignement phylogénétique avec goulot d'étranglement. Il n'existe à notre connaissance qu'un seul article traitant de BTREE$^{\pi}$-MSA [99] (voir aussi la section 2.4.4.2). On trouve dans ce papier les deux résultats suivants. D'une part, lorsque d_{E}^{π} satisfait l'inégalité triangulaire, BTREE$^{\pi}$-MSA admet un algorithme d'approximation de borne $O(\log(\#V))$. D'autre part, lorsque π est la métrique discrète sur $\Sigma \cup \{\varepsilon\}$, BTREE$^{\pi}$-MSA admet un algorithme exact polynomial à $\#V$ fixé. Notons que ce dernier algorithme n'est que vaguement esquissé.

2.4.4 Médiane et centre pour la distance d'édition

Le but du chapitre présent est d'étudier la complexité des problèmes du centre et de la médiane pour la distance d'édition. Dans cette section nous exposons les résultats déjà connus sur chacun de ces deux problèmes ainsi que notre contribution.

2.4.4.1 Le problème de la médiane pour la distance d'édition

On trouve plusieurs articles traitant de la complexité du problème de la médiane pour la distance d'édition [32, 104, 105, 27, 73] ainsi que de son approximabilité [73, 114]. Notons que d_{E}^{π}-MEDIAN correspond à la restriction de TREE$^{\pi}$-MSA aux instances (T, ℓ_0) pour lesquelles T est un arbre-étoile et ℓ_0 est injective. Ainsi, lorsque Σ est fini, d_{E}^{π}-MEDIAN admet un algorithme exact, polynomial à $\#W$ fixé [100].

Un algorithme exact. Considérons le cas particulier du problème de la médiane pour la distance de Levenshtein d_{L}. Le problème d_{L}-MEDIAN admet un algorithme exact, polynomial à $\#W$ fixé, même si $\sigma(W)$ est non borné. Décrivons succinctement cet algorithme [45]. L'idée est de se ramener à l'algorithme de la section 2.4.1 en prenant pour Π la fonction de score donnée par :

$$\Pi\left(\begin{bmatrix}\alpha_1\\ \alpha_2\\ \vdots\\ \alpha_m\end{bmatrix}\right) := m - \max_{i=1}^{m} \left(\#\left\{j \in [1,m] : \alpha_j = \alpha_i\right\}\right)$$

pour tout entier $m \geq 1$ et tous mots $\alpha_1, \alpha_2, \ldots, \alpha_m$ de longueurs au plus 1 tels que $\alpha_1\alpha_2\ldots\alpha_m \neq \varepsilon$. En général, l'image par Π d'un alignement-colonne à m entrées est calculable en temps $O\big(m^2\big)$: pour chaque $i \in [1,m]$, $\#\left\{j \in [1,m] : \alpha_j = \alpha_i\right\}$ est calculable en temps $O(m)$. Remarquons que, dans le cas particulier où l'on a $\Sigma = [1,\sigma]$ pour un certain entier $\sigma \geq 0$, on peut calculer l'image par Π d'un alignement à m entrées en temps $O(m)$ par une méthode de type "*bucket sort*".

Soit une instance W de d_{L}-Median. Notons $m := \#W$ et écrivons $W = \{w_1, w_2, \ldots, w_m\}$. On applique sur l'instance $(w_1, w_2, \ldots, w_m)$ l'algorithme de la section 2.4.1 correspondant à la fonction Π décrite ci-dessus. On obtient un alignement Π-optimal $A = \big[\alpha_{i,j}\big]_{(i,j)\in[1,m]\times[1,n]}$ de $(w_1, w_2, \ldots, w_m)$ en temps $O\big(m^2 2^m \prod_{i=1}^{m}(|w_i|+1)\big)$.

Il ne reste plus qu'à extraire de A une médiane μ de W pour d_{L} en temps linéaire. On procède d'une manière rappelant la proposition 2.2 p. 27. Pour chaque $j \in [1, n]$, on choisit un mot μ_j de longueur au plus 1 tel que le nombre des indices de lignes $i \in [1, m]$ pour lesquels $\mu_j = \alpha_{i,j}$ soit maximal. Alors, $\mu := \mu_1\mu_2\ldots\mu_n$ est une médiane de W pour d_{L} [45, théorème 14.7.2].

Complexité. Pour que le problème de décision associé à d_{E}^{π}-Median soit NP-complet, il suffit que Σ soit de cardinal 2 et que π soit une métrique sur $\Sigma \cup \{\varepsilon\}$ [32]. De plus, nous montrons que d_{L}-Median$_D$ est W[1]-difficile pour le paramètre $\#W$ même si on se restreint à des langages W sur des alphabets à 2 lettres (voir, dans la section 2.5.2, le théorème 2.3 p. 44).

Approximabilité. Si π est une semi-métrique alors d_{E}^{π}-Median admet un algorithme d'approximation de borne $2 - \frac{2}{\#W}$ [45]. Cet algorithme est décrit dans la proposition 2.3 p. 28.

Lorsque Σ est de cardinal 7, on peut trouver une fonction de pondération π_7 telle que : si $d_{\mathrm{E}}^{\pi_7}$-Median admet un P.T.A.S. alors NP = P [114].

2.4.4.2 Le problème du centre pour la distance d'édition

Le problème du centre pour la distance d'édition est équivalent à la restriction de BTree$^{\pi}$-MSA aux instances (T, ℓ_0) pour lesquelles T est un arbre-étoile.

Un seul article, publié en 2000 par de la Higuera et Casacuberta, traite spécifiquement de ce problème [27]. Il est montré dans ce papier que d_{L}-Center$_D$ est NP-complet même si on se restreint à des langages W sur des alphabets à 4 lettres. Dans la section 2.6 (voir aussi la section 2.5.1), nous améliorons ce résultat. Nous considérons la propriété suivante :

$$\forall a, b \in \Sigma \qquad a \neq b \Rightarrow \pi(a, \varepsilon) < \pi(a, b) + \pi(b, \varepsilon)\,. \tag{$*$}$$

Si Σ est de cardinal 2 et si π est une métrique sur $\Sigma \cup \{\varepsilon\}$ vérifiant la propriété $(*)$, nous montrons que d_{E}^{π}-Center$_D$ est NP-difficile et W[1]-difficile pour le paramètre $\#W$. La propriété $(*)$ est un renforcement naturel de l'inégalité triangulaire. Elle signifie qu'effacer une lettre donnée a dans un mot coûte strictement moins que de substituer a par b puis d'effacer b (quelle que soit la lettre b distincte de a).

Par ailleurs, lorsque Σ est fini, nous montrons (voir section 2.7) qu'il existe un algorithme exact pour d_{E}^{π}-Center, polynomial à $\#W$ fixé. De plus, dès que d_{E}^{π} est symétrique et satisfait l'inégalité triangulaire, d_{E}^{π}-Center admet un algorithme d'approximation de borne constante 2 [10]. Cet algorithme est décrit dans la proposition 2.4 p. 29.

2.5 Intraitabilité des problèmes du centre et de la médiane pour la distance de Levenshtein

Dans cette section, nous considérons les problèmes du centre et de la médiane pour la distance de Levenshtein d_{L}. Nous montrons que les problèmes de décision associés à d_{L}-Center et d_{L}-Median sont :

- NP-complets et
- W[1]-difficiles pour le paramètre "nombre de mots pris en entrée" (c'est-à-dire le cardinal de W).

De plus, nos preuves garantissent que ces résultats restent vrais même si on restreint d_{L}-Center et d_{L}-Median aux instances W qui sont des langages sur des alphabets à 2 lettres (c'est-à-dire telles que $\sigma(W) \leq 2$).

Nous commençons (section 2.5.1) par la moins technique des deux preuves, celle concernant d_{L}-Center$_D$. La preuve pour d_{L}-Median$_D$ (section 2.5.2) s'appuie sur des idées du même genre. Dans les deux cas, on exhibe une réduction depuis LCS$_D$ (voir section 1.4.3).

Notons que nos résultats négatifs contrastent avec certains des résultats correspondants au cas où la distance de Levenshtein est remplacée par la distance de Hamming (voir proposition 2.2 p. 27 et la section 2.3.2.1).

2.5.1 Intraitabilité du problème du centre pour la distance de Levenshtein

Dans cette section, nous démontrons que d_{L}-Center$_D$ est intraitable même si l'on se restreint à des langages W sur des alphabets à 2 lettres (théorème 2.2 p. 42). De la Higuera et Casacuberta avaient obtenus en 2000 un résultat similaire, mais valable seulement pour des alphabets à 4 lettres [27]. Nous suivons le schéma de preuve adopté par ces deux auteurs : nous réduisons LCS_D à d_{L}-Center$_D$ par l'intermédiaire du problème LCS^0_D introduit ci-dessous.

Définition 2.5
On nomme LCS^0_D *la restriction du problème* LCS_D *aux instances* (X, k) *pour lesquelles tous les mots appartenant à* X *sont de longueur* $2k$.

Autrement dit, LCS^0_D consiste à décider si un langage donné X, dont tous les mots sont de longueur $2k$, admet une sous-séquence commune de longueur k.

Pour obtenir l'intraitabilité de LCS^0_D par réduction depuis LCS_D (théorème 2.1 p. 40) nous avons besoin de deux lemmes (lemmes 2.1 p. 40 et 2.2 p. 40).

Le lemme 2.1 p. 40 est trivial. Il garantit qu'en concaténant une même lettre a à la fin (resp. au début) de tous les mots d'un langage X, on augmente de 1 la longueur des plus longues sous-séquences communes de X. En effet, on se convainc facilement que, pour tout mot s, s est une sous-séquence commune de Xa si et seulement si s est une sous-séquence commune de X ou la concaténation d'une sous-séquence commune de X et d'un a. On caractérise de même les sous-séquences communes de aX.

Lemme 2.1 *Pour tout langage non vide* X *et toute lettre* a, *on a* $\mathrm{lcs}(Xa) = \mathrm{lcs}(X) + 1 = \mathrm{lcs}(aX)$.

Étant donné un langage X, le lemme 2.2 ci-dessous donne un procédé permettant de construire un langage Y formé de mots arbitrairement longs mais possédant les mêmes sous-séquences communes que X.

Lemme 2.2 *Soient un langage* X *et deux familles de mots* $\left(z_x\right)_{x\in X}$ *et* $\left(u_x\right)_{x\in X}$ *telles que, pour tout* $x \in X$, z_x *et* u_x *n'aient pas de lettres en commun. Soit* $Y := \bigcup_{x\in X} \left\{xz_x, xu_x\right\}$.
Alors, X *et* Y *ont les mêmes sous-séquence communes.*

Preuve. L'astuce consiste à remarquer que, pour tout mot s, s est une sous-séquence de x si et seulement si s est sous-séquence de xz_x et de xu_x. **Q.E.D.**

Il est montré dans [27, Proposition 1] que la restriction de LCS^0_D aux instances (X, k) vérifiant $\sigma(X) \leq 4$ est NP-complète. À l'aide des deux lemmes précédents, nous pouvons maintenant améliorer ce résultat.

Théorème 2.1
Même si on le restreint aux instances (X, k) *pour lesquelles on a* $\sigma(X) = 2$, *le problème* LCS^0_D *reste* NP*-complet et* W[1]*-difficile pour le paramètre* $\#X$.

Preuve. LCS^0_D est dans NP comme restriction de LCS_D. On va maintenant réduire le problème général LCS_D (paramétré par $\#X$) à sa restriction LCS^0_D (paramétrée par $\#X$) de manière à pouvoir appliquer les résultats donnés dans la section 1.4.3.

Soit (X, k) une instance de LCS_D avec $\sigma(X) = 2$. Sans perte de généralité on peut supposer $X \subseteq \{\mathtt{0}, \mathtt{1}\}^\star$. On calcule $M := \max_{x\in X} |x|$, $L := 2k + M + 1$ puis

$$\hat{X} := \bigcup_{x\in X} \left\{x\mathtt{1}0^M\mathtt{0}^{L-|x|}, x\mathtt{1}0^M\mathtt{1}^{L-|x|}\right\} \quad \text{et} \quad \hat{k} := k + M + 1\,.$$

Par construction tout les mots de $\hat{X}$ sont de longueur $1 + M + L = 2\hat{k}$ donc $(\hat{X}, \hat{k})$ est une instance de LCS^0_D. De plus, la construction de $(\hat{X}, \hat{k})$ à partir de (X, k) peut facilement se réaliser en temps polynomial.

Or, le lemme 2.2 p. 40 garantit $\mathrm{lcs}(\hat{X}) = \mathrm{lcs}(X\mathtt{1}\mathtt{0}^M)$; en appliquant $M+1$ fois le lemme 2.1 p. 40, on obtient donc $\mathrm{lcs}(\hat{X}) = \mathrm{lcs}(X\mathtt{1}\mathtt{0}^M) = \mathrm{lcs}(X) + M + 1$. On en déduit :

$$\mathrm{lcs}(\hat{X}) \geq \hat{k} \iff \mathrm{lcs}(X) \geq k\,.$$

et, par suite, la transformation de (X,k) en $(\hat{X},\hat{k})$ est une réduction de Karp de LCS_D à LCS^0_D.

Vérifions maintenant que le cardinal de $\hat{X}$ ne dépend que du cardinal de X (et pas de la nature des mots appartenant à X, ni de k) :

Lemme 2.3 $\#\hat{X} = 2 \times \#X$.

Preuve. On a trivialement $\#\hat{X} \leq 2 \times \#X$. Il reste donc à montrer que les $x\mathtt{1}\mathtt{0}^M a^{L-|x|}$ ($(x,a) \in X \times \{\mathtt{0},\mathtt{1}\}$) sont deux à deux distincts. Soient deux éléments (x,a), $(y,b) \in X \times \{\mathtt{0},\mathtt{1}\}$. Posons $u := x\mathtt{1}\mathtt{0}^M a^{L-|x|}$, $v := y\mathtt{1}\mathtt{0}^M b^{L-|y|}$ et supposons que $u = v$.

Comme on a $L > M \geq |x|$, la dernière lettre de u est un a. De même la dernière lettre de v est un b. On peut alors déduire de $u = v$ que $a = b$.

D'autre part, l'égalité de u et v garantit aussi que x est préfixe de y ou que y est préfixe de x. Or, si (absurde) x était un préfixe propre de y alors on aurait $v[|y|+1] = \mathtt{1} \neq \mathtt{0} = u[|y|+1]$: contradiction. On exclut de même le cas où y est un préfixe propre de x. La seule possibilité restante est que $x = y$.

On a ainsi montré que $(x,a) = (y,b)$. **Q.E.D.**

Le problème LCS_D reste NP-complet et W[1]-difficile pour le paramètre $\#X$ même si on le restreint aux instances (X,k) pour lesquelles on a $\sigma(X) \leq 2$ [81, 97]. Comme les instances correspondantes $(\hat{X},\hat{k})$ de LCS^0_D que construit notre réduction de Karp vérifient à la fois

- $\#\hat{X} = 2 \times \#X$ (lemme 2.3 p. 41) et
- $\sigma(\hat{X}) = 2$ car $\mathtt{0}$ et $\mathtt{1}$ sont les deux seules lettres apparaissant dans $\hat{X}$,

on a démontré notre théorème 2.1. **Q.E.D.**

Le lemme 2.4 ci-dessous met en exergue le lien entre la distance de Levenshtein et la notion de sous-séquence que nous exploitons pour réduire LCS^0_D à d_{L}-Center$_D$. Pour prouver le lemme 2.4 (ainsi que d'autres résultats par la suite), nous aurons besoin de la remarque suivante qui est une conséquence directe de la remarque 1.3 p. 22 :

Remarque 2.1 *Pour tout mot g, on a $d_{\mathrm{L}}(g,\varepsilon) = d_{\mathrm{L}}(\varepsilon,g) = |g|$.*

Autrement dit, la distance de Levenshtein de ε à tout mot g est égale au nombre de lettres de g. Les plus courtes SÉÉ transformant ε en g (resp. g en ε) sont celles dans lesquelles il n'apparaît que des insertions (resp. des délétions), une pour chaque lettre de g.

Lemme 2.4 *Pour tous mots x, y, on a :*

(i). $d_{\mathrm{L}}(y,x) \geq |y| - |x|$ et

(ii). $d_{\mathrm{L}}(y,x) = |y| - |x|$ si et seulement si x est une sous-séquence de y.

Preuve. Sans perte de généralité, on peut supposer que y est plus long que x.

(i). En combinant l'inégalité triangulaire $d_{\mathrm{L}}(y,\varepsilon) \leq d_{\mathrm{L}}(y,x) + d_{\mathrm{L}}(x,\varepsilon)$ et la remarque 2.1 p. 41, on obtient : $d_{\mathrm{L}}(y,x) \geq d_{\mathrm{L}}(y,\varepsilon) - d_{\mathrm{L}}(x,\varepsilon) = |y| - |x|$.

(ii). Si x est une sous-séquence de y alors on peut obtenir x à partir de y en effaçant $|y| - |x|$ lettres de y. On en déduit $d_{\mathrm{L}}(y,x) \leq |y| - |x|$ et l'inégalité inverse est garantie par le point (i).

Réciproquement, supposons que $d_{\mathrm{L}}(y,x) = |y| - |x|$. Dans toute SÉÉ transformant y en x il doit apparaître au moins $|y| - |x|$ délétions car aucun autre type d'OÉÉ ne fait décroître la longueur. Par suite, l'hypothèse $d_{\mathrm{L}}(y,x) = |y| - |x|$ assure que dans une SÉÉ de longueur *minimale* transformant y en x, il n'apparaît *que* des délétions. On en déduit que x est une sous-séquence de y.

Q.E.D.

Le lemme suivant, conséquence du point (i) du lemme 2.4 p. 41 et de la remarque 2.1 p. 41, sert uniquement à montrer que d_{L}-Center$_D$ (et d_{L}-Median$_D$) sont dans NP.

Lemme 2.5 *Soit W un langage fini et non vide.*
Alors, tout centre et toute médiane de W pour d_{L} est de longueur au plus $2 \times \max_{w \in W} |w|$.

Preuve. Soient $w_1 \in W$ et g un centre de W pour d_{L}. On a

$$|g| - |w_1| \leq d_{\mathrm{L}}(g, w_1) \leq \max_{w \in W} d_{\mathrm{L}}(g, w) \leq \max_{w \in W} d_{\mathrm{L}}(\varepsilon, w) = \max_{w \in W} |w|$$

car la première inégalité provient du point (i) du lemme 2.4 p. 41 et la dernière égalité de la remarque 2.1 p. 41. On en déduit :

$$|g| \leq |w_1| + \max_{w \in W} |w| \leq 2 \times \max_{w \in W} |w| \ .$$

Soient μ une médiane de W pour d_{L} et $w_\mu \in W$ minimisant $d_{\mathrm{L}}(\mu, w_\mu)$. On a :

$$\begin{aligned}
|\mu| - |w_\mu| &\leq d_{\mathrm{L}}(\mu, w_\mu) && \text{par le point } (i) \text{ lemme 2.4 p. 41} \\
&\leq \frac{1}{\#W} \sum_{w \in W} d_{\mathrm{L}}(\mu, w) && \text{car } w_\mu \in W \text{ minimise } d_{\mathrm{L}}(\mu, w_\mu) \\
&\leq \frac{1}{\#W} \sum_{w \in W} d_{\mathrm{L}}(\varepsilon, w) && \text{car } \mu \text{ minimise } \sum_{w \in W} d_{\mathrm{L}}(\mu, w) \\
&= \frac{1}{\#W} |W| && \text{par la remarque 2.1 p. 41 .}
\end{aligned}$$

Remarquons que $\frac{1}{\#W} |W|$ est la longueur moyenne des mots appartenant à W. Il vient :

$$|\mu| \leq |w_\mu| + \frac{1}{\#W} |W| \leq 2 \times \max_{w \in W} |w| \ .$$

Q.E.D.

Théorème 2.2
Le problème de décision associé au problème du centre pour la distance de Levenshtein (d_{L}-Center$_D$) est à la fois NP*-complet et* W[1]*-difficile pour le paramètre $\#W$. De plus, ces résultats restent vrais, même si on restreint d_{L}-Center aux instances W telles que $\sigma(W) = 2$.*

Preuve. Rappelons que d_{L} est calculable en temps polynomial. Ceci permet de construire un algorithme polynomial retournant `oui` si et seulement si son entrée est de la forme $((W, k), g)$ où
– (W, k) est une instance de d_{L}-Center$_D$ et où
– g est un mot vérifiant $d_{\mathrm{L}}(g, w) \leq k$ pour chaque $w \in W$.
Par le lemme 2.5 p. 42, la taille d'un centre g de W pour d_{L} est polynomiale en la taille de (W, k). Il en résulte que l'on a fabriqué un algorithme de validation polynomial pour d_{L}-Center$_D$, d'où d_{L}-Center$_D$ est dans NP.

On va maintenant réduire LCS^0_D (paramétré par $\#X$) à d_{L}-Center$_D$ (paramétré par $\#W$) de manière à pouvoir appliquer le théorème 2.1 p. 40.

Soit (X, k) une instance de LCS^0_D : pour tout $x \in X$, on a $|x| = 2k$. Si $\varepsilon \in X$ alors $k = 0$, $X = \{\varepsilon\}$ et la seule sous-séquence commune de X est le mot vide dont la longueur est nulle. Sans perte de généralité, on peut donc supposer que $\varepsilon \notin X$. On construit facilement en temps polynomial l'instance (W, k) de d_{L}-Center$_D$ où $W := X \cup \{\varepsilon\}$. On a alors $\sigma(W) = \sigma(X)$ mais aussi $\#W = \#X + 1$. Ainsi, le paramètre $\#W$ de l'instance réduite (W, k) n'est fonction que du paramètre $\#X$ de l'instance initiale (X, k).

Il reste à vérifier que notre transformation est bien une M-réduction. Plus précisément, nous allons démontrer l'assertion suivante : pour tout mot g, g est une sous-séquence commune de X de longueur k si et seulement si $\max_{w \in W} d_{\mathrm{L}}(g, w)$ est au plus égal à k.

• Supposons que g soit une sous-séquence commune de X de longueur k.
Le point (ii) du lemme 2.4 p. 41 garantit que, pour tout $x \in X$, on a

$$d_{\mathrm{L}}(g, x) = d_{\mathrm{L}}(x, g) = |x| - |g| = 2k - k = k \, .$$

De plus, par la remarque 2.1 p. 41, on a aussi $d_{\mathrm{L}}(g,\varepsilon) = |g| = k$. On en déduit que, pour tout $w \in W$, on a $d_{\mathrm{L}}(g,w) = k$ d'où $\max_{w\in W} d_{\mathrm{L}}(g,w) = k$, ce qu'on voulait à ce stade.

• Réciproquement, supposons que l'on ait $\max_{w\in W} d_{\mathrm{L}}(g,w) \leq k$.
Soit $x \in X$. On a :

$$\begin{array}{ll}
k \geq d_{\mathrm{L}}(g,x) & \text{car } x \in W \\
= d_{\mathrm{L}}(x,g) & \\
\geq |x| - |g| & \text{par le point } (i) \text{ du lemme 2.4 p. 41} \\
= 2k - |g| & \text{car } X \text{ est une instance de } \mathrm{LCS}^0_D \\
= 2k - d_{\mathrm{L}}(g,\varepsilon) & \text{par la remarque 2.1 p. 41} \\
\geq 2k - k & \text{car } \varepsilon \in W \\
= k\,. &
\end{array}$$

Ainsi, toutes les inégalités apparaissant ci-dessus sont en fait des égalités. On en déduit

1. que $2k - |g| = k$ d'où g est de longueur k et
2. que $d_{\mathrm{L}}(x,g) = |x| - |g|$ d'où g est une sous-séquence de x par le point (ii) du lemme 2.4 p. 41.

On a ainsi montré que g était une sous-séquence commune de X de longueur k.

Q.E.D.

2.5.2 Intraitabilité du problème de la médiane pour la distance de Levenshtein

Dans cette section, nous considérons le problème de décision d_{L}-Median$_D$ associé au problème de la médiane pour la distance de Levenshtein d_{L}. Nous montrons (théorème 2.3 p. 44) que d_{L}-Median$_D$ est NP-complet et W[1]-difficile pour le paramètre "cardinal du langage W pris en entrée", même si on se restreint à des langages W sur des alphabets à deux lettres.

Nous avons publiés ces résultats en 2003 [90]. Décrivons les meilleurs résultats connus à ce moment-là.

1. En 2000, de la Higuera et Casacuberta avaient obtenus des résultats pour d_{L}-Median$_D$ similaires aux nôtres, mais sans pouvoir imposer de borne sur $\sigma(W)$ [27]. Nous nous inspirons de leur preuve.
2. Par ailleurs, Sim et Park avaient prouvé en 1999 le résultat suivant : si Σ est de cardinal 6, alors il existe une métrique π_6 sur $\Sigma \cup \{\varepsilon\}$ telle que $d_{\mathrm{E}}^{\pi_6}$-Median$_D$ soit NP-complet [104]. Leur preuve consiste en une réduction depuis le problème de décision associé à Shortest Common Supersequence (SCS$_D$). Comme cette réduction a de bonnes propriétés, on déduit des résultats de Pietrzak concernant la complexité paramétrique de SCS$_D$ [97] que $d_{\mathrm{E}}^{\pi_6}$-Median$_D$ est W[1]-difficile pour le paramètre $\#W$.

Juste après la publication des résultats exposés dans cette section, est paru un article décisif d'Isaac Elias [32]. Celui-ci démontre que d_{E}^{π}-Median$_D$ est NP-complet pour tout alphabet Σ de cardinal 2 et toute métrique π sur $\Sigma \cup \{\varepsilon\}$. Néanmoins, sa réduction depuis Vertex Cover ne permet de tirer aucune information sur la complexité paramétrique de d_{E}^{π}-Median$_D$.

Pour pouvoir réduire LCS$_D$ à d_{L}-Median$_D$ et prouver le théorème 2.3 p. 44 annoncé ci-dessus, nous avons besoin de lier la distance de Levenshtein et la notion de sous-séquence à l'aide d'une inégalité plus fine que celle énoncée dans le point (i) du lemme 2.4 p. 41.

Lemme 2.6 *Pour tous mots x, y, on a $d_{\mathrm{L}}(x,y) \geq \max\{|x|,|y|\} - \mathrm{lcs}(\{x,y\})$.*

Preuve. Soit δ la métrique discrète sur l'ensemble des symboles de l'univers augmenté de ε. Soit A un alignement de (x,y) : on note n le nombre total de colonnes de A et l le nombre de colonnes de A qui sont des appariements exacts (c'est-à-dire le nombre de colonnes de la forme $\left[\begin{smallmatrix} a \\ a \end{smallmatrix}\right]$ où a est une lettre).

En ne considérant que les l colonnes de A qui sont des appariements exacts, on peut construire une sous-séquence commune de $\{x,y\}$ de longueur l (voir section 1.5.3). On en déduit $\mathrm{lcs}(\{x,y\}) \geq l$.

D'autre part, chaque lettre de x (resp. de y) apparaît dans l'une des n colonnes de A. On en déduit $n \geq |x|$ (resp. $n \geq |y|$).

On a ainsi montré $n-l \geq \max\{|x|, |y|\} - \text{lcs}(\{x,y\})$. Or, $n-l$ est exactement le δ-score de A donc, quitte à prendre A de δ-score minimum $d_{\text{L}}(x,y)$, on obtient l'inégalité recherchée. **Q.E.D.**

L'inégalité donnée dans le lemme 2.6 ci-dessus ne fait intervenir que 2 mots. Pour la généraliser à un nombre quelconque de mots (lemme 2.8 p. 44) nous avons besoin du lemme suivant (lemme 2.7 p. 44).

Expliquons sa signification intuitive. Pour tout ensemble fini M et tous sous-ensembles S_1, $S_2 \subseteq M$, on a $S_1 \cup S_2 \subseteq M$ d'où $\#M \geq \#(S_1 \cup S_2) = \#S_1 + \#S_2 - \#(S_1 \cap S_2)$ et par suite S_1 et S_2 ont une intersection de cardinal au moins $\#S_1 + \#S_2 - \#M$. Le lemme 2.7 p. 44 est l'analogue de cette minoration en terme de sous-séquences.

Lemme 2.7 *Soient un mot μ et deux sous-séquences s_1 et s_2 de μ.*
Alors, $\{s_1, s_2\}$ admet une sous-séquence commune de longueur au moins $|s_1| + |s_2| - |\mu|$.

Preuve. Soit $M := [1, |\mu|]$. Pour chaque $i \in \{1,2\}$, soit une partie $S_i \subseteq M$ telle que $s_i = \text{sseq}(\mu, S_i)$. Alors, $\text{sseq}(\mu, S_1 \cap S_2)$ est une sous-séquence commune de $\{s_1, s_2\}$ de longueur $\#(S_1 \cap S_2) \geq \#S_1 + \#S_2 - \#M = |s_1| + |s_2| - |\mu|$. **Q.E.D.**

Lemme 2.8 *Pour tout mot μ et tout langage fini Y, on a :*

$$\sum_{y \in Y} d_{\text{L}}(\mu, y) + (\#Y - 1)\,|\mu| \geq |Y| - \text{lcs}(\{\mu\} \cup Y)\,.$$

Preuve. On procède par récurrence sur $\#Y$. Si Y est de cardinal 0 alors les deux membres de l'inégalité que l'on cherche à montrer sont égaux à $-|\mu|$. Supposons maintenant $\#Y \geq 1$.

Soient $y_0 \in Y$ et $Y' := Y \setminus \{y_0\}$. En appliquant le lemme 2.6 p. 43 avec $x := \mu$ et $y := y_0$, on obtient :

$$d_{\text{L}}(\mu, y_0) \geq |y_0| - \text{lcs}(\{\mu, y_0\})\,. \tag{2.14}$$

Par ailleurs, l'hypothèse de récurrence appliquée à Y' donne :

$$\sum_{y' \in Y'} d_{\text{L}}(\mu, y') + (\#Y' - 1)\,|\mu| \geq |Y'| - \text{lcs}(\{\mu\} \cup Y')\,. \tag{2.15}$$

On ajoute maintenant (2.14), (2.15) et l'inégalité triviale $|\mu| \geq |\mu|$. Il en résulte :

$$\sum_{y \in Y} d_{\text{L}}(\mu, y) + (\#Y')\,|\mu| \geq |Y'| - \text{lcs}(\{\mu\} \cup Y') + |y_0| - \text{lcs}(\{\mu, y_0\}) + |\mu|\ .$$

Comme $\#Y' = \#Y - 1$ et comme $|Y'| + |y_0| = |Y|$, on a en fait :

$$\sum_{y \in Y} d_{\text{L}}(\mu, y) + (\#Y - 1)\,|\mu| \geq |Y| - (\text{lcs}(\{\mu\} \cup Y') + \text{lcs}(\{\mu, y_0\}) - |\mu|)\ . \tag{2.16}$$

Considérons alors une sous-séquence commune s_1 de $\{\mu\} \cup Y'$ de longueur maximale $\text{lcs}(\{\mu\} \cup Y')$ et une sous-séquence commune s_2 de $\{\mu, y_0\}$ de longueur maximale $\text{lcs}(\{\mu, y_0\})$. Par le lemme 2.7 p. 44, $\{s_1, s_2\}$ admet une sous-séquence commune s de longueur au moins $\text{lcs}(\{\mu\} \cup Y') + \text{lcs}(\{\mu, y_0\}) - |\mu|$. Or, s est aussi une sous-séquence commune de $(\{\mu\} \cup Y') \cup \{\mu, y_0\} = \{\mu\} \cup Y$ d'où :

$$\text{lcs}(\{\mu\} \cup Y') + \text{lcs}(\{\mu, y_0\}) - |\mu| \leq |s| \leq \text{lcs}(\{\mu\} \cup Y)\,.$$

En combinant cette dernière inégalité avec (2.16), on obtient le résultat recherché. **Q.E.D.**

Notons que lorsque Y est réduit à un singleton, le lemme 2.8 p. 44 se déduit du lemme 2.6 p. 43 (et réciproquement). Nous avons maintenant tous les outils pour démontrer le théorème 2.3 p. 44.

Théorème 2.3
Le problème de décision associé au problème de la médiane pour la distance de Levenshtein (d_{L}-MEDIAN$_D$) est à la fois NP*-complet et* W[1]*-difficile pour le paramètre $\#W$. De plus, ces résultats restent vrais même si on restreint d_{L}-MEDIAN aux instances W telles que $\sigma(W) = 2$.*

Preuve. En s'appuyant sur le lemme 2.5 p. 42 et sur le fait que d_{L} est calculable en temps polynomial, on montre facilement (voir la preuve du théorème 2.2 p. 42) que toute instance (W, K) de d_{L}-MEDIAN$_D$ admet comme certificats polynomiaux les médianes de W pour d_{L}. Ainsi, d_{L}-MEDIAN$_D$ est dans NP.

Il reste à réduire LCS$_D$ (paramétré par $\#X$) à d_{L}-MEDIAN$_D$ (paramétré par $\#W$). Soit (X, k) une instance de LCS$_D$ avec $X \subseteq \{\mathtt{0}, \mathtt{1}\}^\star$. Posant $m := \#X$ et

$$N := \max\left\{|X| + \tfrac{1}{2}m(m-1) - k, m-1\right\}$$

on construit une instance (W, K) de d_{L}-MEDIAN$_D$ avec :

$$\begin{aligned} W &:= X\mathtt{0}^N \cup \left\{\mathtt{0}^i : i \in [1, m-1]\right\}\,, \\ K &:= |X| + (m-1)N - k - \tfrac{1}{2}m(m-1)\,. \end{aligned}$$

Cette construction s'effectue trivialement en temps polynomial. De plus,

$$\#W = \#(X\mathtt{0}^N) + (m-1) = 2 \times \#X - 1$$

donc le paramètre $\#W$ de l'instance réduite (W, K) n'est fonction que du paramètre $\#X$ de l'instance initiale (X, k). Il reste à vérifier que notre transformation est bien une M-réduction c'est-à-dire que : X admet une sous-séquence commune de longueur (au moins) k si et seulement s'il existe $\mu \in \{\mathtt{0}, \mathtt{1}\}^\star$ vérifiant $\sum_{w\in W} d_{\mathrm{L}}(\mu, w) \leq K$.

• Supposons que X admette une sous-séquence commune s de longueur k. On pose $\mu := s\mathtt{0}^N$ et on va montrer que l'on a $\sum_{w\in W} d_{\mathrm{L}}(\mu, w) \leq K$.

Pour chaque $i \in [1, m-1]$, on a $i \leq m-1 \leq N$, d'où $\mathtt{0}^i$ est une sous-séquence de μ et par suite, le point (ii) du lemme 2.4 p. 41 garantit :

$$d_{\mathrm{L}}(\mu, \mathtt{0}^i) = |\mu| - \left|\mathtt{0}^i\right| = k + N - i\,. \tag{2.17}$$

Par ailleurs, $\mu = s\mathtt{0}^N$ est une sous-séquence de $x\mathtt{0}^N$ pour tout $x \in X$, donc le lemme 2.4 p. 41 donne également :

$$d_{\mathrm{L}}(\mu, x\mathtt{0}^N) = \left|x\mathtt{0}^N\right| - |\mu| = \left|x\mathtt{0}^N\right| - \left|s\mathtt{0}^N\right| = |x| - |s| = |x| - k\,. \tag{2.18}$$

À l'aide de (2.17) et (2.18), on peut maintenant calculer $\sum_{w\in W} d_{\mathrm{L}}(\mu, w)$:

$$\begin{aligned} \sum_{w\in W} d_{\mathrm{L}}(\mu, w) &= \sum_{x\in X} d_{\mathrm{L}}(\mu, x\mathtt{0}^N) + \sum_{i=1}^{m-1} d_{\mathrm{L}}(\mu, \mathtt{0}^i) \\ &= \sum_{x\in X} (|x| - k) + \sum_{i=1}^{m-1} (k + N - i) \\ &= (|X| - mk) + \big((m-1)k + (m-1)N - \tfrac{1}{2}m(m-1)\big) \\ &= K\,, \end{aligned}$$

ce qu'on voulait à ce stade.

• Réciproquement, supposons qu'il existe $\mu \in \{\mathtt{0}, \mathtt{1}\}^\star$ vérifiant $\sum_{w\in W} d_{\mathrm{L}}(\mu, w) \leq K$ et montrons que l'on a $\mathrm{lcs}(X) \geq k$.

On commence par montrer que μ contient au moins $m-1$ occurrences de la lettre $\mathtt{0}$. Intuitivement, ceci est dû au fait que l'on a ajouté un nombre convenable, N, de $\mathtt{0}$ à la fin de chaque mot de X. La preuve formelle s'appuie sur la remarque suivante :

Remarque 2.2 *Pour tous mots u, v et toute lettre a, on a $d_{\mathrm{L}}(u, v) \geq |v|_a - |u|_a$.*

La remarque 2.2 p. 45 se justifie par le fait suivant : modifier un mot w quelconque à l'aide d'une OÉÉ fait varier $|w|_a$ d'au plus 1.

Lemme 2.9 $|\mu|_{\mathtt{0}} \geq m-1$.

Preuve. Pour tout $x' \in X\mathtt{0}^N$, on a $|x'|_{\mathtt{0}} \geq N$, d'où

$$N - |\mu|_{\mathtt{0}} \leq |x'|_{\mathtt{0}} - |\mu|_{\mathtt{0}} \leq d_{\mathrm{L}}(\mu, x')\,,$$

la dernière inégalité provenant de la remarque 2.2 p. 45. En sommant sur tous les x' de $X\mathtt{0}^N$, on obtient :

$$mN - m\,|\mu|_{\mathtt{0}} \leq \sum_{x' \in X\mathtt{0}^N} d_{\mathrm{L}}(\mu, x') \leq K$$

d'où :

$$\begin{aligned} m\,|\mu|_{\mathtt{0}} &\geq mN - K \\ &= mN - \big(|X| + (m-1)N - k - \tfrac{1}{2}m(m-1)\big) \\ &= N - |X| + k + \tfrac{1}{2}m(m-1) \\ &\geq \big(|X| + \tfrac{1}{2}m(m-1) - k\big) - |X| + k + \tfrac{1}{2}m(m-1) \\ &= m(m-1)\,. \end{aligned}$$

On en déduit $|\mu|_{\mathtt{0}} \geq m-1$, ce qui finit la preuve du lemme 2.9. **Q.E.D.**

Le lemme 2.9 p. 46 garantit que, pour tout $i \in [1, m-1]$, $\mathtt{0}^i$ est une sous-séquence de μ et donc, $d_{\mathrm{L}}(\mu, \mathtt{0}^i) = |\mu| - i$ par le point (ii) du lemme 2.4 p. 41. Ainsi, on a :

$$\sum_{i=1}^{m-1} d_{\mathrm{L}}(\mu, \mathtt{0}^i) = \sum_{i=1}^{m-1} |\mu| - \sum_{i=1}^{m-1} i = (m-1)\mu - \tfrac{1}{2}m(m-1)\,. \tag{2.19}$$

Par ailleurs, on a aussi :

$$\big|X\mathtt{0}^N\big| = \sum_{x \in X} \big|x\mathtt{0}^N\big| = \sum_{x \in X} (|x| + N) = |X| + mN \tag{2.20}$$

et

$$\mathrm{lcs}(\{\mu\} \cup X\mathtt{0}^N) \leq \mathrm{lcs}(X\mathtt{0}^N) = \mathrm{lcs}(X) + N\,, \tag{2.21}$$

la dernière égalité provenant du lemme 2.1 p. 40.

Il ne reste plus qu'à combiner les résultats précédents. Il vient :

$$\begin{aligned} K &\geq \sum_{w \in W} d_{\mathrm{L}}(\mu, w) \\ &= \sum_{x' \in X\mathtt{0}^N} d_{\mathrm{L}}(\mu, x') + \sum_{i=1}^{m-1} d_{\mathrm{L}}(\mu, \mathtt{0}^i) \\ &= \sum_{x' \in X\mathtt{0}^N} d_{\mathrm{L}}(\mu, x') + (m-1)\,|\mu| - \tfrac{1}{2}m(m-1) && \text{(2.22a)} \\ &\geq \big|X\mathtt{0}^N\big| - \mathrm{lcs}(\{\mu\} \cup X\mathtt{0}^N) - \tfrac{1}{2}m(m-1) && \text{(2.22b)} \\ &= |X| + mN - \mathrm{lcs}(\{\mu\} \cup X\mathtt{0}^N) - \tfrac{1}{2}m(m-1) && \text{(2.22c)} \\ &\geq |X| + (m-1)N - \mathrm{lcs}(X) - \tfrac{1}{2}m(m-1)\,. && \text{(2.22d)} \end{aligned}$$

En effet, les égalités (2.22a) et (2.22c) se déduisent respectivement de (2.19) et (2.20). De plus, l'inégalité (2.22b) s'obtient en appliquant le lemme 2.8 p. 44 avec $Y := X\mathtt{0}^N$, et l'inégalité (2.22d) est une conséquence immédiate de (2.21).

Les calculs ci-avant nous ont permis d'obtenir :

$$K \geq |X| + (m-1)N - \mathrm{lcs}(X) - \tfrac{1}{2}m(m-1)$$

d'où la minoration annoncée :

$$\mathrm{lcs}(X) \geq |X| + (m-1)N - \tfrac{1}{2}m(m-1) - K = k\,.$$

Ceci conclut la preuve du théorème 2.3. **Q.E.D.**

2.6 Intraitabilité du problème du centre pour une large classe de distances d'édition

Dans la section précédente, nous avons montré la difficulté du problème du centre pour une seule distance, celle de Levenshtein (théorème 2.2 p. 42). Dans cette section, nous généralisons ce résultat à une large classe de distances d'édition. On suppose que la fonction de pondération π est une métrique sur $\Sigma \cup \{\varepsilon\}$ satisfaisant la propriété naturelle $(*)$ p. 39. Sous ces hypothèses, nous montrons (théorème 2.5 p. 51) que le problème du centre pour la distance d'édition d_{E}^{π} est

– NP-difficile, et

– W[1]-difficile pour le paramètre "nombre de mots pris en entrée" (c'est-à-dire le cardinal de W).

De plus, notre preuve garantit que ces résultats restent vrais même si Σ est de cardinal 2. Elle est analogue à celle du théorème 2.2 p. 42 sauf qu'elle s'appuie sur une réduction depuis une version *pondérée* de LCS_D^0.

Dans la section 2.6.1, nous commençons par introduire formellement cette variante de LCS_D^0 (définition 2.7 p. 47). Nous démontrons ensuite son intraitabilité dans le théorème 2.4 p. 49. Enfin, dans la section 2.6.2, nous déduisons de ce théorème notre résultat principal concernant d_{E}^{π}-Center.

2.6.1 Le problème de la sous-séquence commune pondérée

On définit tout d'abord la notion de longueur pondérée :

Définition 2.6 (Longueur pondérée)
On appelle longueur pondérée *sur $\Sigma^\star$ toute application $\ell : \Sigma^\star \to \mathbb{N}$ vérifiant :*

– *pour tout $a \in \Sigma$, $\ell(a) \geq 1$ et*

– *pour tous $x, y \in \Sigma^\star$, $\ell(xy) = \ell(x) + \ell(y)$.*

Toute longueur pondérée ℓ est un morphisme de monoïdes libres (on pourra se reporter à [77] pour plus de détails sur les monoïdes libres). Ainsi, on a $\ell(\varepsilon) = 0$ et ℓ est défini par sa restriction à Σ :

$$\forall w \in \Sigma^\star \qquad \ell(w) = \sum_{a \in \Sigma} |w|_a \, \ell(a) \, .$$

La longueur pondérée sur $\Sigma^\star$ qui à toute lettre $a \in \Sigma$ associe 1 est la longueur usuelle $|\cdot|$ sur $\Sigma^\star$.

On peut maintenant introduire le problème de la sous-séquence commune pondérée et sa restriction correspondant à LCS_D^0 :

Définition 2.7
Soit ℓ une longueur pondérée sur $\Sigma^\star$.

*On nomme ℓ-*Weighted Common Subsequence *(ℓ-WCS en abrégé) le problème de décision :* "Étant donné un langage fini et non vide $X \subseteq \Sigma^\star$ et un entier $k \geq 0$, décider si X admet une sous-séquence commune s telle que $\ell(s) = k$".

On note ℓ-WCS^0 la restriction de ℓ-WCS aux instances (X, k) telles que, pour tout $x \in X$, on ait $\ell(x) = 2k$.

Remarque 2.3 *Supposons que $\ell(w) = |w|$ pour tout mot w. Alors ℓ-WCS (resp. ℓ-WCS^0) n'est autre que le problème* LCS_D *(resp.* LCS_D^0*).*

Le but de cette section est d'établir l'intraitabilité de ℓ-WCS^0 pour toute longueur pondérée ℓ (théorème 2.4 p. 49). Pour ce faire, nous introduisons une classe très particulière de morphismes de monoïdes libres, les morphismes amplifiants :

Définition 2.8 (Morphisme amplifiant)
On dit qu'une application $f : \Sigma^\star \to \Sigma^\star$ est un morphisme amplifiant *sur $\Sigma^\star$ lorsqu'elle vérifie les deux propriétés suivantes :*

– *pour tous $x, y \in \Sigma^\star$, $f(xy) = f(x)f(y)$ (autrement dit, f est un morphisme) et*

– *pour tout $a \in \Sigma$, $f(a)$ est une puissance de a non réduite au mot vide.*

Notons que, comme précédemment, un morphisme amplifiant sur $\Sigma^\star$ est entièrement défini par sa restriction à Σ et transforme le mot vide en lui-même.

De plus, tout morphisme amplifiant est injectif. En effet, il est évident que la restriction à Σ d'un morphisme amplifiant f sur $\Sigma^\star$ est injective et que $f(\Sigma)$ est un *code* [77, Chapitre 6]. Un code est un langage C tel que tout mot s'écrit d'au plus une façon comme concaténation de mots appartenant à C.

Exemple 2.10 *Considérons le morphisme amplifiant f sur $\{\mathtt{a},\mathtt{b},\mathtt{c}\}^\star$ donné par : $f(\mathtt{a}) = \mathtt{aaaa}$, $f(\mathtt{b}) = \mathtt{b}$ et $f(\mathtt{c}) = \mathtt{cc}$. On a alors :*

$$f(\mathtt{aaccabcb}) = \mathtt{aaaaaaaaccccaaaabccb}\,.$$

Le lemme suivant énonce la principale propriété des morphismes amplifiants que nous utilisons dans la preuve du théorème 2.4 p. 49.

Lemme 2.10 *Soient un morphisme amplifiant f sur $\Sigma^\star$ et un langage fini, non vide $X \subseteq \Sigma^\star$.*

Alors, pour chaque sous-séquence commune t de $f(X)$, il existe une sous-séquence commune s de X telle que t soit une sous-séquence de $f(s)$.

Preuve. Supposons, en préambule, que $\varepsilon \in X$. Alors, on a $\varepsilon = f(\varepsilon) \in f(X)$ donc, $t := \varepsilon$ est la seule sous-séquence commune de $f(X)$. Par suite, il suffit de prendre $s := \varepsilon$ et le cas où $\varepsilon \in X$ est traité.

Terminons la preuve de notre lemme par récurrence sur $|X|$.

Si un langage non vide X vérifie $|X| = 0$ alors $X = \{\varepsilon\}$. Comme ce cas vient d'être traité, notre récurrence s'initialise bien. Supposons maintenant $|X| \geq 1$. Comme on peut aussi supposer $\varepsilon \notin X$, il suffit de distinguer deux cas complémentaires.

• *Dans un premier temps, supposons que tous les mots de X se terminent par une même lettre $a \in \Sigma$.*

Alors, il existe $X' \subseteq \Sigma^\star$ tel que $X = X'a$: on a $|X'| < |X|$ et $f(X) = f(X')f(a)$. Notons α le plus long suffixe de t qui est sous-séquence de $f(a)$ et écrivons t sous la forme $t = t'\alpha$ avec $t' \in \Sigma^\star$. Par construction, t' est une sous-séquence commune de $f(X')$ donc l'hypothèse de récurrence garantit qu'il existe une sous-séquence commune s' de X telle que t' soit une sous-séquence de $f(s')$. Posant $s := s'a$, on obtient une sous-séquence commune de X telle que t soit une sous-séquence de $f(s)$: ce qu'on voulait à ce stade.

• *Dans un second temps, supposons que X contienne deux mots se terminant par des lettres distinctes.*

Alors, il existe a, $b \in \Sigma$ et u, $v \in \Sigma^\star$ avec $a \neq b$ et $\{ua, vb\} \subseteq X$. Quitte à échanger a et b, on peut supposer que a n'est pas un suffixe de t. Comme t est une sous-séquence de $f(ua) = f(u)f(a) = f(u)a^{|f(a)|}$, t est en fait une sous-séquence de $f(u)$. Par suite, posant $X' := (X \setminus \{ua\}) \cup \{u\}$, t est une sous-séquence commune de $f(X')$ et on a $|X'| < |X|$ (plus précisément, $|X'| = |X| - 1$ si $u \notin X$ et $|X'| = |X| - |ua|$ sinon). L'hypothèse d'induction garantit donc qu'il existe une sous-séquence commune s de X' telle que t soit une sous-séquence de $f(s)$. Or, par construction de X', toute sous-séquence commune de X' est aussi une sous-séquence commune de X. On en déduit que s est une sous-séquence commune de X.

Q.E.D.

Notons que, dans l'énoncé du lemme ci-dessus, nous supposons que le langage X est *fini*. En fait, même si on lève cette hypothèse, le lemme de compacité suivant garantit que la conclusion du lemme 2.10 p. 48 reste encore valide. De toute façon, supposer X fini n'empêche pas l'application du lemme 2.10 p. 48 dans la preuve du théorème 2.4 p. 49.

Lemme 2.11 (Compacité) *Soit X un langage non vide.*

Alors, il existe un langage fini et non vide $X' \subseteq X$, tel que X et X' admettent les mêmes sous-séquences communes.

Preuve. Fixons un mot $u \in X$. Comme un mot donné n'admet qu'un nombre fini de sous-séquences, l'ensemble S des sous-séquences de u qui ne sont pas des sous-séquences communes de X est fini. Pour chaque $s \in S$, on peut trouver un mot $x_s \in X$ n'admettant pas s comme sous-séquence. Posant $X' := \{u\} \cup \{x_s : s \in S\}$, on obtient une partie finie et non vide de X.

Trivialement toute sous-séquence commune de X est une sous-séquence commune de X'.

Réciproquement, supposons (absurde) qu'il existe une sous-séquence commune s de X' qui ne soit pas une sous-séquence commune de X. Comme $u \in X'$, s est une sous-séquence de u donc $s \in S$. Or, s n'est pas sous-séquence du mot x_s qui appartient à X' : contradiction. **Q.E.D.**

Nous sommes maintenant en mesure de montrer le théorème suivant :

Théorème 2.4
Soit ℓ une longueur pondérée sur $\Sigma^\star$.

Alors, ℓ-WCS^0 est à la fois NP*-difficile et* W[1]*-difficile pour le paramètre $\#X$. De plus, ces résultats restent vrais même si Σ est de cardinal* 2.

Preuve. Dans cette preuve, nous allons utiliser le morphisme amplifiant f sur $\{\mathtt{0},\mathtt{1}\}^\star$ donné par : $f(\mathtt{0}) = \mathtt{0}^{\ell(\mathtt{1})}$ et $f(\mathtt{1}) = \mathtt{1}^{\ell(\mathtt{0})}$. Ce morphisme remplace chaque $\mathtt{0}$ (resp. $\mathtt{1}$) par un nombre de $\mathtt{0}$ (resp. de $\mathtt{1}$) égal à la longueur pondérée de $\mathtt{1}$ (resp. de $\mathtt{0}$). Ainsi, on a

$$\ell(f(\mathtt{0})) = \ell(\mathtt{0})\ell(\mathtt{1}) = \ell(f(\mathtt{1}))\,.$$

Donc, pour tout $x \in \{\mathtt{0},\mathtt{1}\}^\star$, la longueur pondérée de $f(x)$ ne dépend que de la longueur (usuelle) de $f(x)$. Plus précisément, on a

$$\forall x \in \{\mathtt{0},\mathtt{1}\}^\star \qquad \ell(f(x)) = \ell(\mathtt{0})\ell(\mathtt{1})\,|x|\ . \tag{2.23}$$

Cette propriété de f va nous permettre de réduire LCS^0_D (paramétré par $\#X$) à ℓ-WCS^0 (paramétré par $\#X$) de manière à pouvoir appliquer le théorème 2.1 p. 40.

Considérons une instance (X,k) de LCS^0_D avec $X \subseteq \{\mathtt{0},\mathtt{1}\}^{2k}$. On construit une instance $(\hat{X},\hat{k})$ de ℓ-WCS^0 avec $\hat{X} := f(X)$ et $\hat{k} := \ell(\mathtt{0})\ell(\mathtt{1})k$. Pour tout $x \in X$, (2.23) garantit que $\ell(f(x)) = \ell(\mathtt{0})\ell(\mathtt{1})(2k) = 2\hat{k}$. Par suite, $(\hat{X},\hat{k})$ est bien une instance de ℓ-WCS^0.

D'autre part, il est évident que l'on peut construire $(\hat{X},\hat{k})$ en temps polynomial à partir de (X,k). De plus, comme f est injectif, X et $\hat{X}$ sont de même cardinal : le paramètre $\#\hat{X}$ de l'instance réduite $(\hat{X},\hat{k})$ n'est fonction que du paramètre $\#X$ de l'instance initiale (X,k). Il ne reste plus qu'à vérifier que notre transformation est bien une M-réduction c'est-à-dire que : X admet une sous-séquence commune de longueur k si et seulement si $\hat{X}$ admet une sous-séquence commune $\hat{s}$ telle que $\ell(\hat{s}) = \hat{k}$.

• Supposons que X admette une sous-séquence commune s de longueur k.

Alors $\hat{s} := f(s)$ est une sous-séquence commune de $\hat{X}$ telle que $\ell(\hat{s}) = \hat{k}$ (appliquer l'équation (2.23) avec $x := s$).

• Réciproquement, supposons que $\hat{X}$ admette une sous-séquence commune $\hat{s}$ telle que $\ell(\hat{s}) = \hat{k}$.

Par le lemme 2.10 p. 48, il existe une sous-séquence commune s de X telle que $\hat{s}$ soit une sous-séquence de $f(s)$. Comme $\hat{s}$ est une sous-séquence de $f(s)$, on a $\ell(\hat{s}) \leq \ell(f(s))$ d'où :

$$\ell(\mathtt{0})\ell(\mathtt{1})k = \hat{k} = \ell(\hat{s}) \leq \ell(f(s)) = \ell(\mathtt{0})\ell(\mathtt{1})\,|s|\ .$$

On en déduit $|s| \geq k$ donc, quitte à effacer $|s| - k$ lettres quelconques de s, on obtient une sous-séquence commune de X de longueur k.

Q.E.D.

2.6.2 Le problème du centre pour la distance d'édition

La propriété $(*)$ p. 39 permet de généraliser le lemme 2.4 p. 41 aux distances d'édition pondérées par des métriques :

Lemme 2.12 *On suppose que π est une métrique sur $\Sigma \cup \{\varepsilon\}$.*

Alors, les 3 *assertions suivantes sont vraies quels que soient x, $y \in \Sigma^\star$:*

(i). $d^\pi_{\mathrm{E}}(y,x) \geq d^\pi_{\mathrm{E}}(y,\varepsilon) - d^\pi_{\mathrm{E}}(x,\varepsilon)$;

(ii). si x est une sous-séquence de y alors $d^\pi_{\mathrm{E}}(y,x) = d^\pi_{\mathrm{E}}(y,\varepsilon) - d^\pi_{\mathrm{E}}(x,\varepsilon)$;

(iii). si π satisfait la propriété $(*)$ et si $d_{\mathrm{E}}^{\pi}(y,x) = d_{\mathrm{E}}^{\pi}(y,\varepsilon) - d_{\mathrm{E}}^{\pi}(x,\varepsilon)$ alors, x est une sous-séquence de y.

Preuve.

(i). Les hypothèses faites sur π garantissent que d_{E}^{π} satisfait l'inégalité triangulaire donc, en particulier, on a $d_{\mathrm{E}}^{\pi}(y,\varepsilon) \leq d_{\mathrm{E}}^{\pi}(y,x) + d_{\mathrm{E}}^{\pi}(x,\varepsilon)$. On en déduit l'assertion (i).

(ii). Supposons que x soit une sous-séquence de y.

Soient $q := |y|$ et $I \subseteq [1,q]$ tel que $x = \mathrm{sseq}(y,I)$. Posons $\alpha_i := y[i]$ pour chaque $i \in I$, $\alpha_i := \varepsilon$ pour chaque $i \in [1,q] \setminus I$ et

$$A := \begin{bmatrix} y[1] & y[2] & \cdots & y[q] \\ \alpha_1 & \alpha_2 & \cdots & \alpha_q \end{bmatrix}.$$

La matrice A est un alignement de (y,x). Sur la ligne supérieure de A, il n'apparaît pas de $-$ (seulement les lettres de y dans l'ordre naturel gauche-droite). De plus, pour chaque $i \in [1,q]$, la i-ème lettre $y[i]$ de y est représentée dans A au dessus de α_i qui est :

- le mot vide si, pour obtenir x à partir de y, on efface $y[i]$,
- $y[i]$ sinon.

Comme A est un alignement de (y,x) , on peut majorer $d_{\mathrm{E}}^{\pi}(y,x)$ par le π-score de A, d'où :

$$d_{\mathrm{E}}^{\pi}(y,x) \leq \sum_{i=1}^{q} \pi(y[i],\alpha_i)\,. \tag{2.24}$$

Or, on a :

$$\sum_{i=1}^{q} \pi(y[i],\alpha_i) = \sum_{i\in[1,q]\setminus I} \pi(y[i],\varepsilon) + \sum_{i\in I} \pi(y[i],y[i]) = \sum_{i\in[1,q]\setminus I} \pi(y[i],\varepsilon)$$

car $\pi(\alpha,\alpha) = 0$ pour tout $\alpha \in \Sigma$. Ceci permet d'écrire :

$$\begin{aligned}\sum_{i=1}^{q} \pi(y[i],\alpha_i) &= \left(\sum_{i\in[1,q]\setminus I} \pi(y[i],\varepsilon) + \sum_{i\in I} \pi(y[i],\varepsilon) \right) - \sum_{i\in I} \pi(y[i],\varepsilon) \\ &= \sum_{i=1}^{q} \pi(y[i],\varepsilon) - \sum_{i\in I} \pi(y[i],\varepsilon)\end{aligned}$$

et, comme la remarque 1.3 p. 22 garantit

$$d_{\mathrm{E}}^{\pi}(y,\varepsilon) = \sum_{i=1}^{q} \pi(y[i],\varepsilon) \qquad \text{et} \qquad d_{\mathrm{E}}^{\pi}(x,\varepsilon) = \sum_{i\in I} \pi(y[i],\varepsilon)\,,$$

il vient :

$$\sum_{i=1}^{q} \pi(y[i],\alpha_i) = d_{\mathrm{E}}^{\pi}(y,\varepsilon) - d_{\mathrm{E}}^{\pi}(x,\varepsilon)\,.$$

En combinant (2.24) avec cette dernière égalité, on obtient $d_{\mathrm{E}}^{\pi}(y,x) \leq d_{\mathrm{E}}^{\pi}(y,\varepsilon) - d_{\mathrm{E}}^{\pi}(x,\varepsilon)$. Il suffit maintenant d'utiliser le point (i) pour obtenir le point (ii).

(iii). Supposons que π vérifie la propriété $(*)$ et que $d_{\mathrm{E}}^{\pi}(y,x) = d_{\mathrm{E}}^{\pi}(y,\varepsilon) - d_{\mathrm{E}}^{\pi}(x,\varepsilon)$.

Soit

$$\begin{bmatrix} \beta_1 & \beta_2 & \cdots & \beta_n \\ \alpha_1 & \alpha_2 & \cdots & \alpha_n \end{bmatrix}$$

un alignement de (y,x) de π-score optimal $\sum_{i=1}^{n} \pi(\beta_i,\alpha_i) = d_{\mathrm{E}}^{\pi}(y,x)$. Pour montrer que x est une sous-séquence de y, il suffit de vérifier que :

$$\forall i \in [1,n] \qquad (\alpha_i = \beta_i) \text{ ou } (\alpha_i = \varepsilon)\,. \tag{2.25}$$

Comme $y = \beta_1\beta_2\dots\beta_n$, la remarque 1.3 p. 22 permet d'écrire : $d_{\mathrm{E}}^{\pi}(y,\varepsilon) = \sum_{i=1}^{n} \pi(\beta_i,\varepsilon)$ car $\pi(\beta_i,\varepsilon) = 0$ dès que $\beta_i = \varepsilon$. De même, on a également $d_{\mathrm{E}}^{\pi}(x,\varepsilon) = \sum_{i=1}^{n} \pi(\alpha_i,\varepsilon)$. Il vient alors :

$$\sum_{i=1}^{n} \pi(\beta_i,\alpha_i) = d_{\mathrm{E}}^{\pi}(y,x) = d_{\mathrm{E}}^{\pi}(y,\varepsilon) - d_{\mathrm{E}}^{\pi}(x,\varepsilon) = \sum_{i=1}^{n} \pi(\beta_i,\varepsilon) - \sum_{i=1}^{n} \pi(\alpha_i,\varepsilon)$$

d'où

$$\sum_{i=1}^{n} \pi(\beta_i,\varepsilon) = \sum_{i=1}^{n} \big(\pi(\beta_i,\alpha_i) + \pi(\alpha_i,\varepsilon)\big)\,. \tag{2.26}$$

Or, pour chaque $i \in [1,n]$, on a l'inégalité triangulaire $\pi(\beta_i,\varepsilon) \leq \pi(\beta_i,\alpha_i) + \pi(\alpha_i,\varepsilon)$. Par suite, les deux sommes sur i intervenant dans (2.26) sont égales terme à terme. Ainsi, pour tout $i \in [1,n]$, on a

$$\pi(\beta_i,\varepsilon) = \pi(\beta_i,\alpha_i) + \pi(\alpha_i,\varepsilon)\,. \tag{2.27}$$

Si on avait (absurde) $\beta_i = \varepsilon$ alors (2.27) donnerait

$$\begin{aligned} 0 = \pi(\varepsilon,\varepsilon) = \pi(\beta_i,\varepsilon) &= \pi(\beta_i,\alpha_i) + \pi(\alpha_i,\varepsilon) = \pi(\varepsilon,\alpha_i) + \pi(\alpha_i,\varepsilon) \\ &= 2 \times \pi(\varepsilon,\alpha_i) \end{aligned}$$

d'où $\alpha_i = \varepsilon = \beta_i$. Comme $\left[\begin{smallmatrix}\alpha_i\\ \beta_i\end{smallmatrix}\right]$ est une colonne d'alignement, on débouche sur une contradiction. Il en résulte $\beta_i \neq \varepsilon$ ou encore $\beta_i \in \Sigma$.

Supposons maintenant (absurde) que l'on ait $\beta_i \neq \alpha_i$ et $\alpha_i \neq \varepsilon$. Alors, α_i et β_i sont deux lettres distinctes. La propriété $(*)$ p. 39 garantit donc $\pi(\beta_i,\varepsilon) < \pi(\beta_i,\alpha_i) + \pi(\alpha_i,\varepsilon)$, ce qui contredit (2.27). On a ainsi démontré (2.25), d'où x est une sous-séquence de y.

Q.E.D.

Pour démontrer l'intraitabilité de d_{E}^{π}-Center$_D$, nous exhibons une réduction depuis le problème de la sous-séquence commune pondérée par la fonction de longueur introduite ci-dessous (voir preuve du théorème 2.5 p. 51).

Lemme 2.13 *Supposons que la fonction de pondération $\pi : (\Sigma \cup \{\varepsilon\}) \times (\Sigma \cup \{\varepsilon\}) \to \mathbb{Z}$ soit positive et séparante.*

Alors, l'application $d_{\mathrm{E}}^{\pi}(\,\cdot\,,\varepsilon)$ qui à tout $w \in \Sigma^\star$, associe $d_{\mathrm{E}}^{\pi}(w,\varepsilon)$ est une longueur pondérée sur $\Sigma^\star$.

Preuve. La remarque 1.3 p. 22 garantit que, pour tout $a \in \Sigma$, on a $d_{\mathrm{E}}^{\pi}(a,\varepsilon) = \pi(a,\varepsilon)$. Comme π est positive, à valeurs entières et séparante, cette dernière quantité est au moins égale à 1.

On tire de plus de la remarque 1.3 p. 22 que, pour tous x, $y \in \Sigma^\star$, on a

$$d_{\mathrm{E}}^{\pi}(xy,\varepsilon) = \sum_{i=1}^{|x|} \pi(x[i],\varepsilon) + \sum_{j=1}^{|y|} \pi(y[j],\varepsilon) = d_{\mathrm{E}}^{\pi}(x,\varepsilon) + d_{\mathrm{E}}^{\pi}(y,\varepsilon)\,.$$

Q.E.D.

Nous sommes maintenant en mesure de montrer le résultat principal de la section 2.6 :

Théorème 2.5
On suppose que la fonction de pondération π est une métrique sur $\Sigma \cup \{\varepsilon\}$ vérifiant la propriété :

$$\forall a,b \in \Sigma \qquad a \neq b \Rightarrow \pi(a,\varepsilon) < \pi(a,b) + \pi(b,\varepsilon)\,. \tag{$*$}$$

Alors, le problème de décision d_{E}^{π}-Center$_D$, associé au problème du centre pour la distance d'édition pondérée par π, est NP*-difficile et* W[1]*-difficile pour le paramètre $\#W$. De plus, ces résultats restent vrais même si l'alphabet Σ est de cardinal* 2.

Preuve. On pose $\ell := d_{\mathrm{E}}^{\pi}(\,\cdot\,,\varepsilon)$. Par le lemme 2.13 p. 51, ℓ est une longueur pondérée sur $\Sigma^\star$. Ainsi, d'après le théorème 2.4 p. 49, il suffit de réduire ℓ-WCS^0 (paramétré par $\#X$) à d_{E}^{π}-Center$_D$ (paramétré par $\#W$) pour démontrer notre théorème 2.5. Le gadget est similaire à celui utilisé dans la preuve du théorème 2.2 p. 42.

Soit (X,k) une instance de ℓ-WCS^0 : k est un entier naturel et X est une partie finie et non vide de $\Sigma^\star$ telle que pour tout $x \in X$, on ait $d_{\mathrm{E}}^{\pi}(x,\varepsilon) = 2k$. On construit l'instance (W,k) de d_{E}^{π}-Center$_D$ où $W := X \cup \{\varepsilon\}$ est un langage sur Σ.

Trivialement, cette réduction prend un temps polynomial. De plus, on peut supposer sans perte de généralité que $\varepsilon \notin X$ et donc que $\#W = \#X + 1$. Ainsi, le paramètre $\#W$ de l'instance réduite (W,k) n'est fonction que du paramètre $\#X$ de l'instance initiale (X,k). Il ne reste plus qu'à vérifier que notre transformation est bien une M-réduction. Plus précisément, nous allons montrer que, pour tout $g \in \Sigma^\star$ on a : g est une sous-séquence commune de X vérifiant $d_{\mathrm{E}}^{\pi}(g,\varepsilon) = k$ si et seulement si $\max_{w\in W} d_{\mathrm{E}}^{\pi}(g,w)$ est au plus égal à k.

• Supposons que g soit une sous-séquence commune de X vérifiant $d_{\mathrm{E}}^{\pi}(g,\varepsilon) = k$.

Pour tout $x \in X$, g est une sous-séquence de x, donc le point (ii) du lemme 2.12 p. 49 garantit

$$d_{\mathrm{E}}^{\pi}(g,x) = d_{\mathrm{E}}^{\pi}(x,g) = d_{\mathrm{E}}^{\pi}(x,\varepsilon) - d_{\mathrm{E}}^{\pi}(g,\varepsilon) = 2k - k = k\,.$$

On a ainsi $d_{\mathrm{E}}^{\pi}(g,w) = k$ pour tout $w \in W$, d'où $\max_{w\in W} d_{\mathrm{E}}^{\pi}(g,w) = k$.

• Réciproquement, supposons $\max_{w\in W} d_{\mathrm{E}}^{\pi}(g,w) \leq k$.

Soit $x \in X$. On a :

$$\begin{array}{ll}
k \geq d_{\mathrm{E}}^{\pi}(g,x) & \text{car } x \in W \\
\quad = d_{\mathrm{E}}^{\pi}(x,g) & \\
\quad \geq d_{\mathrm{E}}^{\pi}(x,\varepsilon) - d_{\mathrm{E}}^{\pi}(g,\varepsilon) & \text{par le point } (i) \text{ du lemme 2.12 p. 49} \\
\quad \geq 2k - d_{\mathrm{E}}^{\pi}(g,\varepsilon) & \text{car } X \text{ est une instance de } \ell\text{-WCS}^0 \\
\quad \geq 2k - k & \text{car } \varepsilon \in W \\
\quad = k\,. &
\end{array}$$

Ainsi, toutes les inégalités apparaissant ci-dessus sont en fait des égalités. On en déduit, d'une part, que $2k - d_{\mathrm{E}}^{\pi}(g,\varepsilon) = k$ d'où $d_{\mathrm{E}}^{\pi}(g,\varepsilon) = k$ et, d'autre part, que $d_{\mathrm{E}}^{\pi}(x,g) = d_{\mathrm{E}}^{\pi}(x,\varepsilon) - d_{\mathrm{E}}^{\pi}(g,\varepsilon)$, d'où g est une sous-séquence de x par le point (iii) du lemme 2.4 p. 41.

On a ainsi montré que g est une sous-séquence commune de X telle que $d_{\mathrm{E}}^{\pi}(g,\varepsilon) = k$.

Q.E.D.

2.7 Un algorithme exact pour le problème du centre pour la distance d'édition

Nous décrivons ici un algorithme exact pour le problème du centre pour n'importe quelle distance d'édition. Cet algorithme est vaguement esquissé par Ravi et Kececioglu dans le cas de la particulier de la distance de Levenshtein [99]. Il possède une propriété théorique intéressante : celle d'être polynomial lorsque le nombre m de mots pris en entrée est fixé. En revanche, il est inefficace en pratique, même si m est petit. Ceci n'est pas étonnant, compte tenu du théorème 2.5 p. 51 qui garantit que le problème du centre est en général W[1]-difficile pour le paramètre m. Notons qu'il ne semble pas possible de se ramener à l'algorithme générique d'alignement multiple de la section 2.4.1.

Dans toute la suite de la section 2.7,

– Σ désigne un alphabet fini et

– $\pi : (\Sigma \cup \{\varepsilon\}) \times (\Sigma \cup \{\varepsilon\}) \to \mathbb{Z}$ est une application quelconque.

On se donne à résoudre le problème d_{E}^{π}-Center, c'est-à-dire le problème du centre pour la distance d'édition d_{E}^{π}, pondérée par π. On pose de plus :

$$\kappa := \max_{\alpha,\beta\in\Sigma\cup\{\varepsilon\}} |\pi(\alpha,\beta)|$$

de manière à obtenir une constante positive vérifiant $-\kappa \leq \pi(\alpha, \beta) \leq +\kappa$ pour tous α, $\beta \in \Sigma \cup \{\varepsilon\}$.

Avant de décrire notre algorithme (section 2.7.2), nous avons besoin de quelques résultats préliminaires.

2.7.1 Résultats préliminaires

Si le coût de la délétion d'au moins une lettre de l'alphabet Σ est strictement négatif, alors aucun langage fini sur Σ n'admet de centre pour la distance d'édition :

Remarque 2.4 *On suppose qu'il existe $a \in \Sigma$ tel que $\pi(a, \varepsilon) < 0$.*

Alors, on a $\lim_{n\to\infty} d_{\mathrm{E}}^{\pi}(a^n, w) = -\infty$ pour tout $w \in \Sigma^\star$. Il en résulte que, pour toute partie finie $W \subseteq \Sigma^\star$, on a $\mathrm{R}_{d_{\mathrm{E}}^{\pi}}(W) = -\infty$ et, par suite, W n'admet pas de centre pour d_{E}^{π}.

En revanche, si le coût de toute délétion de lettre est positif ou nul, alors on peut obtenir la borne suivante et en déduire que tout langage fini et non vide admet un centre pour la distance d'édition :

Proposition 2.10
Supposons que la fonction de pondération π vérifie $\pi(a, \varepsilon) \geq 0$ pour tout $a \in \Sigma$.

Alors,

(i). pour tous g, $w \in \Sigma^\star$, on a $d_{\mathrm{E}}^{\pi}(g, w) \geq -\kappa\,|w|$, et

(ii). toute partie finie et non vide de $\Sigma^\star$ admet un centre pour d_{E}^{π}.

Preuve.

(i). Soit un alignement

$$\begin{bmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_n \\ \beta_1 & \beta_2 & \cdots & \beta_n \end{bmatrix}$$

de (g, w) de π-score optimal

$$d_{\mathrm{E}}^{\pi}(g, w) = \sum_{j=1}^{n} \pi(\alpha_j, \beta_j)\,.$$

Minorons cette somme en minorant individuellement chacun de ses n termes.

Notons j_1, j_2, …, $j_{|w|}$ les $|w|$ indices vérifiant :

$$1 \leq j_1 < j_2 < \cdots < j_{|w|} \leq n \qquad \text{et} \qquad w = \beta_{j_1}\beta_{j_2}\ldots\beta_{j_{|w|}}\,.$$

- Pour chacun des $|w|$ indices $j \in \left\{j_1, j_2, \ldots, j_{|w|}\right\}$ on a trivialement $\pi(\alpha_j, \beta_j) \geq -\kappa$, par définition de κ.
- Pour chacun des $n - |w|$ indices $j \in [1, n] \setminus \left\{j_1, j_2, \ldots, j_{|w|}\right\}$ on a $\beta_j = \varepsilon$ donc $\alpha_j \in \Sigma$ et, par suite, l'hypothèse faite sur la fonction de pondération π dans cette proposition garantit $\pi(\alpha_j, \beta_j) = \pi(\alpha_j, \varepsilon) \geq 0$.

Il en résulte :

$$d_{\mathrm{E}}^{\pi}(g, w) \geq (-\kappa) \cdot |w| + 0 \cdot (n - |w|) = -\kappa\,|w|\ ,$$

ce qu'on voulait à ce stade.

(ii). Soit une partie finie $W \subseteq \Sigma^\star$. Posons $\ell := \min_{w\in W} |w|$.

Le point (i) permet de minorer par $-\kappa\ell$ l'ensemble d'entiers

$$\left\{ \max_{w\in W} d_{\mathrm{E}}^{\pi}(g, w) : g \in \Sigma^\star \right\}\,.$$

Cet ensemble admet donc un plus petit élément, ce qui signifie que W admet bien au moins un centre pour d_{E}^{π}.

Q.E.D.

Nous aurons également besoin de la borne suivante :

Proposition 2.11
Pour toute partie finie $W \subseteq \Sigma^\star$, on a $\mathrm{R}_{d_{\mathrm{E}}^\pi}(W) \leq \kappa M$ où $M := \max_{w\in W} |w|$.

Preuve. Pour tout $w \in W$, la remarque 1.3 p. 22 permet d'écrire :

$$d_{\mathrm{E}}^\pi(\varepsilon, w) = \sum_{i=1}^{|w|} \pi(w[i], \varepsilon) \leq \kappa\,|w| \leq \kappa M$$

d'où :

$$\mathrm{R}_{d_{\mathrm{E}}^\pi}(W) \leq \max_{w\in W} d_{\mathrm{E}}^\pi(\varepsilon, w) \leq \kappa M\,.$$

Q.E.D.

2.7.2 Description de l'algorithme

On suppose donnés un entier $m \geq 1$ et m mots quelconques $w_1, w_2, \ldots, w_m \in \Sigma^\star$. Dans cette section, nous décrivons un algorithme calculant un centre et le rayon de $W := \{w_1, w_2, \ldots, w_m\}$ pour la distance d'édition pondérée par π. Cet algorithme est polynomial à m fixé. Il est fondé sur le principe de programmation dynamique.

Dans toute la suite de la section 2.7, on fait l'hypothèse suivante :

$$\forall a \in \Sigma \qquad \pi(a, \varepsilon) \geq 0\,. \tag{$**$}$$

D'après la remarque 2.4 p. 53, cette hypothèse est nécessaire pour que toute partie finie de $\Sigma^\star$ admette un centre pour d_{E}^π. Elle est également suffisante par le point (ii) de la proposition 2.10 p. 53.

2.7.2.1 Table de programmation dynamique

Notre algorithme calcule dynamiquement une partie finie de la table $2m$-dimensionnelle de booléens définie ci-dessous, au lieu d'un tableau m-dimensionnel d'entiers comme dans la section 2.4.1.2.

On note :

$$P := [0, |w_1|] \times [0, |w_2|] \times \cdots \times [0, |w_m|]$$

et, pour tout $\mathbf{p} = (p_1, p_2, \ldots, p_m) \in P$ et tout $\mathbf{d} = (d_1, d_2, \ldots, d_m) \in \mathbb{Z}^m$, on pose :

$$B[\mathbf{p}][\mathbf{d}] := \begin{cases} \texttt{oui} & \text{si } \exists g \in \Sigma^\star,\ \forall i \in [1, m],\ d_{\mathrm{E}}^\pi\left(g, w_i^{(p_i)}\right) \leq d_i \\ \texttt{non} & \text{sinon} \end{cases}.$$

Expliquons comment la table $B[\,\cdot\,][\,\cdot\,]$ peut être utilisée pour résoudre le problème du calcul du rayon de W. On peut remarquer dès maintenant que le rayon de W pour d_{E}^π peut être calculé uniquement à partir de la table booléenne m-dimensionnelle $B[(|w_1|, |w_2|, \ldots, |w_m|)][\,\cdot\,]$:

Remarque 2.5 *Le rayon de W pour d_{E}^π est la borne inférieure des entiers s'écrivant sous la forme $\max_{1\leq i\leq m} d_i$ avec $\mathbf{d} = (d_1, d_2, \ldots, d_m) \in \mathbb{Z}^m$ vérifiant $B[(|w_1|, |w_2|, \ldots, |w_m|)][\mathbf{d}] = \texttt{oui}$.*

Posons :

$$M := \max\{|w_1|, |w_2|, \ldots, |w_m|\}$$

et

$$D := [-\kappa\,|w_1|, \kappa M] \times [-\kappa\,|w_2|, \kappa M] \times \cdots \times [-\kappa\,|w_m|, \kappa M]\,.$$

La proposition 2.12 ci-dessous se démontre à l'aide des résultats de la section 2.7.1. Cette proposition garantit que, pour calculer le rayon de W, il suffit d'examiner un nombre *fini* d'entrées de la table $B[\,\cdot\,][\,\cdot\,]$, contrairement à ce que suggère la remarque 2.5 p. 54. Ce nombre d'entrées est au plus égal au cardinal de D que l'on peut majorer par $(2\kappa M)^m$. Il est donc polynomial en la taille de l'instance W pour m fixé.

Proposition 2.12
Le rayon de W pour d_{E}^{π} est le plus petit entier s'écrivant sous la forme $\max_{1\leq i\leq m} d_i$ avec $(d_1, d_2, \ldots, d_m) \in D$ vérifiant

$$B[(|w_1|, |w_2|, \ldots, |w_m|)][(d_1, d_2, \ldots, d_m)] = \texttt{oui}\,.$$

Preuve. Soit $\mathbf{d} = (d_1, d_2, \ldots, d_m) \in D$ avec

$$B[(|w_1|, |w_2|, \ldots, |w_m|)][\mathbf{d}] = \texttt{oui}\,, \tag{2.28}$$

tel que $R := \max_{1\leq i\leq m} d_i$ soit minimum. Il s'agit de montrer que $\mathrm{R}_{d_{\mathrm{E}}^{\pi}}(W) = R$.

On déduit de (2.28) qu'il existe $g_{\mathbf{d}} \in \Sigma^{\star}$ tel que, pour tout $i \in [1, m]$, on ait

$$d_{\mathrm{E}}^{\pi}(g_{\mathbf{d}}, w_i) = d_{\mathrm{E}}^{\pi}\left(g_{\mathbf{d}}, w_i^{(|w_i|)}\right) \leq d_i \leq R\,.$$

On a ainsi

$$\mathrm{R}_{d_{\mathrm{E}}^{\pi}}(W) \leq \max_{1\leq i\leq m} d_{\mathrm{E}}^{\pi}(g_{\mathbf{d}}, w_i) \leq R$$

donc il ne reste plus qu'à vérifier l'inégalité inverse, c'est-à-dire $R \leq \mathrm{R}_{d_{\mathrm{E}}^{\pi}}(W)$.

D'après le point (ii) de la proposition 2.10 p. 53, W admet un centre g pour d_{E}^{π}. On a trivialement :

$$B[(|w_1|, |w_2|, \ldots, |w_m|)]\,[(d_{\mathrm{E}}^{\pi}(g, w_1), d_{\mathrm{E}}^{\pi}(g, w_2), \ldots, d_{\mathrm{E}}^{\pi}(g, w_m))] = \texttt{oui}\,.$$

D'autre part, le point (i) de la proposition 2.10 p. 53 et la proposition 2.11 p. 54 garantissent que

$$(d_{\mathrm{E}}^{\pi}(g, w_1), d_{\mathrm{E}}^{\pi}(g, w_2), \ldots, d_{\mathrm{E}}^{\pi}(g, w_m)) \in D\,.$$

La minimalité de R permet alors d'écrire

$$R \leq \max_{1\leq i\leq m} d_{\mathrm{E}}^{\pi}(g, w_i) = \mathrm{R}_{d_{\mathrm{E}}^{\pi}}(W)\,.$$

Q.E.D.

La première étape de notre algorithme pour d_{E}^{π}-CENTER consiste à calculer dynamiquement $B[\mathbf{p}][\mathbf{d}]$ pour tout $(\mathbf{p}, \mathbf{d}) \in P \times D$. D'après la proposition 2.12 p. 55, on pourra alors calculer facilement le rayon de W pour d_{E}^{π}. Dans un second temps, on reconstruira un centre de W à l'aide de la méthode classique du "backtracking".

2.7.2.2 Caractérisation inductive de la table de programmation dynamique

Dans cette section nous montrons exhibons la caractérisation inductive de $B[\,\cdot\,][\,\cdot\,]$ sur laquelle s'appuie notre algorithme de programmation dynamique.

Proposition 2.13 (Initialisation)
Soit $\mathbf{d} = (d_1, d_2, \ldots, d_m) \in \mathbb{Z}^m$.

Alors, $B[(0, 0, \ldots, 0)][\mathbf{d}] = \texttt{oui}$ si et seulement si on a $d_i \geq 0$ pour tout $i \in [1, m]$.

Preuve.

• Supposons que l'on ait $d_i \geq 0$ pour tout $i \in [1, m]$. La matrice à 2 lignes et 0 colonne, dont le π-score est nul, est l'unique alignement de $(\varepsilon, \varepsilon)$ donc on a :

$$\forall i \in [1, m] \qquad d_{\mathrm{E}}^{\pi}\left(\varepsilon, w_i^{(0)}\right) = d_{\mathrm{E}}^{\pi}\left(\varepsilon, \varepsilon\right) = 0\,.$$

Il en résulte que $B[(0, 0, \ldots, 0)][\mathbf{d}] = \texttt{oui}$.

• Réciproquement, supposons que l'on ait $B[(0,0,\ldots,0)][\mathbf{d}] = \texttt{oui}$. Alors, il existe $g \in \Sigma^\star$ vérifiant $d_{\mathrm{E}}^{\pi}\left(g, w_i^{(0)}\right) \leq d_i$ pour tout $i \in [1,m]$. On a ainsi :

$$d_i \geq d_{\mathrm{E}}^{\pi}\left(g, w_i^{(0)}\right) = d_{\mathrm{E}}^{\pi}(g,\varepsilon) = \sum_{j=1}^{|g|} \pi(g[j],\varepsilon)$$

d'après la remarque 1.3 p. 22. Or, l'hypothèse $(**)$ p. 54 assure que tous les $\pi(g[j],\varepsilon)$ $(j \in [1,|g|])$ sont positifs ou nuls donc il en est de même pour leur somme. On en déduit $d_i \geq 0$.

Q.E.D.

Pour tout $i \in [1,m]$, on note $\mathbf{e}_i$ le m-uplet dans toutes les composantes sont nulles sauf la i-ème qui est égale à 1 :

$$\forall (x_1,x_2,\ldots,x_m) \in \mathbb{Z}^m \qquad \sum_{i=1}^{m} x_i \mathbf{e}_i = (x_1,x_2,\ldots,x_m)\,.$$

Théorème 2.6 (Induction)
Soient $\mathbf{p} = (p_1,p_2,\ldots,p_m) \in P$ *avec* $\mathbf{p} \neq (0,0,\ldots,0)$, $\mathbf{d} \in \mathbb{Z}^m$ *et* $I := \{i \in [1,m] : p_i \neq 0\}$.

Alors, on a $B[\mathbf{p}][\mathbf{d}] = \texttt{oui}$ *si et seulement si l'une au moins des deux assertions suivantes est vérifiée :*

1. *il existe* $j \in I$ *tel que*
$$B[\mathbf{p} - \mathbf{e}_j][\mathbf{d} - \pi(\varepsilon, w_j[p_j])\mathbf{e}_j] = \texttt{oui} \qquad (E_j)$$
ou
2. *il existe* $a \in \Sigma$ *et une partie non vide* $J \subseteq I$ *tels que*
$$B\left[\mathbf{p} - \sum_{i\in J} \mathbf{e}_i\right]\left[\mathbf{d} - \sum_{i\in[1,m]\setminus J} \pi(a,\varepsilon)\mathbf{e}_i - \sum_{i\in J} \pi(a,w_i[p_i])\mathbf{e}_i\right] = \texttt{oui}\,. \qquad (F_{a,J})$$

La preuve de ce théorème est longue mais ne fait intervenir que des idées simples. Elle est reléguée en annexe, page 121.

2.7.2.3 Calcul de la table de programmation dynamique

On déduit du théorème 2.6 p. 56 et des résultats précédents, le corollaire suivant.

Corollaire 2.1
Supposons donné un couple $(\mathbf{p},\mathbf{d}) \in P \times D$ *vérifiant* $\mathbf{p} \neq (0,0,\ldots,0)$ *et l'hypothèse suivante : pour tout* $(\mathbf{p}',\mathbf{d}') \in P \times D$ *avec* $\mathbf{p}'$ *strictement avant* $\mathbf{p}$ *dans l'ordre lexicographique,* $B[\mathbf{p}'][\mathbf{d}']$ *est calculable en temps* $O(m)$ *(linéaire en la taille de* $\mathbf{p}'$ *et* $\mathbf{d}'$*).*

Alors, on peut calculer $B[\mathbf{p}][\mathbf{d}]$ *en temps* $O(m2^m)$.

Preuve. Vérifions tout d'abord que l'on peut légèrement renforcer l'hypothèse de notre corollaire. Posant

$$\bar{D} := \{d'_1,d'_2,\ldots,d'_m) \in \mathbb{Z}^m : \forall i \in [1,m]\,,\ d'_i \leq \kappa M\}\ ,$$

on a $D \subseteq \bar{D}$, et $\bar{D} \setminus D$ est l'ensemble des $(d'_1,d'_2,\ldots,d'_m) \in \mathbb{Z}^m$ pour lesquels il existe $i \in [1,m]$ vérifiant $d'_i < -\kappa\,|w_i|$. On déduit ainsi facilement du point (i) de la proposition 2.10 p. 53 la remarque suivante.

Remarque 2.6 *Pour tout* $\mathbf{p}' \in P$ *et tout* $\mathbf{d}' \in \bar{D} \setminus D$, *on a* $B[\mathbf{p}'][\mathbf{d}'] = \texttt{non}$.

On peut ainsi remplacer D par $\bar{D}$ dans l'énoncé du corollaire 2.1. Autrement dit, on peut considérer que $B[\mathbf{p}'][\mathbf{d}']$ est calculable en temps $O(m)$ pour tout $(\mathbf{p}',\mathbf{d}') \in P \times \bar{D}$ tel que $\mathbf{p}'$ soit strictement avant $\mathbf{p}$ dans l'ordre lexicographique.

Reprenons les notations du théorème 2.6 p. 56 : $\mathbf{p} = (p_1,p_2,\ldots,p_m)$ et $I = \{i \in [1,m] : p_i \neq 0\}$. Ce théorème permet de calculer $B[\mathbf{p}][\mathbf{d}]$ à l'aide d'entrées de $B[\,\cdot\,][\,\cdot\,]$ calculables en temps $O(m)$, comme l'atteste les deux remarques suivantes.

Remarque 2.7 *Considérons les deux m-uplets d'entiers*

$$\mathbf{p} - \mathbf{e}_j \qquad et \qquad \mathbf{p} - \sum_{i \in J} \mathbf{e}_i \,,$$

qui interviennent respectivement dans les égalités (E_j) et $(F_{a,J})$ de l'énoncé du théorème 2.6 p. 56.

Dès que J est non vide, ces deux m-uplets sont situés strictement avant $\mathbf{p}$ dans l'ordre lexicographique.

Remarque 2.8 *Considérons les deux m-uplets d'entiers*

$$\mathbf{d} - \pi(\varepsilon, w_j[p_j])\mathbf{e}_j \qquad et \qquad \mathbf{d} - \sum_{i \in [1,m] \setminus J} \pi(a, \varepsilon)\mathbf{e}_i - \sum_{i \in J} \pi(a, w_i[p_i])\mathbf{e}_i$$

qui interviennent respectivement dans les égalités (E_j) et $(F_{a,J})$ de l'énoncé du théorème 2.6 p. 56.

Si $\mathbf{d} \in \bar{D}$ alors ces deux m-uplets sont des éléments de $\bar{D}$.

Pour calculer $B[\mathbf{p}][\mathbf{d}]$ à l'aide de l'équivalence fournie par le théorème 2.6 p. 56 :

1. on teste l'égalité (E_j) pour les $\#I$ éléments j de I, et
2. on teste l'égalité $(F_{a,J})$ pour au plus $(\#\Sigma)(2^{\#I} - 1)$ couples (a, J) distincts.

Chacun des $\#I + (\#\Sigma)(2^{\#I} - 1)$ tests évoqués ci-dessus est réalisable en temps $O(m)$ d'après ce qui précède. Comme I est une partie de $[1, m]$, on a :

$$\#I + (\#\Sigma)(2^{\#I} - 1) \leq m + (\#\Sigma)(2^m - 1) = O(2^m)$$

donc on peut calculer $B[\mathbf{p}][\mathbf{d}]$ en temps $O(m2^m)$. **Q.E.D.**

Notre algorithme fonctionne de la manière suivante.

1. Pour tout $\mathbf{d} \in D$, on calcule $B[(0, 0, \ldots, 0)][\mathbf{d}]$ en temps $O(m)$ en examinant le signe de chacune des entrées de $\mathbf{d}$, comme le suggère la proposition 2.13 p. 55. Cette étape prend un temps $O(m \cdot \#D) \subseteq O(m(2\kappa M)^m)$.
2. On énumère les m-uplets $\mathbf{p} \in P$ avec $\mathbf{p} \neq (0, 0, \ldots, 0)$ dans l'ordre lexicographique. Pour chaque $\mathbf{p}$, on calcule, pour chaque $\mathbf{d} \in D$, $B[\mathbf{p}][\mathbf{d}]$ en temps $O(m2^m)$, comme le permet le corollaire 2.1 p. 56.
3. D'après la proposition 2.12 p. 55, il suffit de faire une passe sur la restriction à D de $B[(|w_1|, |w_2|, \ldots, |w_m|)][\cdot]$ pour obtenir le rayon de W pour d_{E}^{π}. Cette étape prend un temps $O(m \cdot \#D) \subseteq O(m(2\kappa M)^m)$.
4. À l'aide d'un "backtracking", on récupère un centre de W pour d_{E}^{π}.

L'étape la plus coûteuse en temps est la deuxième : il y a $(\#P - 1)(\#D)$ entrées de $B[\cdot][\cdot]$ à calculer et on passe un temps $O(m2^m)$ sur chacune. Notre algorithme est donc de complexité $O(m2^m(\#P - 1)(\#D))$. Cette complexité est bien polynomiale à m fixé car on a

$$\begin{aligned} m2^m(\#P - 1)(\#D) &\leq m2^m \cdot ((M + 1)^m - 1)) \cdot (2\kappa M)^m \\ &= O\Big(m\kappa^m (2M)^{2m}\Big) \,. \end{aligned}$$

2.8 Conclusion

2.8.1 Récapitulatif des résultats connus sur la complexité des problèmes d'alignement multiple

Chacun des problèmes d'alignement multiple évoqués dans la section 2.4 admet un algorithme exact dont la complexité devient polynomiale lorsque le nombre de mots pris en entrée est fixé (voir les sections 2.4.1 et 2.7).

En revanche, les problèmes décision associés sont NP-complets même dans le cas où l'alphabet d'entrée est binaire (voir [32] et les sections 2.5 et 2.6).

Supposons que la fonction de pondération π soit une semi-métrique sur l'alphabet d'entrée Σ augmenté du mot vide. Alors, les problèmes SP$^\pi$-MSA, d_{E}^π-Median et d_{E}^π-Center sont 2-approximables [45, 10] (voir aussi les propositions 2.3 p. 28 et 2.4 p. 29). Si de plus le degré des arbres phylogénétiques pris en entrée est borné alors Tree$^\pi$-MSA admet un P.T.A.S [113]. Mais, on peut trouver une fonction de pondération π telle que SP$^\pi$-MSA (resp. d_{E}^π-Median) n'admette pas de P.T.A.S., sauf si NP = P [58] (resp. [114]).

2.8.2 Questions ouvertes

Le problème du centre pour la distance d'édition. La complexité du problème du centre pour les distances d'édition dont la fonction de pondération π est une métrique sur $\Sigma \cup \{\varepsilon\}$ ne satisfaisant pas $(*)$ p. 39 reste ouverte. Il est probable qu'il faille mettre en œuvre des idées nouvelles par rapport à celles développées dans la section 2.6. En effet, quand Isaac Elias a démontré, en 2003, la NP-complétude de d_{E}^π-Median pour toute métrique π sur $\{\mathtt{0}, \mathtt{1}, \varepsilon\}$, il a été est obligé de considérer séparément les cas $\pi(\mathtt{0},\mathtt{1}) < \pi(\mathtt{0},\varepsilon) + \pi(\varepsilon,\mathtt{1})$ et $\pi(\mathtt{0},\mathtt{1}) = \pi(\mathtt{0},\varepsilon) + \pi(\varepsilon,\mathtt{1})$ [32].

Approximabilité. La question de l'existence d'un P.T.A.S. reste ouverte pour d_{L}-Median, d_{L}-Center et SP$^\pi$-MSA lorsque π est la métrique discrète sur $\Sigma \cup \{\varepsilon\}$.

On sait que d_{E}^π-Median et SP$^\pi$-MSA sont difficiles à approximer pour certaines fonctions de pondération π mais ces fonctions font jouer des rôles très différents aux lettres de l'alphabet (contrairement à la métrique discrète sur $\Sigma \cup \{\varepsilon\}$) [114, 58].

En revanche, pour une légère variante de la distance de Levenshtein, le problème de la médiane admet un P.T.A.S. [73].

Complexité paramétrique. Pour la distance de Hamming, le problème du centre a été montré F.P.T. pour son paramètre "seuil d'acceptation" [44] (voir aussi la section 2.3.2). Pour la distance Levenshtein, cette question reste ouverte. Plus explicitement, supposons donnés un langage fini W et un entier $d \geq 0$. On ne sait pas s'il est possible décider de l'existence d'un mot g vérifiant $\max_{w \in W} d_{\mathrm{L}}(g,w) \leq d$ en temps $O\Big(\varphi(d)\,|W|^{O(1)}\Big)$, même si l'application $\varphi : \mathbb{N} \to \mathbb{N}$ peut-être choisie arbitrairement, indépendamment de W et de d. Notons que l'on ne connaît pas non plus la complexité de d_{L}-Median$_D$ pour son paramètre "seuil d'acceptation".

Chapitre 3

Complexité du problème de la plus longue sous-séquence commune pour les mots non orientés et/ou circulaires

3.1 Introduction

De nombreux objets, rencontrés par exemple en *reconnaissance des formes* et en *bio-informatique*, sont naturellement modélisés par des séquences circulaires et/ou non orientés de symboles. Pour comparer ces mots sans début ni fin et/ou dont on a "oublié" le sens de lecture, il est souvent pertinent de les aligner.

3.1.1 Application des alignements de mots circulaires et/ou non orientés

Recensons quelques situations concrètes où il est intéressant d'aligner des mots circulaires et/ou non orientés.

3.1.1.1 En bio-informatique

Dans la nature, l'information héréditaire est portée par des molécules linéaires ou *circulaires* d'ADN ou d'ARN [1]. Ces molécules sont représentées par des mots linéaires ou circulaires sur l'alphabet $\{\mathtt{A}, \mathtt{T}, \mathtt{U}, \mathtt{G}, \mathtt{C}\}$.

- Les génomes bactériens, chloroplastiques et mitochondriaux sont en majorité circulaires.
- Le matériel génétique des virus est souvent porté par des molécules d'ADN circulaires. C'est le cas en particulier des bactériophages, qui sont largement utilisés comme vecteurs pour cloner les gènes de diverses espèces.
- Les plasmides sont de petites molécules circulaires, capables de se répliquer de manière autonome. Ils ont de nombreuses applications bio-technologiques.

Par ailleurs, l'*orientation* des molécules d'ADN et d'ARN (par convention dans le sens $5' \to 3'$) peut poser des problèmes pour le séquençage. Au cours de ce processus, maintenant automatisé, l'ADN que l'on traite est inséré dans un vecteur double brin pour y être cloné. Or, pour des raisons techniques, il n'est pas toujours possible de savoir dans quel sens, la molécule considérée est insérée dans le vecteur, c'est-à-dire sur quel brin. Par exemple, pour comparer des EST (Expressed Sequence Tags), il est nécessaire de tenir compte des deux orientations possibles.

3.1.1.2 En reconnaissance des formes

Le contour d'une forme planaire peut être représenté par un mot *circulaire*. Une bonne manière pour comparer deux formes est de calculer le score minimum d'un alignement circulaire entre leurs contours [96, 83, 17]. En pratique, ce type de comparaison est utilisé dans l'industrie pour classifier des objets défilants sur un tapis roulant. Si le côté sur lequel les objets sont posés peut être arbitraire alors son contour peut

être encodé dans un sens ou dans l'autre. Il est donc nécessaire d'effectuer une comparaison circulaire et non *orientée*.

3.1.2 Complexité du problème de l'alignement multiple pour les mots circulaires et/ou non orientés

3.1.2.1 État de l'art sommaire

Il existe des algorithmes polynomiaux calculant, pour deux mots circulaires donnés, un alignement de score optimum [80, 102] (voir section 3.3.1). Des heuristiques plus rapides que les algorithmes exacts sont également utilisées en reconnaissance de formes [96, 87].

Pour un nombre quelconque de mots circulaires, la NP-complétude du problème de la plus courte super-séquence commune à été établie [85] (voir section 3.3.2). Ce dernier problème d'optimisation peut être vu comme un cas particulier du problème de l'alignement multiple de score optimum.

3.1.2.2 Description sommaire de notre contribution

D'après l'exemple 2.9 p. 35, le problème de la plus longue sous-séquence commune (LCS) peut être vu comme un cas particulier du problème de l'alignement multiple de score minimum pour les mots linéaires orientés. Dans ce chapitre, nous considérons trois variantes de LCS :

1. celle pour les mots *non orientés* (LCUS),
2. celle pour les mots *circulaires* (LCCS) et
3. celle pour les mots *circulaires et non orientés* (LCUCS).

Le problème LCUS (resp. LCCS, resp. LCUCS) peut être vu comme un cas particulier du problème de l'alignement de score minimum pour les mots non orientés (resp. circulaires, resp. circulaires et non orientés). Nous étudions de manière à peu près exhaustive la complexité algorithmique des problèmes LCUS, LCCS et LCUCS, tant du point de vue paramétrique, que de l'approximabilité.

3.2 Définitions et notations

Pour tout langage fini et non vide X, nous utiliserons, les deux notations suivantes :
- $M(X) := \max_{x \in X} |x|$ dans la section 3.4 et
- $\ell(X) := \min_{x \in X} |x|$ dans les sections 3.5 et 3.6.2.

3.2.1 Autour des mots circulaires et/ou non orientés

Dans cette section, nous expliquons comment définir formellement la notion de mot circulaire (orienté ou non). La notion de mot circulaire est naturellement liée à la notion suivante de conjugaison.

Définition 3.1 (Conjugaison)
Soient deux mots x et y. On dit que x est un conjugué *de y ou que x est* conjugué *avec y, et l'on note $x \overset{c}{\equiv} y$, lorsqu'il existe deux mots u, v tels que $x = uv$ et $y = vu$.*

Tous les conjugués de y sont de même longueur que y et, si y est non vide, y admet au plus $|y|$ conjugués distincts.

Un conjugué de y est souvent appelé *décalé circulaire* de y.

Définition 3.2
On note γ la fonction qui, à tout mot non vide w, associe son premier décalé circulaire *(vers la gauche) $\gamma(w) := w[2]w[3]\dots w[|w|]w[1]$. On convient que $\gamma(\varepsilon) := \varepsilon$.*

On vérifie facilement que γ induit une bijection de l'ensemble des mots dans lui-même.

Pour tout $n \in \mathbb{Z}$, on peut ainsi définir γ^n comme étant la fonction obtenue en composant entre eux

– n exemplaires de la fonction γ lorsque n est positif et,
– $-n$ exemplaires de la bijection réciproque de γ lorsque n est négatif.
Ainsi, γ^0 est la fonction identité sur les mots, $\gamma^1 = \gamma$, γ^{-k} est la bijection réciproque de γ^k pour tout $k \in \mathbb{Z}$ et, $\gamma^{m+n}(w) = \gamma^m(\gamma^n(w))$ pour tout mot w et tous m, $n \in \mathbb{Z}$.

Exemple 3.1 *Les conjugués de* $x := \texttt{allezlom}$ *sont :*

$$\gamma^0(x) = x\,, \qquad \gamma^1(x) = \texttt{llezloma}\,, \qquad \gamma^2(x) = \texttt{lezlomal}\,,$$
$$\gamma^3(x) = \texttt{ezlomall}\,, \qquad \gamma^4(x) = \texttt{zlomalle}\,, \qquad \gamma^5(x) = \texttt{lomallez}\,,$$
$$\gamma^6(x) = \texttt{omallezl}\,, \qquad \gamma^7(x) = \texttt{mallezlo}\,.$$

Les remarques 3.1 et 3.2 ci-après permettent de démontrer le lemme 3.1 p. 61 qui relie la notion de conjugaison et la fonction γ.

Remarque 3.1 *Pour tout entier* $p \geq 0$ *et tous mots* u, v *avec* $|u| = p$, *on a* $\gamma^p(uv) = vu$.

En particulier, on a $\gamma^{|u|}(u) = u$ ainsi que l'assertion équivalente $u = \gamma^{-|u|}(u)$. Par suite, on démontre facilement par récurrence que, pour tout entier $q \geq 0$, on a $\gamma^{q|u|}(u) = u$ et $u = \gamma^{-q|u|}(u)$. En substituant $\gamma^r(w)$ à u, on obtient, pour tous q, $r \in \mathbb{Z}$: $\gamma^{q|w|+r}(w) = \gamma^{q|w|}(\gamma^r(w)) = \gamma^r(w)$. On en déduit :

Remarque 3.2 *Pour tout* $n \in \mathbb{Z}$ *et tout mot* w, $\gamma^n(w) = \gamma^r(w)$ *où* r *est le reste de la division euclidienne de* $|w|$ *par* n.

Lemme 3.1 *Pour tous mots* x, y, *les deux assertions suivantes sont équivalentes :*

(*i*). $x \overset{\text{C}}{\equiv} y$,

(*ii*). *il existe* $n \in \mathbb{Z}$ *tel que* $x = \gamma^n(y)$ *et*

(*iii*). *il existe* $n \in [0, |y| - 1]$ *tel que* $x = \gamma^n(y)$.

Preuve. La remarque 3.1 p. 61 permet de prouver $(i) \iff (iii)$ et la remarque 3.2 p. 61 permet de prouver $(ii) \iff (iii)$. **Q.E.D.**

On déduit facilement la proposition suivante de l'équivalence des points (i) et (ii) du lemme 3.1 ci-dessus :

Proposition 3.1 ([76, 77])
La relation de conjugaison $\overset{\text{C}}{\equiv}$ *est une relation d'équivalence sur l'ensemble des mots.*

Preuve. Soient trois mots x, y, z. On s'appuie en permanence sur le lemme 3.1 p. 61.

• Posant $u := \varepsilon$ et $v := x$, on peut écrire $x = uv$ et $x = vu$ d'où $x \overset{\text{C}}{\equiv} x$: la relation $\overset{\text{C}}{\equiv}$ est réflexive.

• On déduit immédiatement de la définition 3.1 p. 60 que la relation $\overset{\text{C}}{\equiv}$ est symétrique.

• Supposons $x \overset{\text{C}}{\equiv} y$ et $y \overset{\text{C}}{\equiv} z$. Alors, il existe p, $q \in \mathbb{Z}$ tels que $x = \gamma^p(y)$ et $y = \gamma^q(z)$. Par suite, on a $x = \gamma^{p+q}(z)$ d'où $x \overset{\text{C}}{\equiv} z$: la relation $\overset{\text{C}}{\equiv}$ est transitive.

Q.E.D.

Pour tout mot w, on appelle *classe de conjugaison* de w, la classe d'équivalence de w pour $\overset{\text{C}}{\equiv}$. La classe de conjugaison de w est l'orbite de w sous γ, c'est-à-dire $\{w, \gamma(w), \gamma^2(w), \ldots, \gamma^{|w|-1}(w)\}$. On voit maintenant qu'il est cohérent de définir formellement le *mot circulaire orienté obtenu en joignant les deux extrémités de* w comme étant la classe de conjugaison de w.

Remarquons maintenant qu'identifier un mot x avec son image miroir $\tilde{x}$ revient à oublier son orientation. Ainsi, la notion de mot circulaire non orienté est naturellement liée à la relation suivante :

Définition 3.3
Soient deux mots x *et* y. *On dit que* x *est* conjugué avec y à l'orientation près *et l'on note* $x \overset{\text{UC}}{\equiv} y$, *lorsque* x *est conjugué avec* y *ou avec* $\tilde{y}$.

Notons que UC est l'abréviation de "Unoriented Cyclic". Pour tous mots x, y, on a : $x \overset{\text{UC}}{\equiv} y$ si et seulement si $x \overset{\text{C}}{\equiv} y$ ou $x \overset{\text{C}}{\equiv} \tilde{y}$.

La proposition 3.2 ci-après garantit que $\overset{\text{UC}}{\equiv}$ est une relation d'équivalence sur les mots. Ainsi, pour tout mot w, on peut définir formellement *le mot circulaire non orienté obtenu en joignant les deux extrémités de* w comme étant la classe d'équivalence pour $\overset{\text{UC}}{\equiv}$ de w.

Pour montrer la proposition 3.2 p. 62, nous avons besoin du lemme suivant :

Lemme 3.2 (Compatibilité avec le passage à l'image miroir) *Soient deux mots x et y. Alors, x est conjugué avec y si et seulement si $\tilde{x}$ est conjugué avec $\tilde{y}$.*

Preuve. Supposons $x \overset{\text{C}}{\equiv} y$. Alors, il existe des mots u, v tels que $x = uv$ et $y = vu$. On a ainsi $\tilde{x} = \tilde{v}\tilde{u}$ et $\tilde{y} = \tilde{u}\tilde{v}$, d'où $\tilde{x} \overset{\text{C}}{\equiv} \tilde{y}$.

On a ainsi montré l'implication : $x \overset{\text{C}}{\equiv} y \Rightarrow \tilde{x} \overset{\text{C}}{\equiv} \tilde{y}$. L'implication réciproque s'en déduit car le passage à l'image miroir est involutif (l'image miroir de l'image miroir d'un mot est le mot lui-même). **Q.E.D.**

On déduit de la proposition 3.1 p. 61 et du lemme 3.2 p. 62 :

Proposition 3.2
La relation $\overset{\text{UC}}{\equiv}$ est une relation d'équivalence sur les mots.

Preuve. On s'appuie en permanence sur la proposition 3.1 p. 61. Soient trois mots x, y, z.

• Comme tout mot est conjugué avec lui-même, on a $x \overset{\text{UC}}{\equiv} x$. Notre relation est réflexive.

• Supposons $x \overset{\text{UC}}{\equiv} y$ et montrons $y \overset{\text{UC}}{\equiv} x$. Deux cas se présentent : premièrement, si on a $x \overset{\text{C}}{\equiv} y$ alors on a $y \overset{\text{C}}{\equiv} x$; deuxièmement, si on a $x \overset{\text{C}}{\equiv} \tilde{y}$ alors, par le lemme 3.2 p. 62, on a $\tilde{x} \overset{\text{C}}{\equiv} y$, d'où $y \overset{\text{C}}{\equiv} \tilde{x}$. Dans les 2 cas, on a bien $y \overset{\text{UC}}{\equiv} x$. Notre relation est symétrique.

• Supposons $x \overset{\text{UC}}{\equiv} y$ et $y \overset{\text{UC}}{\equiv} z$. Montrons $x \overset{\text{UC}}{\equiv} z$. Quatre cas se présentent : premièrement, si on a $x \overset{\text{C}}{\equiv} y$ et $y \overset{\text{C}}{\equiv} z$ alors on a $x \overset{\text{C}}{\equiv} z$; deuxièmement, si on a $x \overset{\text{C}}{\equiv} y$ et $y \overset{\text{C}}{\equiv} \tilde{z}$ alors on a $x \overset{\text{C}}{\equiv} \tilde{z}$; troisièmement, si on a $x \overset{\text{C}}{\equiv} \tilde{y}$ et $y \overset{\text{C}}{\equiv} z$ alors on a $x \overset{\text{C}}{\equiv} \tilde{y}$ et $\tilde{y} \overset{\text{C}}{\equiv} \tilde{z}$ (lemme 3.2 p. 62), d'où $x \overset{\text{C}}{\equiv} \tilde{z}$; quatrièmement, si on a $x \overset{\text{C}}{\equiv} \tilde{y}$ et $y \overset{\text{C}}{\equiv} \tilde{z}$ alors on a $x \overset{\text{C}}{\equiv} \tilde{y}$ et $\tilde{y} \overset{\text{C}}{\equiv} z$ (lemme 3.2 p. 62), d'où $x \overset{\text{C}}{\equiv} z$. Dans les 4 cas, on a bien $x \overset{\text{UC}}{\equiv} z$. Notre relation est transitive.

Q.E.D.

3.2.2 Position des problèmes

Dans cette section, nous définissons formellement :
- la notion de sous-séquence circulaire et/ou non orientée (définition 3.4 p. 62) ainsi que
- les problèmes d'optimisation consistant à rechercher une sous-séquence circulaire et/ou non orientée, de longueur maximale, commune à tous les mots d'un langage donné (définition 3.6 p. 64).

Rappelons (voir section 1.4) que, pour tous mots x et s, on dit que s est une sous-séquence de x lorsqu'on peut obtenir s à partir de x en y effaçant des lettres. Le problème de la plus longue sous-séquence commune (LCS) consiste à rechercher un mot de longueur maximale qui est une sous-séquence de chacun des mots d'un langage donné.

Définition 3.4
Soient x et s deux mots quelconques.
- *On dit que s est une sous-séquence non orientée (ou* U-sous-séquence*) de x lorsque s est une sous-séquence de x ou l'image miroir d'une sous-séquence de x.*
- *On dit que s est une sous-séquence circulaire (ou* C-sous-séquence*) de x lorsque s est conjugué avec une sous-séquence de x.*
- *On dit que s est une sous-séquence circulaire non orientée (ou* UC-sous-séquence*) de x lorsque s est conjugué avec une sous-séquence de x ou avec l'image miroir d'une sous-séquence de x.*

Utilisons les notations introduites dans les définitions 3.1 p. 60 et 3.3 p. 61 : s est une C-sous-séquence (resp. une UC-sous-séquence) de x si et seulement s'il existe une sous-séquence t de x vérifiant $s \overset{\text{C}}{\equiv} t$ (resp. $s \overset{\text{UC}}{\equiv} t$).

La remarque suivante pourrait constituer une alternative à la définition 3.4.

Remarque 3.3 *Soient x et s deux mots quelconques :*
- *s est une U-sous-séquence de x si et seulement si s est une sous-séquence de x ou de $\tilde{x}$;*
- *s est une C-sous-séquence de x si et seulement si s est une sous-séquence d'un mot conjugué avec x ;*
- *s est une UC-sous-séquence de x si et seulement si s est une U-sous-séquence d'un mot conjugué avec x.*

Remarque 3.4 *Soit x un mot quelconque.*
- *Toute sous-séquence de x est à la fois une U-sous-séquence et une C-sous-séquence de x.*
- *Toute U-sous-séquence et toute C-sous-séquence de x est une UC-sous-séquence de x.*

Exemple 3.2 (Suite de l'exemple 3.1 p. 61) *Soit $x :=$* `allezlom`.
- `mela` *est U-sous-séquence de x qui n'est pas une C-sous-séquence de x.*
- `malle` *est une C-sous-séquence de x car on a* `malle` $\overset{\text{C}}{\equiv}$ `allem` *et* `allem` *est une sous-séquence de x. Mais* `malle` *n'est pas une U-sous-séquence de x.*
- `lamo` *est une UC-sous-séquence de x car on a* `lamo` $\overset{\text{C}}{\equiv}$ `mola` *et* `mola` *est l'image miroir de la sous-séquence* `alom` *de x. Mais* `lamo` *n'est ni une U-sous-séquence, ni une C-sous-séquence de x.*

Définition 3.5
Soit X un langage non vide.

On appelle U-sous-séquence (resp. C-sous-séquence, resp. UC-sous-séquence) commune *de X tout mot qui est une U-sous-séquence (resp. une C-sous-séquence, resp. une UC-sous-séquence) de chacun des mots de X.*

On note $\mathrm{lcus}(X)$ *(resp.* $\mathrm{lccs}(X)$*, resp.* $\mathrm{lcucs}(X)$*) la longueur d'une plus longue U-sous-séquence (resp. C-sous-séquence, resp. UC-sous-séquence) commune de X.*

Les notations lcus, lccs et lcucs ont été choisies car elles constituent des abréviations naturelles de "Longest Commun Unoriented Subsequence", "Longest Common Cyclic Subsequence" et "Longest Common Unoriented Cyclic Subsequence".

On déduit de la remarque 3.4 p. 63 :

Remarque 3.5 *Pour chaque langage non vide X, on a :*

$$\mathrm{lcs}(X) \leq \mathrm{lcus}(X) \leq \mathrm{lcucs}(X) \quad \text{et} \quad \mathrm{lcs}(X) \leq \mathrm{lccs}(X) \leq \mathrm{lcucs}(X)\,.$$

L'exemple suivant montre que les quatre inégalités ci-dessus peuvent être strictes et donc que la définition 3.5 p. 63 est pertinente. Il montre également que les fonctions lcus et lccs ne sont pas comparables.

Exemple 3.3 *On a :*

$$\begin{array}{rcccccccccc} \mathrm{lcs}(X_1) & = & 1 & < & 2 & = & \mathrm{lcus}(X_1) & = & \mathrm{lccs}(X_1) & = & \mathrm{lcucs}(X_1) \\ \mathrm{lcus}(X_2) & = & 2 & < & 3 & = & \mathrm{lccs}(X_2) & = & \mathrm{lcucs}(X_2) & & \\ \mathrm{lccs}(X_3) & = & 5 & < & 6 & = & \mathrm{lcus}(X_3) & = & \mathrm{lcucs}(X_3) & & \end{array}$$

pour $X_1 := \{\mathtt{01}, \mathtt{10}\}$, $X_2 := \{\mathtt{011}, \mathtt{101}\}$ *et* $X_3 := \{\mathtt{101001}, \mathtt{100101}\}$.

Introduisons maintenant les problèmes dont nous étudions la complexité dans ce chapitre :

Définition 3.6
On nomme LCUS *(resp.* LCCS, *resp.* LCUCS*) le problème d'optimisation :* "Étant donné un langage fini et non vide X, trouver une U-sous-séquence (resp. une C-sous-séquence, resp. une UC-sous-séquence) commune de X de longueur maximale".

Le problème de décision LCUS_D *(resp.* LCCS_D, *resp.* LCUCS_D*) associé à* LCUS *(resp.* LCCS, *resp.* LCUCS*) s'énonce :* "Étant donnés un langage fini et non vide X et un entier $k \geq 0$, décider si X admet une U-sous-séquence (resp. une C-sous-séquence, resp. une UC-sous-séquence) commune de longueur k".

3.3 Résultats connus et notre contribution

Plusieurs problèmes naturellement liés à LCCS ont été abordés dans littérature. Les principaux résultats connus sur ces problèmes sont exposés dans les sections 3.3.1 et 3.3.2 ci-après. Ensuite, dans la section 3.3.3, nous décrivons les résultats nouveaux que nous démontrons sur la complexité des problèmes LCUS, LCCS et LCUCS.

3.3.1 Alignement de deux mots circulaires et/ou non orientés

Le problème de l'alignement de deux mots circulaires a été étudié par : Maes [80] (voir aussi [22]), Landau, Myers et Schmidt [68], et Schmidt [102]. Ce problème est la généralisation naturelle de la restriction de LCCS à des paires de mots.

Définition 3.7
Soient deux mots x et y.
- *On appelle* alignement non orienté *(ou* U-alignement*) de (x, y) tout alignement de (x, y) ou de $(x, \tilde{y})$.*
- *On appelle* alignement circulaire *(ou* C-alignement*) de (x, y) tout alignement d'un couple de la forme (x, y') où y' est un mot conjugué avec y.*
- *On appelle* alignement circulaire non orienté *(ou* UC-alignement*) de (x, y) tout alignement d'un couple de la forme (x, y') où y' est un mot conjugué avec y ou avec $\tilde{y}$.*

Dans toute la suite de cette section, Σ désigne un alphabet fini ou infini et $\pi : (\Sigma \cup \{\varepsilon\}) \times (\Sigma \cup \{\varepsilon\}) \to \mathbb{Z}$ est une fonction de pondération calculable en temps constant.

On dit qu'un alignement (resp. un U-alignement, resp. un C-alignement, resp. un UC-alignement) de (x, y) est *π-optimal* lorsqu'il est de π-score minimum parmi tous les alignements (resp. les U-alignements, resp. les C-alignements, resp. les UC-alignements) de (x, y). Étant donnés deux mots non vides x, y sur Σ, nous considérons les trois problèmes consistant respectivement à rechercher : un U-alignement, un C-alignement ou un UC-alignement π-optimal de (x, y).

Rappelons que l'on peut calculer un alignement (classique) π-optimal de (x, y) en temps quadratique $O(|x|\,|y|)$ [45] (voir section 1.5.4).

3.3.1.1 Le problème de l'alignement circulaire de score minimum

Pour calculer un C-alignement π-optimal de (x, y), une méthode naïve consiste à

1. calculer un alignement π-optimal de (x, y') pour chaque mot y' conjugué avec y puis à
2. retourner un alignement de π-score minimum parmi tous les alignements calculés à l'étape précédente.

Comme y admet au plus $|y|$ conjugués distincts, il y a au plus $|y|$ alignements à calculer à l'étape 1 et, comme chaque conjugué de y est de longueur $|y|$, cette procédure prend, au total, un temps $O\Big(|x|\,|y|^2\Big)$.

L'algorithme de Maes fait mieux : il calcule un C-alignement π-optimal de (x, y) en temps $O(|x|\,|y| \log |y|)$ [80]. De plus, pour des fonctions de pondération particulières, il est possible de faire encore mieux.

• Supposons que π soit la métrique discrète sur Σ, c'est-à-dire que $d_{\mathrm{E}}^{\pi} = d_{\mathrm{L}}$. Alors, le π-score d'un C-alignement π-optimal de (x, y) est égal à

$$e := \min\Big\{d_{\mathrm{L}}(x, y') : y' \overset{\mathrm{C}}{\equiv} y\Big\}$$

et un tel C-alignement est calculable en temps $O(e \cdot |y|)$ [68].

• Supposons qu'il existe deux constantes entières M, $S \geq 0$ telles que :

$$\forall \alpha, \beta \in \Sigma \cup \{\varepsilon\} \qquad \pi(\alpha, \beta) = \begin{cases} -M & \text{si } \alpha = \beta \\ +S & \text{sinon} \end{cases}.$$

Alors, on peut calculer un C-alignement π-optimal de (x, y) en temps quadratique $O(|x|\,|y|)$ [102].

3.3.1.2 Élimination des problèmes d'orientation

Pour calculer un U-alignement (resp. un UC-alignement) π-optimal de (x, y), il suffit de calculer
- un alignement (resp. un C-alignement) π-optimal de (x, y) et
- un alignement (resp. un C-alignement) π-optimal de $(x, \tilde{y})$.

Ensuite, parmi les deux alignements calculés, on en retourne un de π-score minimum.

On a ainsi ramené le problème de l'U-alignement (resp. de l'UC-alignement) π-optimal au problème de l'alignement (resp. du C-alignement) π-optimal. Il en résulte que l'on peut calculer un U-alignement (resp. un UC-alignement) π-optimal de (x, y) en temps $O(|x|\,|y|) + O(|x|\,|\tilde{y}|) = O(|x|\,|y|)$ (resp. $O(|x|\,|y| \log |y|)$ en se fondant sur l'algorithme de Maes [80]).

3.3.1.3 Sous-séquences circulaires et/ou non orientées communes à deux mots

Dans la section 1.5.3 p. 23, nous avons introduit une fonction de pondération λ vérifiant la propriété suivante : calculer (le λ-score d')un alignement λ-optimal de (x, y) revient à calculer (la longueur d')une plus longue sous-séquence commune de $\{x, y\}$. Cette propriété montre que le problème de l'alignement de deux mots généralise le problème de la plus longue sous-séquence commune à deux mots. Elle s'étend naturellement aux problèmes étudiés dans ce chapitre.

Soit A un U-alignement (resp. C-alignement, resp. UC-alignement) λ-optimal de (x, y).

• On vérifie facilement que le λ-score de A est égal à $|x|+|y|-2\times\text{lcus}(\{x, y\})$ (resp. $|x|+|y|-2\times\text{lccs}(\{x, y\})$, resp. $|x| + |y| - 2 \times \text{lcucs}(\{x, y\})$).

• De plus, soit y' le mot tel que A soit un alignement de (x, y'). Alors, A est un alignement λ-optimal de (x, y'). Par suite, on peut extraire de A en temps linéaire une sous-séquence commune s de $\{x, y'\}$ de longueur maximale $\text{lcs}(\{x, y'\})$ (voir section 1.5.3). On vérifie facilement que s est une U-sous-séquence (resp. C-sous-séquence, resp. UC-sous-séquence) commune de $\{x, y\}$ de longueur maximale.

3.3.2 Le problème de la plus courte super-séquence commune et sa variante pour les mots circulaires

Le problème de la plus longue sous-séquence commune (voir section 1.4) admet pour problème "dual" le problème de la plus courte super-séquence commune. Ce dernier problème a fait l'objet de nombreux travaux (voir section 3.3.2.1 ci-dessous). Sa variante pour les mots circulaires a été également abordée dans la littérature (voir section 3.3.2.2).

3.3.2.1 Le problème de la plus courte super-séquence commune dans sa forme classique

Soient un langage fini X et un mot s. On dit que s est une *super-séquence commune* de X lorsque tout mot appartenant à X est une sous-séquence de s. On nomme Shortest Common Supersequence (SCS) le problème d'optimisation suivant : "*Étant donné un langage fini X, trouver une super-séquence commune de X de longueur minimale*".

Algorithme exact. Tout comme LCS, SCS admet un algorithme exact, polynomial pour $\#X$ fixé [60]. En effet, il suffit de considérer l'algorithme de la section 2.4.1 correspondant à une fonction de pondération Π bien choisie. On prend en l'occurrence pour Π la fonction qui, à chaque alignement-colonne C, associe :

- 1 s'il n'apparaît pas dans C deux lettres distinctes,
- $+\infty$ sinon.

Autrement dit,

$$\Pi\left(\begin{bmatrix}\alpha_1\\ \alpha_2\\ \vdots\\ \alpha_m\end{bmatrix}\right) = \begin{cases}1 & \text{si } \#\{\alpha_1,\alpha_2,\ldots,\alpha_m\} \leq 2\\ +\infty & \text{sinon}\end{cases}$$

pour tout entier $m \geq 1$ et tous mots α_1, α_2, …, α_m vides ou réduits à une lettre vérifiant $\alpha_1\alpha_2\ldots\alpha_m \neq \varepsilon$. En toute rigueur, nous n'avons pas autorisé la fonction Π à prendre la valeur $+\infty$ dans la section 2.4.1. Néanmoins, on peut facilement se rendre compte que, même dans ce cas, l'algorithme décrit dans la section 2.4.1.2 reste encore correct et de même complexité. De toute façon, le lecteur pointilleux pourra toujours considérer que, dans la suite, $\Pi(C) = m+1$ pour tout alignement-colonne C à m entrées dans lequel apparaît deux lettres distinctes.

Supposons donnés m mots x_1, x_2, …, x_m. À l'aide de l'algorithme de la section 2.4.1, on calcule, en temps polynomial pour m fixé, un alignement Π-optimal $A = \left[\alpha_{i,j}\right]_{(i,j)\in[1,m]\times[1,n]}$ de $(x_1, x_2, \ldots, x_m)$. Remarquons que chacune des colonnes d'un alignement Π-optimal est de Π-score 1. Le nombre n de colonnes de A est la longueur d'une plus courte super-séquence commune de $\{x_1, x_2, \ldots, x_m\}$. De plus, pour chaque $j \in [1, n]$, notons a_j la seule lettre apparaissant dans la j-ème colonne de A : on a $\{\alpha_{1,j}, \alpha_{2,j}, \ldots, \alpha_{m,j}\} \subseteq \{a_j, \varepsilon\}$. Alors, $a_1a_2\ldots a_n$ est une plus courte super-séquence commune de $\{x_1, x_2, \ldots, x_m\}$.

Complexité. La complexité du problème de décision SCS_D associé à SCS a été exhaustivement étudiée. Les résultats obtenus sont similaires à ceux concernant LCS_D.

Lorsque le nombre de mots pris en entrée est non borné, SCS_D est NP-complet [81]. De plus, même si l'on restreint SCS aux langages X vérifiant $\sigma(X) = 2$, SCS_D reste NP-difficile [98]. D'autre part, SCS_D reste également NP-difficile si l'on restreint SCS aux langages X vérifiant $|x| = 2$ pour tout $x \in X$ [111].

Par ailleurs, SCS_D est W[1]-difficile pour le paramètre $\#X$ (c'est-à-dire le nombre de mots pris en entrée) et ce résultat reste vrai même si l'on restreint SCS aux langages X vérifiant $\sigma(X) = 2$ [97].

Approximabilité. On trouve dans [55] trois résultats concernant l'approximabilité de SCS.

1. Le problème SCS admet un algorithme d'approximation de borne $\sigma(X)$.
2. Si SCS admet un algorithme d'approximation de borne constante alors NP = P.
3. Il existe une constante $\delta > 0$ telle que : si SCS admet un algorithme d'approximation de borne $\log^{\delta}(\#X)$ alors on a
$$\mathrm{NP} \subseteq \mathrm{DTIME}\left(2^{\log^{O(1)} n}\right).$$

3.3.2.2 Le problème de la plus courte super-séquence commune pour les mots circulaires

On nomme Shortest Common Cyclic Supersequence (SCCS) le problème d'optimisation suivant : "*Étant donné un langage fini X, trouver un mot s de longueur minimale tel que tout mot appartenant à X soit une C-sous-séquence de s*". Ainsi, SCCS est à SCS ce que LCCS est à LCS. La complexité du problème de décision SCCS_D, associé à SCCS, a été étudiée par Middendorf [85]. Ce dernier a montré que SCCS_D dans sa forme générale est NP-complet. Il obtenu également les trois résultats suivants.

1. Lorsqu'on le restreint aux langages X tels que $|x| = 2$ pour tout $x \in X$, SCCS admet un algorithme polynomial exact, contrairement à SCS.
2. En revanche, SCCS_D reste NP-difficile, même si l'on restreint SCCS aux langages X tels que $|x| = 3$ pour tout $x \in X$.
3. Par ailleurs, si l'on restreint SCCS aux langages X pour lesquels on a $\sigma(X) = 2$ alors SCCS_D reste NP-difficile.

3.3.3 Notre contribution

Le but du chapitre présent est d'étudier la complexité des problèmes LCUS, LCCS et LCUCS, définis page 64, lorsque le nombre de mots pris en entrée est non borné.

Difficulté de la résolution exacte des problèmes LCUS, LCCS et LCUCS. Nous obtenons un certain nombre de résultats sur la complexité des problèmes de décision LCUS_D, LCCS_D et LCUCS_D. Ces résultats sont pour la plupart établis dans la section 3.4. Ils sont résumés dans le tableau ci-dessous :

$\#X$	$\sigma(X)$	k	LCUS_D / LCCS_D / LCUCS_D
borné	–	–	polynomial (prop. 3.3 p. 68)
–	param.	param.	F.P.T. (rem. 3.7 p. 68)
–	au plus 2	–	NP-complet (th. 3.1 p. 70)
param.	au plus 2	–	W[1]-difficile (th. 3.1 p. 70)
–	–	param.	W[1]-difficile (prop. 3.6 p. 80 et 3.7 p. 83)
param.	–	param.	cas non abordé ici
param.	param.	–	W[t]-difficile $\forall t \geq 1$ (th. 3.1 p. 70)

.

Pour toute instance (X, k) de LCUS_D (resp. LCCS_D, resp. LCUCS_D),
- $\#X$ est le nombre de mots pris en entrée,
- $\sigma(X)$ est le cardinal de l'alphabet d'entrée et
- k est la longueur minimum acceptable pour une plus longue sous-séquence commune de X.

Approximabilité des problèmes LCUS, LCCS et LCUCS. Dans la section 3.5, nous généralisons à LCUS, LCCS et LCUCS les deux bornes d'approximation connues pour LCS [55, 46]. Enfin, dans la section 3.6.2, nous montrons qu'il est difficile d'approximer LCUS, LCCS et LCUCS avec des bornes d'ordre de grandeur nettement inférieur. En effet, à l'aide de réductions, nous montrons que :

1. LCUS et LCUS sont au moins aussi difficiles à approximer que le problème de l'indépendant maximum pour les graphes dans la section 3.6.2.2 et, que
2. LCCS et LCUCS sont au moins aussi difficiles à approximer que le problème de l'indépendant maximum pour les hypergraphes 3-uniformes dans la section 3.6.2.3.

Ces réductions permettent d'appliquer à nos problèmes, les résultats de Håstad concernant le problème de l'indépendant maximum [47] (voir section 3.6.2.4).

3.4 Résolution exacte de LCUS, LCCS et LCUCS

Dans cette section, nous étudions la difficulté de la résolution exacte des problèmes LCUS, LCCS et LCUCS. Nous généralisons à ces trois problèmes plusieurs résultats importants déjà connus pour LCS.

Commençons par deux remarques et une proposition faciles avant de démontrer nos résultats principaux dans la section 3.4.2.

3.4.1 Résultats positifs

Dans toute cette section, X désigne un langage fini et non vide, et k est un entier positif.

Remarque 3.6 *Les problèmes* LCUS, LCCS *et* LCUCS *sont dans* NPO *(voir section 1.2.5.2 pour une définition). Par suite, leurs problèmes de décision associés,* LCUS_D, LCCS_D *et* LCUCS_D *sont dans* NP.

Détaillons un peu.
- Calculer la longueur d'une U-sous-séquence (resp. C-sous-séquence, resp. UC-sous-séquence) commune de X *donnée* prend trivialement un temps polynomial : la propriété ($NPO1$) est vérifiée.
- Toute U-sous-séquence (resp. C-sous-séquence, resp. UC-sous-séquence) commune de X est de longueur au plus $\ell(X) = \min_{x \in X} |x|$, donc sa taille est inférieure à celle de X : la propriété ($NPO2$) est vérifiée.

– Décider si un mot donné est une U-sous-séquence (resp. C-sous-séquence, resp. UC-sous-séquence) commune de X prend un temps polynomial : la propriété ($NPO3$) est vérifiée.

Remarque 3.7 *Les problèmes* LCS_D, LCUS_D, LCCS_D *et* LCUCS_D *sont F.P.T. pour le paramètre* $(\sigma(X), k)$.

En effet, cherchons à résoudre LCS_D (resp. LCUS, resp. LCCS, resp. LCUCS) sur l'instance (X, k).

1. On calcule le plus petit alphabet Σ tel que $X \subseteq \Sigma^\star$.
2. Pour chaque $s \in \Sigma^k$, on teste si s est une sous-séquence (resp. U-sous-séquence, resp. C-sous-séquence, resp. UC-sous-séquence) commune de X.

L'étape 1 prend trivialement un temps polynomial. Comme Σ est de cardinal $\sigma(X)$, il y a $(\sigma(X))^k$ mots à énumérer à l'étape 2 et chacun des $(\sigma(X))^k$ tests à effectuer prend un temps polynomial. Ceci justifie la remarque 3.7 p. 68.

Par ailleurs, nos trois problèmes sont, tout comme LCS, résolubles en temps polynomial lorsque le nombre de mots pris en entrée est fixé.

Proposition 3.3
Chacun des problèmes LCUS, LCCS *et* LCUCS *admet un algorithme exact, polynomial lorsque le nombre* $\#X$ *de mots pris en entrée est fixé.*

Preuve. Soit un langage fini et non vide X. On construit un algorithme exact pour LCUS (resp. LCCS, resp. LCUCS) de la manière suivante.

1. Posant $m := \#X$, on numérote les éléments de X : $X = \{x_1, x_2, \ldots, x_m\}$.
2. Pour chaque m-uplet $(y_1, y_2, \ldots, y_m)$ de mots vérifiant $y_i \in \{x_i, \tilde{x}_i\}$ (resp. $y_i \overset{\text{C}}{\equiv} x_i$, resp. $y_i \overset{\text{UC}}{\equiv} x_i$) pour tout $i \in [1, m]$, on calcule une plus longue sous-séquence commune de $\{y_1, y_2, \ldots, y_m\}$.
3. Parmi les sous-séquences calculées à l'étape 2, on en retourne une de longueur maximale.

On vérifie facilement que cet algorithme est correct (voir remarque 3.3 p. 63). Reste à borner sa complexité. L'étape 2 est celle qui prend le plus de temps. Tout d'abord, il y a au plus 2^m (resp. $\prod_{i=1}^m |x_i|$, resp. $2^m \prod_{i=1}^m |x_i|$) m-uplets $(y_1, y_2, \ldots, y_m)$ à énumérer. Ensuite, en utilisant l'algorithme exact standard de programmation dynamique pour LCS, on peut calculer une plus longue sous-séquence commune de chaque ensemble $\{y_1, y_2, \ldots, y_m\}$ en temps $O(m \prod_{i=1}^m |x_i|)$ [53]. Au total, la complexité de notre algorithme est donc $O(m2^m \prod_{i=1}^m |x_i|)$ (resp. $O\left(m \prod_{i=1}^m |x_i|^2\right)$, resp. $O\left(m2^m \prod_{i=1}^m |x_i|^2\right)$) : elle est bien polynomiale à m fixé. **Q.E.D.**

3.4.2 Résultats négatifs

Cette section est dévolue à la preuve du théorème 3.1 p. 70. Ce théorème généralise à LCUS_D, LCCS_D et LCUCS_D les principaux résultats connus sur la complexité de LCS_D (voir section 1.4). Essentiellement, nous montrons que LCUS_D, LCCS_D et LCUCS_D sont : NP-complets, et W[1]-difficiles pour le paramètre "nombre de mots pris en entrée" (c'est-à-dire le cardinal de X). De plus, nos preuves garantissent que ces résultats restent vrais même si on restreint LCUS, LCCS et LCUCS aux instances X qui sont des langages sur des alphabets à 2 lettres (c'est-à-dire telles que $\sigma(X) \leq 2$). Notons que ces résultats sont tout à fait intuitifs. En effet, il est naturel de penser que LCUS, LCCS et LCUCS sont au moins aussi difficiles que LCS. Néanmoins, la preuve du théorème 3.1 p. 70, qui consiste à exhiber des réductions depuis LCS_D, n'est pas triviale.

On commence par établir le lemme de synchronisation suivant :

Lemme 3.3 (Synchronisation) *Soient deux mots* x *et* y *et soient quatre entiers* m, n, p, q *vérifiant :* $m > 0$, $n > 0$, $p > |x|$ *et* $q > |x|$.

Alors, pour que les mots $\mathtt{0}^m x \mathtt{1}^n$ *et* $\mathtt{0}^p y \mathtt{1}^q$ *soient égaux, il suffit que* $\mathtt{0}^m x \mathtt{1}^n$ *soit conjugué avec* $\mathtt{0}^p y \mathtt{1}^q$ *ou avec l'image miroir de* $\mathtt{0}^p y \mathtt{1}^q$.

Preuve. Posons $u := \mathtt{0}^m x \mathtt{1}^n$ et $v := \mathtt{0}^p y \mathtt{1}^q$. Tout conjugué de u *distinct* de u est de l'une des trois formes suivantes :

(a). $\mathtt{0}^j x \mathtt{1}^n \mathtt{0}^{m-j}$ avec $j \in [1, m-1]$,

(b). $\mathtt{1}^j \mathtt{0}^m x \mathtt{1}^{n-j}$ avec $j \in [1, n-1]$,

(c). $x_2 \mathtt{1}^n \mathtt{0}^m x_1$ où x_1 et x_2 sont deux mots tels que $x = x_1 x_2$.

• Supposons $u \overset{\text{C}}{\equiv} v$ et montrons que $u = v$.

La dernière lettre du mot v est un $\mathtt{1}$ donc v n'est pas de la forme (a). De même, la première lettre de v est un $\mathtt{0}$ donc v n'est pas non plus de de la forme (b). Enfin, x est de longueur *strictement* inférieure à p donc v admet $\mathtt{0}^{|x|+1}$ comme préfixe, ce qui n'est pas le cas des mots de la forme (c). Ainsi, la seule possibilité est que $u = v$.

• Supposons (absurde) $u \overset{\text{C}}{\equiv} \tilde{v}$ et $u \neq v$.

Comme $\tilde{v} = \mathtt{1}^q \tilde{y} \mathtt{0}^p$ commence par un $\mathtt{1}$, $\tilde{v}$ n'est pas de la forme (a). De même, comme $\tilde{v}$ finit par un $\mathtt{0}$, $\tilde{v}$ n'est pas de la forme (b). On en déduit que $\tilde{v}$ est de la forme (c). Ainsi, il existe deux mots x_1, x_2 vérifiant $x = x_1 x_2$ et $\mathtt{1}^q \tilde{y} \mathtt{0}^p = x_2 \mathtt{1}^n \mathtt{0}^m x_1$. Comme on a supposé $q \geq |x| \geq |x_2|$ et $p \geq |x| \geq |x_1|$, on a nécessairement $x_2 = \mathtt{1}^{|x_2|}$ et $x_1 = \mathtt{0}^{|x_1|}$. Connaissant maintenant la forme de x_1 et celle de x_2, on peut écrire : $u = \mathtt{0}^m x_1 x_2 \mathtt{1}^n = \mathtt{0}^{|x_1|+m} \mathtt{1}^{|x_2|+n}$ et $\tilde{v} = x_2 \mathtt{1}^n \mathtt{0}^m x_1 = \mathtt{1}^{|x_2|+n} \mathtt{0}^{|x_1|+m}$. On voit alors que $u = v$: contradiction.

Q.E.D.

Introduisons maintenant notre gadget :

Définition 3.8 (Gadget)
Pour tout langage fini X, on note :

$$X^{\text{UC}} := \left\{ \mathtt{0}^{2M(X)+1} x \mathtt{1}^{2M(X)+1} : x \in X \right\} \quad \textit{où} \quad M(X) = \max_{x \in X} |x| \,.$$

Le lemme suivant décrit le fonctionnement de notre gadget :

Lemme 3.4 *Pour tout langage non vide X, les quatre quantités* $\text{lcs}(X^{\text{UC}})$, $\text{lcus}(X^{\text{UC}})$, $\text{lccs}(X^{\text{UC}})$ *et* $\text{lcucs}(X^{\text{UC}})$ *sont toutes égales à*

$$\text{lcs}(X) + 4M(X) + 2 \,.$$

Preuve. En appliquant plusieurs fois le lemme 2.1 p. 40 on obtient

$$\begin{aligned} \text{lcs}(X^{\text{UC}}) &= \text{lcs}\big(\mathtt{0}^{2M(X)+1} X \mathtt{1}^{2M(X)+1}\big) = \text{lcs}\big(X \mathtt{1}^{2M(X)+1}\big) + 2M(X) + 1 \\ &= \text{lcs}(X) + 4M(X) + 2 \,. \end{aligned} \tag{3.1}$$

Il ne reste donc plus qu'à montrer les égalités :

$$\text{lcucs}(X^{\text{UC}}) = \text{lccs}(X^{\text{UC}}) = \text{lcus}(X^{\text{UC}}) = \text{lcs}(X^{\text{UC}})$$

ou, plus simplement, l'inégalité :

$$\text{lcucs}(X^{\text{UC}}) \leq \text{lcs}(X^{\text{UC}}) \tag{3.2}$$

d'après la remarque 3.5 p. 63.

Soit t une UC-sous-séquence commune de X^{UC} de longueur maximale $\text{lcucs}(X^{\text{UC}})$. Pour chaque $x \in X$, il existe une sous-séquence t_x de $\mathtt{0}^{2M(X)+1} x \mathtt{1}^{2M(X)+1}$ vérifiant $t \overset{\text{UC}}{\equiv} t_x$. Remarquons alors que la relation $t \overset{\text{UC}}{\equiv} t_x$ force $|t_x| = |t| = \text{lcucs}(X^{\text{UC}})$ pour tout $x \in X$. Par suite, si l'on parvient à montrer que les t_x ($x \in X$) sont tous égaux, alors on obtiendra une sous-séquence commune de X^{UC} de longueur $\text{lcucs}(X^{\text{UC}})$. Ainsi, on aura montré (3.2) et on aura terminé la preuve du lemme 3.4.

Prenons x, $y \in X$ quelconques et montrons que $t_x = t_y$.

Comme t_x est une sous-séquence de $\mathtt{0}^{2M(X)+1} x \mathtt{1}^{2M(X)+1}$, on peut écrire t_x sous la forme :

$$t_x = \mathtt{0}^m s_x \mathtt{1}^n \text{ avec } s_x \text{ sous-séquence de } x \text{ et } m,\, n \in [0, 2M(X)+1].$$

De même, on peut écrire :

$$t_y = \mathtt{0}^p s_y \mathtt{1}^q \text{ avec } s_y \text{ sous-séquence de } y \text{ et } p,\, q \in [0, 2M(X)+1].$$

Par ailleurs, on a $t \overset{\text{UC}}{\equiv} t_x$ et $t \overset{\text{UC}}{\equiv} t_y$ donc, par la proposition 3.2, on a également $t_x \overset{\text{UC}}{\equiv} t_y$. Autrement dit, $\mathtt{0}^m s_x \mathtt{1}^n$ est conjugué avec $\mathtt{0}^p s_y \mathtt{1}^q$ ou avec $\mathtt{0}^p s_y \mathtt{1}^q$.

Pour prouver que $t_x = t_y$, on va appliquer le lemme 3.3 p. 68 en substituant, dans son énoncé, s_x à x et s_y à y. Mais, avant de pouvoir appliquer ce lemme de synchronisation, il reste à vérifier que ses hypothèses sont satisfaites. On doit donc montrer les quatre inégalités $m > 0$, $n > 0$, $p > |s_x|$ et $q > |s_y|$. En fait, on va montrer légèrement mieux, à savoir que les quatre entiers m, n, p, q sont tous strictement supérieurs à $M(X)$. Comme on a $M(X) \geq |x| \geq |s_x|$ et $M(X) \geq |y| \geq |s_y|$, on aura fini.

Supposons (absurde) $p \leq M(X)$. On a alors

$$\begin{array}{rcccccc} \text{lcucs}(X^{\text{UC}}) = |t| = |t_y| & = & p & + & |s_y| & + & q \\ & \leq & M(X) & + & |y| & + & 2M(X)+1 \\ & \leq & M(X) & + & M(X) & + & 2M(X)+1 \end{array},$$

d'où

$$\text{lcucs}(X^{\text{UC}}) < 4M(X) + 2 \leq \text{lcs}(X) + 4M(X) + 2 = \text{lcs}(X^{\text{UC}}),$$

la dernière égalité provenant de (3.1). On obtient ainsi $\text{lcucs}(X^{\text{UC}}) < \text{lcs}(X^{\text{UC}})$, ce qui ne se peut pas (remarque 3.5 p. 63). Notre raisonnement par l'absurde garantit alors $p > M(X)$. En raisonnant de la même manière, on démontre que m, n et q sont strictement supérieurs à $M(X)$. **Q.E.D.**

En fait, on pourrait montrer que, pour tout mot t, on a équivalence entre :

1. t est une plus longue UC-sous-séquence commune de X^{UC} et
2. il existe une plus longue sous-séquence commune s de X telle que $t \overset{\text{UC}}{\equiv} \mathtt{0}^{2M(X)+1} s \mathtt{1}^{2M(X)+1}$.

Théorème 3.1

- *Quel que soit l'entier $t \geq 1$, les problèmes* LCUS_D, LCCS_D *et* LCUCS_D *sont* W[t]*-difficiles pour le paramètre* $(\#X, \sigma(X))$.
- *Même si on les restreint aux instances (X, k) vérifiant $\sigma(X) = 2$, les problèmes* LCUS_D, LCCS_D *et* LCUCS_D *restent :*
  - NP*-complets et*
  - W[1]*-difficiles pour le paramètre* $\#X$.

Preuve. Par la remarque 3.6 p. 67, nos trois problèmes sont dans NP. Montrons maintenant les divers résultats d'intraitabilité.

Notons tout d'abord que restreindre le problème LCS aux instances X pour lesquelles on a $\sigma(X) \geq 2$ ne change pas sa difficulté. En effet, si un langage fini et non vide X vérifie $\sigma(X) \leq 1$, alors X admet une unique sous-séquence commune de longueur $\text{lcs}(X)$ qui se calcule trivialement : c'est le mot le plus court appartenant à X.

Considérons l'application qui, à toute instance (X, k) de LCS_D avec $\sigma(X) \geq 2$, associe le couple $\big(X^{\text{UC}}, k + 4M(X) + 2\big)$. Cette application est trivialement calculable en temps polynomial. D'après le lemme 3.4 p. 69, elle réalise ainsi une réduction de Karp de LCS_D à l'un quelconque des trois problèmes LCUS_D, LCCS_D ou LCUCS_D. De plus, elle conserve les paramètres qui nous intéressent : $\#X^{\text{UC}} = \#X$ et on peut toujours supposer que $\sigma(X^{\text{UC}}) = \sigma(X)$. En effet, comme on a $\sigma(X) \geq 2$, on peut facilement modifier l'instance (X, k) de LCS_D de manière à ce que chacune des lettres $\mathtt{0}$ et $\mathtt{1}$ apparaissent dans au moins un mot de X : il suffit de renommer deux lettres quelconques apparaissant dans X.

Ceci permet de généraliser à LCUS_D, LCCS_D et LCUCS_D les résultats suivants concernant LCS_D (voir aussi la section 1.4.3) :

- NP-complétude même lorsqu'on se restreint aux instances (X, k) vérifiant $\sigma(X) = 2$ [81] ;
- W[1]-difficulté pour le paramètre $\#X$ même lorsqu'on se restreint aux instances (X, k) vérifiant $\sigma(X) = 2$ [97] ;
- W[t]-difficulté pour le paramètre $(\#X, \sigma(X))$ quel que soit l'entier $t \geq 1$ [13].

Q.E.D.

3.5 Algorithmes d'approximation pour LCS, LCUS, LCCS et LCUCS

Les propositions 3.4 et 3.5 suivantes généralisent à LCUS, LCCS et LCUCS les deux résultats positifs connus sur l'approximabilité de LCS [55, 46]. Nous démontrons que chacun des quatre problèmes LCS, LCUS, LCCS et LCUCS admet les deux bornes d'approximation suivantes : $\sigma(X)$ dans la proposition 3.4 p. 71, et $\ell(X)/\log \ell(X)$ dans la proposition 3.5 p. 73. Rappelons que $\ell(X) = \min_{x\in X} |x|$. Même si ces bornes semblent au premier abord ridiculement élevées, le corollaire 3.2 p. 83 garantit qu'elles sont quasi-optimales.

Proposition 3.4
Chacun des quatre problèmes LCS, LCUS, LCCS *et* LCUCS *admet un algorithme d'approximation de borne* $\sigma(X)$ *(le cardinal de l'alphabet d'entrée).*

Preuve. Considérons l'algorithme suivant, baptisé Long-Run par Jiang et Li [114] : pour chaque langage fini et non vide X pris en entrée, Long-Run calcule une sous-séquence commune *unaire* u_X de X de longueur maximale.

Cet algorithme est implantable en temps polynomial : parmi toutes les lettres apparaissant dans au moins un mot de X, on choisit une lettre a maximisant $\ell_a(X) := \min_{x\in X} |x|_a$ et on retourne $u_X := a^{\ell_a(X)}$.

Par ailleurs, quel que soit X, u_X est à la fois une sous-séquence, une U-sous-séquence, une C-sous-séquence et une UC-sous-séquence commune de X (remarque 3.4 p. 63). Ainsi, Long-Run peut-être vu comme un algorithme d'approximation pour LCS, LCUS, LCCS et LCUCS. Reste à montrer qu'il admet bien les bornes annoncées c'est-à-dire que u_X est de longueur supérieure à $\mathrm{lcs}(X) \,/\, \sigma(X)$, $\mathrm{lcus}(X) \,/\, \sigma(X)$, $\mathrm{lccs}(X) \,/\, \sigma(X)$ et à $\mathrm{lcucs}(X) \,/\, \sigma(X)$.

En fait, il suffit de vérifier l'inégalité $|u_X| \geq \mathrm{lcucs}(X) \,/\, \sigma(X)$ (remarque 3.5 p. 63). La preuve que nous en donnons est fondée sur la remarque suivante.

Remarque 3.8 *Soit s un mot quelconque et soit a^* une lettre dont le nombre d'occurrences dans s est maximal :*

$$|s|_{a^*} = \max_{a\in\Sigma} |s|_a \text{ avec } \Sigma := \{s[i] : i \in [1, |s|]\}.$$

Alors, le nombre d'occurrences de a^* dans s est au moins égal au quotient de la longueur de s par le nombre $\sigma(\{s\})$ de lettres distinctes apparaissant dans s :

$$|s|_{a^*} \geq \frac{1}{\#\Sigma} \sum_{a\in\Sigma} |s|_a = \frac{1}{\#\Sigma} |s| = \frac{1}{\sigma(\{s\})} |s| \ .$$

Prenons pour s une UC-sous-séquence commune de X de longueur maximale $\mathrm{lcucs}(X)$. Soit u l'unique mot unaire de longueur $|s|_{a^*}$ sur l'alphabet $\{a^*\}$: $u = a^*a^* \dots a^*$ avec $|u| = |s|_{a^*}$. Toute lettre apparaissant dans s apparaît aussi dans les mots de X donc on a $\sigma(\{s\}) \leq \sigma(X)$ et par suite, il vient

$$|u| = |s|_{a^*} \geq \frac{|s|}{\sigma(\{s\})} = \frac{\mathrm{lcucs}(X)}{\sigma(\{s\})} \geq \frac{\mathrm{lcucs}(X)}{\sigma(X)} \ .$$

Or, u est une sous-séquence de s donc une sous-séquence commune unaire de X. Par maximalité de $|u_X|$, on a donc :

$$|u_X| \geq |u| \geq \frac{\mathrm{lcucs}(X)}{\sigma(X)} \ .$$

Q.E.D.

Supposons donné un langage fini et non vide X. Considérons l'algorithme trivial, retournant :
– une lettre commune à tous les des mots de X, s'il en existe et
– le mot vide sinon.

On obtient ainsi un algorithme d'approximation pour LCS, LCUS, LCCS et LCUCS de borne $\ell(X)$. La méthode d'*approximation par partitionnement* de Halldòrsson [46], permet d'obtenir la borne légèrement meilleure de $\ell(X) \,/\, \log \ell(X)$ (proposition 3.5 p. 73).

Pour démontrer cette proposition, nous avons besoin de montrer l'existence d'un certain type d'algorithmes *exponentiels* exacts pour nos problèmes (lemme 3.6 p. 72). Les algorithmes d'approximation *polynomiaux* décrits dans la démonstration de la proposition 3.5 p. 73 appliquent ces algorithmes exacts sur des entrées de tailles *logarithmiques*.

Avant de pouvoir démontrer le lemme 3.6 p. 72 comme annoncé, nous avons besoin d'un résultat préliminaire, relevant essentiellement du folklore.

Lemme 3.5 *Soit un entier $n \geq 0$.*

(i). L'ensemble des sous-séquences distinctes admises par un mot de longueur n est de cardinal au plus de 2^n.

(ii). On peut construire cet ensemble en temps $O(n2^n)$.

Preuve. Soit w un mot de longueur n.

(i). Considérons l'application $\mathrm{sseq}(w, \cdot\,)$ qui, à toute partie $I \subseteq [1, n]$, associe $\mathrm{sseq}(w, I) : \mathrm{sseq}(w, \cdot\,)$ induit une surjection de l'ensemble des parties de $[1, n]$ sur l'ensemble des sous-séquences de w. On en déduit que w admet au plus 2^n sous-séquences distinctes, c'est-à-dire (i).

(ii). On construit facilement le langage $\{\mathtt{0}, \mathtt{1}\}^n$ en un temps $O(n2^n)$ (linéaire en sa taille). Ensuite, pour chaque mot $u \in \{\mathtt{0}, \mathtt{1}\}^n$, on construit en temps $O(n)$ la sous-séquence $\mathrm{sseq}(w, \{i \in [1, n] : u[i] = \mathtt{1}\})$ de w. On obtient ainsi en temps $O(n2^n)$ une liste de longueur 2^n dans laquelle chaque sous-séquence de w apparaît au moins une fois. On élimine les répétitions en temps linéaire à l'aide d'un tri par ordre lexicographique.

Q.E.D.

On peut vérifier que la borne donnée dans le point (i) est atteinte pour les *permutations* de longueur n.

Lemme 3.6 *Il existe un algorithme exact pour* LCS *(resp.* LCUS*, resp.* LCCS*, resp.* LCUCS*) de complexité $O\big(2^{\ell(X)} \,|X|^c\big)$ où c est une constante positive.*

Preuve. On construit un algorithme exact pour LCS (resp. LCUS, resp. LCCS, resp. LCUCS) de la manière suivante. Supposons donné un langage fini et non vide X.

1. On choisit un mot $x_* \in X$ de longueur minimum $\ell(X)$.
2. On construit l'ensemble S des sous-séquences de x_*.
3. On construit le sous-ensemble S' des mots de S qui sont des sous-séquences (resp. U-sous-séquences, resp. C-sous-séquences, resp. UC-sous-séquences) communes de X.
4. On retourne un mot de longueur maximale appartenant à S'.

Clairement notre algorithme est correct. Reste à borner sa complexité.

L'étape 1 prend un temps $O(\#X)$ et, par le point (ii) du lemme 3.5 p. 72, l'étape 2 prend un temps $O\big(\ell(X)2^{\ell(X)}\big)$. Par ailleurs, tester si un mot donné est une sous-séquence (resp. U-sous-séquence, resp. C-sous-séquence, resp. UC-sous-séquence) commune de X prend un temps polynomial. Au cours de l'étape 3, on réalise un test de ce genre pour chaque mot de S donc cette étape prend un temps $O\Big((\#S)\,|X|^{O(1)}\Big)$. Enfin, l'étape 4 prend un temps $O(\#S')$.

On a $\#S' \leq \#S \leq 2^{\ell(X)}$ par le point (i) du lemme 3.5 p. 72. Donc, au total, le temps de calcul de notre algorithme est

$$O\Big(\#X + \ell(X)2^{\ell(X)} + (\#S)\,|X|^{O(1)} + \#S'\Big) \subseteq O\Big(2^{\ell(X)}\,|X|^{O(1)}\Big)\,.$$

Q.E.D.

Notons que le lemme 3.6 garantit que LCS_D, LCUS_D, LCCS_D et LCUCS_D sont F.P.T. pour le paramètre "longueur du plus court mot pris en entrée" (c'est-à-dire $\ell(X)$).

Avant d'énoncer la proposition 3.5, rappelons que le rapport $\ell \,/\, \log \ell$ n'est pas défini lorsque $\ell \in \{0, 1\}$.

Proposition 3.5
Lorsqu'on les restreint aux instances X pour lesquelles $\ell(X)$ est au moins 2, les quatre problèmes LCS, LCUS, LCCS *et* LCUCS *admettent chacun un algorithme d'approximation de borne $\ell(X) \,/\, \log \ell(X)$.*

Preuve. Supposons donné un langage fini et non vide X avec $\ell(X) \geq 2$, et considérons l'algorithme suivant.

1. On prend un mot $x_* \in X$ de longueur minimale $\ell(X)$ et on le factorise sous la forme $x_* = x_1 x_2 \dots x_p$ où
   - $p := \left\lfloor \dfrac{\ell(X)}{1 + \lfloor \log \ell(X) \rfloor} \right\rfloor$ est un entier inférieur à $\ell(X) \,/\, \log \ell(X)$, et où
   - pour tout $i \in [1, p]$, $x_i := x_*(\lfloor (i-1) \cdot \ell(X) \,/\, p \rfloor \,;\, \lfloor i \cdot \ell(X) \,/\, p \rfloor]$ est un mot de longueur $O(\log \ell(X))$.
2. Pour chaque $i \in [1, p]$, on calcule une sous-séquence (resp. une U-sous-séquence, resp. une C-sous-séquence, resp. une UC-sous-séquence) commune s_i^* de $X \setminus \{x_*\} \cup \{x_i\}$ de longueur maximale.
3. On calcule un indice $j \in [1, p]$ maximisant $\left|s_j^*\right|$ et on retourne s_j^*.

• *Montrons que notre algorithme est polynomial.*
Notons que $1 + \lfloor \log \ell(X) \rfloor$ est le nombre de chiffres apparaissant dans l'écriture en base 2 du nombre $\ell(X)$ et que, pour tous entiers $m \geq 0$ et $n \geq 1$, $\lfloor m \,/\, n \rfloor$ est le quotient de la division euclidienne de m par n. Ainsi, l'entier p et les x_i sont aisément calculables en machine : l'étape 1 prend un temps polynomial.
La seule difficulté se situe, par conséquent, au niveau de l'étape 2. Pour chaque $i \in [1, p]$, le lemme 3.6 p. 72 garantit que s_i^* est calculable en temps $O\Big(2^{|x_i|}\,|X|^{O(1)}\Big)$. Or, le facteur exponentiel $2^{|x_i|}$ est polynomial en $\ell(X)$ puisque $|x_i| = O(\log \ell(X))$. Il en résulte que s_i^* est calculable en un temps polynomial en la taille de X. Notre algorithme est bien polynomial.

• *Montrons que notre algorithme est bien un algorithme d'approximation pour* LCS *(resp.* LCUS*, resp.* LCCS*, resp.* LCUCS*).*
Le mot s_j^* retourné par notre algorithme est une sous-séquence (resp. U-sous-séquence, resp. C-sous-séquence, resp. UC-sous-séquence) commune de $X \setminus \{x_*\} \cup \{x_j\}$ donc à plus forte raison de X : c'est bien une solution à notre problème.

• *Montrons que notre algorithme admet bien pour borne $\ell(X) \,/\, \log \ell(X)$.*
Il existe une sous-séquence s^* de x_* qui est également une sous-séquence (resp. une U-sous-séquence, resp. une C-sous-séquence, resp. une UC-sous-séquence) commune de X de longueur maximale : $|s^*| = \mathrm{lcs}(X)$ (resp. $|s^*| = \mathrm{lcus}(X)$, resp. $|s^*| = \mathrm{lccs}(X)$, resp. $|s^*| = \mathrm{lcucs}(X)$). Factorisons s^* sous la forme

$$s^* = s_1 s_2 \dots s_p \text{ où } s_i \text{ est une sous-séquence de } x_i \text{ pour tout } i \in [1, p].$$

Pour tout $i \in [1, p]$, s_i est une sous-séquence (resp. U-sous-séquence, resp. C-sous-séquence, resp. UC-sous-séquence) commune de $X \setminus \{x_*\} \cup \{x_i\}$ donc, on a

$$\forall i \in [1, p] \qquad |s_i^*| \geq |s_i| \ .$$

Par suite, la longueur du mot retourné par notre algorithme vérifie

$$\left|s_j^*\right| = \max_{i=1}^{p} |s_i^*| \geq \max_{i=1}^{p} |s_i| \geq \frac{1}{p} \sum_{i=1}^{p} |s_i| = \frac{1}{p}\,|s^*| \geq \frac{|s^*|}{\ell(X) \,/\, \log \ell(X)}\,,$$

d'où la borne annoncée.

Q.E.D.

3.6 Inapproximabilité de MISH, LCS, LCUS, LCCS et LCUCS

Le but de cette section est de montrer que les problèmes d'optimisation LCS et LCUS (resp. LCCS et LCUCS), introduits dans la définition 3.6 p. 64, sont au moins aussi difficiles à approximer que le problème de l'indépendant maximum pour les graphes (resp. les hypergraphes 3-uniformes). Rappelons que le problème de l'indépendant maximum pour les hypergraphes (MISH) ainsi que les diverses restrictions

de ce problème qui nous intéressent, sont introduits dans la section 1.3. On commence par établir dans la section 3.6.1 quelques résultats liminaires concernant le problème de l'indépendant maximum dans les hypergraphes uniformes. Ces résultats vont nous permettre de réduire plus facilement nos problèmes de plus longues sous-séquences communes à 2-MISH et à 3-MISH dans la section 3.6.2.

3.6.1 Difficulté du problème MISH

Dans toute cette section, on se donne une constante entière $r \geq 2$. On cherche à comparer la complexité du problème de l'indépendant maximum pour les hypergraphes r-uniformes (r-MISH) avec celle du problème de l'indépendant maximum pour les graphes (2-MISH). Nous montrons que, du point de vue de l'approximation et de la complexité paramétrique, r-MISH est au moins aussi difficile que 2-MISH (théorèmes 3.2 p. 76 et 3.3 p. 76). Plus précisément, nous prouvons dans la section 3.6.1.2 que

- le problème de décision r-MISH$_D$ associé à r-MISH est W[1]-difficile pour son paramètre "seuil d'acceptation" (théorème 3.2 p. 76) et que
- toute borne d'approximation pour r-MISH est valable également pour 2-MISH (théorème 3.3 p. 76).

Les deux principaux résultats de la section 3.6.1.1 suivante sont les lemmes 3.7 p. 74 et 3.9 p. 75. Ils ont pour fonction d'éliminer les "effets de bords" engendrés par plusieurs de nos gadgets. En effet, ces gadgets ne permettent pas de réduire, comme nous l'aurions voulu, les problèmes 2-MISH, 2-MISH$_D$, 3-MISH ou 3-MISH$_D$ dans leur généralité à un certain nombre de nos problèmes. Nous parvenons néanmoins à effectuer des réductions depuis de légères restrictions de 2-MISH, 2-MISH$_D$, 3-MISH et 3-MISH$_D$. Nous montrons dans la section 3.6.1.1 que ces restrictions sont aussi difficiles que les problèmes originaux.

3.6.1.1 Gestion des "effets de bord"

Les lemmes techniques 3.7 et 3.9 ci-dessous seront utilisés plusieurs fois dans ce mémoire.

Lemme 3.7 *Soit une constante entière $k_0 \geq 0$.*

Le problème de décision associé au problème 2-MISH *reste* W[1]*-difficile pour le paramètre k même si on le restreint aux instances (G, k) dont le seuil d'acceptation k est au moins k_0.*

Preuve. Supposons donnée une instance quelconque (G, k) de 2-MISH$_D$. On note $\mathcal{E}$ l'ensemble des arêtes de G. On peut supposer que $G = ([1, |G|], \mathcal{E})$.

Soit $I_0 := [|G| + 1, |G| + k_0]$: I_0 est un ensemble de cardinal k_0 ne contenant aucun sommet de G. On construit une nouvelle instance $(\hat{G}, \hat{k})$ de 2-MISH$_D$ où $\hat{G} := (V \cup I_0, \mathcal{E})$ et où $\hat{k} := k + k_0$. Informellement, $\hat{G}$ est obtenu à partir de G en lui ajoutant k_0 sommets isolés. Notons que l'on a $\hat{k} \geq k_0$.

Cette construction prend trivialement un temps polynomial. De plus, seuil d'acceptation $\hat{k}$ de l'instance réduite $(\hat{G}, \hat{k})$ n'est fonction que du seuil d'acceptation de l'instance initiale (G, k). Il reste à montrer que notre transformation est bien une M-réduction c'est-à-dire que : G admet un indépendant de cardinal k si et seulement si $\hat{G}$ admet un indépendant de cardinal $\hat{k}$.

Si G admet un indépendant I de cardinal k alors $\hat{G}$ admet pour indépendant $I \cup I_0$ qui est de cardinal $\hat{k}$. Réciproquement si $\hat{G}$ admet un indépendant $\hat{I}$ de cardinal $\hat{k}$ alors $I := \hat{I} \setminus I_0$ est un indépendant de G de cardinal au moins k donc G admet des indépendants de cardinal exactement k. **Q.E.D.**

Le lemme 3.8 ci-dessous n'a pour fonction que de faciliter la preuve du lemme 3.9 qui le suit. Rappelons que $\alpha(H)$ désigne le cardinal maximum pour un indépendant de H.

Lemme 3.8 *Soit une constante entière $k_0 \geq 0$.*

Alors, il existe un algorithme polynomial retournant, pour chaque hypergraphe H pris en entrée, un indépendant $J_H^{k_0}$ de H de cardinal $\min\{\alpha(H), k_0\}$.

Preuve. Supposons donné un hypergraphe H.

1. On énumère les $O\left(|H|^{k_0}\right)$ sous-ensembles de cardinal au plus k_0 de l'ensemble des sommets de H.
2. Parmi tous les sous-ensembles énumérés à l'étape précédente, on élimine ceux qui ne sont pas des indépendants de H.

3. Parmi les ensembles restants, on en retourne un de cardinal maximum.

Cet algorithme prend un temps polynomial à k_0 fixé et retourne un indépendant de H de cardinal voulu. **Q.E.D.**

Dans le cas où on a $\alpha(H) \leq k_0$, $J_H^{k_0}$ est un indépendant de H de cardinal maximal. Dans le cas contraire, $J_H^{k_0}$ est un indépendant de H de cardinal k_0.

Lemme 3.9 *Soit une constante entière $k_0 \geq 0$ et soit une fonction ρ associant, à chaque hypergraphe r-uniforme H, un réel $\rho(H) \geq 1$.*

*On considère la restriction de r-*MISH *aux hypergraphes r-uniformes H vérifiant $\alpha(H) \,/\, \rho(H) \geq k_0$. On suppose que cette restriction admet un algorithme d'approximation A de borne $\rho(H)$.*

*Alors, le problème r-*MISH *dans sa généralité admet un algorithme d'approximation de borne $\rho(H)$.*

Preuve. On note $\mathcal{H}^r_{k_0,\rho}$ la classe des hypergraphes r-uniformes H vérifiant $\alpha(H) \,/\, \rho(H) \geq k_0$. Soit un polynôme f tel que, pour tout $H \in \mathcal{H}^r_{k_0,\rho}$ pris en entrée, l'algorithme A s'arrête au bout d'un temps au plus $f(|H|)$.

On considère l'algorithme B suivant. Pour chaque hypergraphe r-uniforme H pris en entrée, B commence par simuler le comportement de l'algorithme A sur H en comptant le nombre d'opérations simulées.

(i). Si au bout d'un temps au plus $f(|H|)$, l'algorithme A retourne un indépendant I de H de cardinal au moins k_0 alors, B retourne I.

(ii). Dans le cas contraire (c'est-à-dire lorsqu'au bout d'un temps $f(|H|)$, A n'a rien retourné ou a retourné un objet qui n'est *pas* un indépendant de H de cardinal au moins k_0), B retourne $J_H^{k_0}$.

L'algorithme B prend un temps polynomial (on exploite le lemme 3.8 p. 74 pour implanter (ii)). Il reste à montrer que B admet pour borne $\rho(H)$.

• Supposons tout d'abord que $H \in \mathcal{H}^r_{k_0,\rho}$. L'algorithme A appliqué à H s'arrête au bout d'un temps au plus $f(|H|)$ et retourne, par hypothèse, un indépendant I de H de cardinal $\#I \geq \alpha(H) \,/\, \rho(H) \geq k_0$. Ainsi, l'algorithme B se trouve dans le cas (i) et retourne I. Il se comporte donc de manière correcte dans ce cas.

• Supposons maintenant que $H \notin \mathcal{H}^r_{k_0,\rho}$. On a alors $k_0 > \alpha(H) \,/\, \rho(H)$ mais aussi $\rho(H) \geq 1$ d'où $\min\{\alpha(H), k_0\} \geq \alpha(H) \,/\, \rho(H)$. Dans le cas ($i$), B retourne un indépendant de H de cardinal au moins k_0 et, dans le cas (ii), B retourne un indépendant de H de cardinal $\min\{\alpha(H), k_0\}$. De toute façon, B retourne un indépendant de H de cardinal au moins $\min\{\alpha(H), k_0\}$ ce qui est suffisant.

Q.E.D.

3.6.1.2 Résultats concernant MISH

Nous prouvons ici les résultats annoncés au début la section 3.6.1 (théorèmes 3.2 p. 76 et 3.3 p. 76 ci-dessous). Nos preuves consistent en des réductions de 2-MISH à r-MISH. Elles s'appuient sur le gadget suivant :

Définition 3.9 (Le gadget)
Soit $G = (V, \mathcal{E})$ un graphe quelconque. On note H^r_G l'hypergraphe r-uniforme
- *admettant V pour ensemble de sommets et*
- *dont les hyperarêtes sont les parties de V de cardinal r admettant comme sous-ensemble au moins un élément de $\mathcal{E}$.*

Notons que Krivelevich et Sudakov utilisent le gadget ci-dessus pour démontrer l'analogue de notre théorème 3.3 p. 76 correspondant au problème de la coloration [64].

Le lemme suivant décrit le fonctionnement de notre gadget :

Lemme 3.10 *Soit G un graphe.*

(i). *L'application qui, à tout graphe G, associe H^r_G est calculable en temps polynomial.*

(ii). *Tout indépendant de G est un indépendant de H^r_G.*

(iii). Tout indépendant de H_G^r de cardinal au moins r est un indépendant de G.

Preuve.

(i). Calculer l'ensemble des sommets de H_G^r ne pose pas de problème puisque c'est le même que celui de G. De plus, on peut calculer en temps polynomial l'ensemble des arêtes de H_G^r à partir des arêtes de G tout simplement en utilisant la force brute. Dans un premier temps, on énumère les $\binom{|G|}{r} = O(|G|^r)$ ensembles de sommets de G de cardinal r. Ensuite, on sélectionne parmi eux ceux qui contiennent au moins une arête de G. Cette procédure prend un temps polynomial à r fixé.

(ii). Le point *(ii)* se déduit du fait que toute arête de H_G^r admet pour sous-ensemble une arête de G.

(iii). On justifie le point *(iii)* en remarquant que tout ensemble de sommets de cardinal au moins r contenant une arête E de G, contient aussi une hyperarête de H_G^r contenant E.

Q.E.D.

Notons que, en général, G et H_G^r n'ont pas exactement les mêmes indépendants.

Exemple 3.4 *Pour $r = 3$ et $G = ([1,4], \{\{1,2\},\{1,3\},\{2,4\}\})$, on a :*

$$H_G^3 = ([1,4], \{\{1,2,3\},\{1,2,4\},\{1,3,4\},\{2,3,4\}\}) .$$

L'arête $\{1,2\}$ de G est un indépendant de H_G^3 mais pas de G.

Théorème 3.2
*Le problème de décision associé au problème r-*MISH *est* W[1]*-difficile pour le paramètre k (le seuil d'acceptation de l'instance). De plus, quelle que soit la constante entière $k_0 \geq 0$, r-*MISH$_D$ *reste* W[1]*-difficile même si on le restreint aux instances (H,k) pour lesquelles on a $k \geq k_0$.*

Preuve. On va réduire

la restriction de 2-MISH$_D$ aux instances (G,k) vérifiant $k \geq k_0$
à
la restriction de r-MISH$_D$ aux instances (H,k) vérifiant $k \geq k_0$,

de manière à pouvoir appliquer le lemme 3.7 p. 74.

Sans perte de généralité, on peut supposer $k_0 \geq r$. Transformons toute instance (G,k) de 2-MISH$_D$ avec $k \geq k_0$, en l'instance (H_G^r, k) de r-MISH$_D$.

Le point *(i)* du lemme 3.10 p. 75 garantit que cette construction peut s'effectuer en temps polynomial. De plus, le seuil d'acceptation de l'instance réduite (H_G^r, k) est le même que celui de l'instance initiale (G,k). Enfin, les points *(ii)* et *(iii)* du lemme 3.10 p. 75 garantissent que, pour tout ensemble I de cardinal k, G admet I comme indépendant si et seulement si H admet I comme indépendant. On a ainsi montré que notre transformation était en même temps une FPT-réduction et une réduction de Karp. **Q.E.D.**

Théorème 3.3
*Soit une application b associant, à chaque entier naturel n, un réel $b(n) \geq 1$. On suppose que r-*MISH *admet un algorithme d'approximation de borne $b(|H|)$.*

Alors, il existe un algorithme d'approximation pour 2-MISH *de borne $b(|G|)$.*

Preuve. Soit A un algorithme d'approximation pour r-MISH de borne $b(|H|)$. On va construire un algorithme d'approximation de borne $b(|G|)$ pour la restriction de 2-MISH aux graphes G vérifiant $\alpha(G)/b(|G|) \geq r$. On pourra alors appliquer le lemme 3.9 p. 75 avec $r := 2$ et $\rho(G) := b(|G|)$, et on aura prouvé notre théorème 3.3.

Supposons donné un graphe G vérifiant $\alpha(G) \,/\, b(|G|) \geq r$. On procède la manière suivante :

1. on calcule l'hypergraphe r-uniforme H_G^r puis
2. en simulant une application de l'algorithme A sur H_G^r, on calcule un indépendant I de H_G^r de cardinal au moins $\alpha(H_G^r) \,/\, b(|H_G^r|)$ et on retourne I.

L'algorithme décrit ci-dessus est implantable en temps polynomial : l'étape 1 d'après le point (i) du lemme 3.10 p. 75 et l'étape 2 parce que A est polynomial. Il reste à montrer que notre algorithme admet bien la borne annoncée c'est-à-dire qu'il retourne toujours un indépendant I de G de cardinal $\#I \geq \alpha(G) \,/\, b(|G|)$.

Comme G et H_G^r ont le même ensemble de sommets, on a $|H_G^r| = |G|$. De plus, le point (ii) du lemme 3.10 p. 75 donne $\alpha(G) \leq \alpha(H_G^r)$. On en déduit que l'ensemble I calculé à l'étape 2 vérifie

$$\#I \geq \frac{\alpha(H_G^r)}{b(|H_G^r|)} = \frac{\alpha(H_G^r)}{b(|G|)} \geq \frac{\alpha(G)}{b(|G|)}\,.$$

Or, on s'est restreint aux graphes G vérifiant $\alpha(G) \,/\, b(|G|) \geq r$ donc on peut appliquer à I le point (iii) du lemme 3.10 p. 75 : I est bien un indépendant de G. **Q.E.D.**

Håstad a démontré que, pour tout réel δ vérifiant $0 \leq \delta < 1$, il n'existe pas d'algorithme d'approximation pour 2-MISH de borne $|G|^{\delta}$, sauf si NP = ZPP [47]. Le théorème 3.3 p. 76 permet de généraliser ce résultat :

Corollaire 3.1
Soit un réel δ avec $0 \leq \delta < 1$.

*Alors, il n'existe pas d'algorithme d'approximation pour r-*MISH *de borne $|H|^{\delta}$, sauf si* NP = ZPP*.*

3.6.2 Inapproximabilité de LCS, LCUS, LCCS et LCUCS

Dans cette section, nous considérons les problèmes d'optimisation LCUS, LCCS et LCUCS introduits page 64. Nous généralisons à ces trois problèmes les résultats connus sur la difficulté à approximer LCS lorsque le cardinal de l'alphabet d'entrée (c'est-à-dire $\sigma(X)$) est non borné [55]. Nous montrons par réduction que

1. LCS et LCUS sont au moins aussi difficiles à approximer que le problème de l'indépendant maximum pour les graphes (théorème 3.4 p. 80), et que
2. LCCS et LCUCS sont au moins aussi difficiles à approximer que le problème de l'indépendant maximum pour les hypergraphes 3-uniformes (théorème 3.5 p. 83).

Notons que, d'après le théorème 3.3 p. 76, le problème de l'indépendant maximum pour les hypergraphes 3-uniformes est au moins aussi difficile à approximer que le problème de l'indépendant maximum pour les graphes.

Comme sous-produit de nos réductions, nous obtenons que chacun des trois problèmes de décision LCUS_D, LCCS_D et LCUCS_D est W[1]-difficile pour son paramètre "seuil d'acceptation" (propositions 3.6 p. 80 et 3.7 p. 83). Notons qu'il est connu depuis 1995, que LCS_D est W[2]-difficile pour ce paramètre [14].

3.6.2.1 Mots croissants, mots décroissants

Dans toute la suite de la section 3.6.2, nous convenons d'identifier l'ensemble des symboles de l'univers avec l'ensemble $\mathbb{N}$ des entiers naturels. On peut donc définir naturellement les notions de mots monotones et strictement monotones.

Définition 3.10
Soient w un mot et $n := |w|$. On dit que :
- *w est* croissant *lorsqu'on a $w[1] \leq w[2] \leq \cdots \leq w[n]$;*
- *w est* décroissant *lorsqu'on a $w[1] \geq w[2] \geq \cdots \geq w[n]$;*
- *w est* strictement croissant *lorsqu'on a $w[1] < w[2] < \cdots < w[n]$;*
- *w est* strictement décroissant *lorsqu'on a $w[1] > w[2] > \cdots > w[n]$.*

Tout mot strictement croissant ou strictement décroissant est une permutation mais la réciproque est fausse comme le montre le contre-exemple 132. Les mots 233355699 et 23569 sont respectivement croissant et strictement croissant. Leurs images miroirs sont respectivement décroissante et strictement décroissante. Plus généralement, on peut faire la remarque suivante :

Remarque 3.9 *Un mot est croissant (resp. strictement croissant) si et seulement si son image miroir est décroissante (resp. strictement décroissante).*

Remarque 3.10 *Si un mot donné a la propriété d'être croissant (resp. strictement croissant, resp. décroissant, resp. strictement décroissant) alors toutes ses sous-séquences possèdent la même propriété.*

Remarque 3.11 *Le mot vide et les mots réduits à une lettre sont les seuls mots à être la fois croissants et strictement décroissants (resp. décroissants et strictement croissants).*

Le lemme 3.11 ci-après permet de généraliser aux problèmes LCUS et LCUCS les gadgets initialement prévus pour leurs versions "orientées" LCS et LCCS (voir les lemmes 3.12 p. 79 et 3.14 p. 81).

Lemme 3.11 *Soit un entier $n \geq 1$ et soit un mot w pouvant s'écrire comme la concaténation de n mots croissants.*

(*i*). *Toute U-sous-séquence strictement croissante de w de longueur au moins $n+1$ est une sous-séquence de w.*

(*ii*). *Toute UC-sous-séquence strictement croissante de w de longueur au moins $n+2$ est une C-sous-séquence de w.*

Preuve. Par hypothèse, on peut écrire w sous la forme $w = w_1 w_2 \dots w_n$ où w_i est un mot croissant pour tout $i \in [1, n]$.

(*i*). *Soit s une U-sous-séquence strictement croissante de w vérifiant $|s| \geq n+1$. Montrons que s est une sous-séquence de w.*

Comme s est une U-sous-séquence de w, s ou $\tilde{s}$ est une sous-séquence de w. On va raisonner par l'absurde et supposer que $\tilde{s}$ est une sous-séquence de w. Ceci permet d'écrire s sous la forme $\tilde{s} = s_1 s_2 \dots s_n$ avec s_i sous-séquence de w_i pour tout $i \in [1, n]$. Fixons provisoirement $i \in [1, n]$. Le mot s_i est à la fois une sous-séquence de w_i et une sous-séquence de $\tilde{s}$. Or,

– w_i est croissant et

– $\tilde{s}$ est strictement décroissant, comme image miroir du mot strictement croissant s (remarque 3.9 p. 77).

Ainsi, d'après la remarque 3.10 p. 78, s_i est à la fois croissant et strictement décroissant. Il en résulte que s_i est vide ou réduit à une lettre (remarque 3.11 p. 78). Comme ceci est vrai pour tout i, on a $|s| = |s_1| + |s_2| + \dots + |s_n| \leq n$: contradiction.

(*ii*). *Soit s' une UC-sous-séquence strictement croissante de w vérifiant $|s'| \geq n+2$. Montrons que s' est une C-sous-séquence de w.*

s' est une U-sous-séquence d'un certain conjugué w' de w (voir la remarque 3.3 p. 63). Or, tout conjugué de w s'écrit comme la concaténation de (au plus) $n+1$ mots strictement croissants. On peut ainsi appliquer le point (*i*) avec : w' au lieu de w, $n+1$ au lieu de n et s' au lieu de s. On obtient ainsi que s' est une sous-séquence de w' et donc, s' est une C-sous-séquence de w.

Q.E.D.

L'exemple suivant montre que les deux bornes données dans le lemme précédent sont optimales.

Exemple 3.5 *Soient $w := 1623534125$. Le mot w peut s'écrire comme la concaténation de $n := 4$ mots strictement croissants :* 16*,* 235*,* 34*, et* 125*.*

(*i*). *$s := 2356$ est une U-sous-séquence strictement croissante de w de longueur n, mais s n'est pas une sous-séquence de w.*

(*ii*). *$s' := 12356$ est une UC-sous-séquence strictement croissante de w de longueur $n+1$ (car s' est une sous-séquence de l'image miroir 1521435326 du conjugué $\gamma(w) = 6235341251$ de w) ; en revanche, s' n'est pas une C-sous-séquence de w.*

La notation suivante sera intensivement utilisée dans la suite pour encoder des ensembles avec des mots.

Définition 3.11
Pour tout ensemble fini $S \subseteq \mathbb{N}$, on note $S\uparrow$ l'unique mot sur S strictement croissant de longueur $\#S$.

Autrement dit, $S\uparrow$ est le résultat du *tri* par ordre croissant des $\#S$ éléments de S. Par exemple, $\{1,8,9,4\}\uparrow = 1489$, $[2,6]\uparrow = 23456$ et $\emptyset\uparrow = \varepsilon$. Pour tout $i \in [1,\#S]$, $S\uparrow[i]$ est le i-ème élément de S pour l'ordre croissant et $S = \{S\uparrow[1], S\uparrow[2], \ldots, S\uparrow[\#S]\}$.

On a de plus la propriété de monotonie :

Remarque 3.12 *Pour tous ensemble finis $S, T \subseteq \mathbb{N}$, on a $S \subseteq T$ si et seulement si $S\uparrow$ est une sous-séquence de $T\uparrow$.*

3.6.2.2 Résultats concernant LCS et LCUS

Pour tout graphe $G = (V, \mathcal{E})$ et tout sommet $v \in V$, on note $\bar{N}_G(v) := \{u \in V : \{u,v\} \notin \mathcal{E}\}$ l'ensemble des sommets de G non adjacents à v. Remarquons que $v \in \bar{N}_G(v)$.

Le gadget suivant, inspiré par Jiang et Li [55], permet de réduire 2-MISH à LCS et à LCUS de manière à pouvoir démontrer la proposition 3.6 p. 80 et le théorème 3.4 p. 80 :

Définition 3.12 (Le gadget)
Soit G un graphe. On pose $n := |G|$ et on suppose que G admet $[1,n]$ pour ensemble de sommets.

On définit les mots suivants :

$$x_{G,1} := [1,n]\uparrow \quad \text{et} \quad x_{G,v} := \bar{N}_G(v)\uparrow\ [1,v-1]\uparrow\ [v+1,n]\uparrow$$

pour tout $v \in [2,n]$. On pose :

$$X_G := \{x_{G,v} : v \in [1,n]\}\ .$$

Le lemme 3.13 ci-après synthétise les propriétés du gadget X_G. Décrivons succinctement le fonctionnement de ce dernier. Le mot $x_{G,1}$ force les sous-séquences communes de X_G à être strictement croissantes. Pour chaque sommet $v \neq 1$, le mot $x_{G,v}$ pose une contrainte sur les sous-séquences communes de X_G contenant v : il les empêche de contenir les lettres-sommets u adjacentes à v vérifiant $u < v$. Un mot donné est ainsi une sous-séquence commune de X_G si et seulement s'il est de la forme $I\uparrow$ avec I indépendant de G (voir les points (*iii*) et (*iv*) du lemme 3.13 p. 79). Le lemme 3.12 ci-dessous va garantir que le gadget X_G fonctionne indifféremment pour LCS et LCUS.

Lemme 3.12 *Soit G un graphe admettant $[1,n]$ pour ensemble de sommets et soit s une U-sous-séquence commune de X_G de longueur au moins 3.*

Alors, l'un des deux mots s ou $\tilde{s}$ est une sous-séquence commune strictement croissante de X_G.

Preuve. Notons t celui des mots s ou $\tilde{s}$ qui est une sous-séquence de $x_{G,1} = [1,n]\uparrow$. Alors, t est strictement croissant (remarque 3.10 p. 78).

Soit $v \in [2,n]$. t est une U-sous-séquence strictement croissante de longueur au moins 3 de $x_{G,v}$. Or, ce dernier mot s'écrit comme la concaténation des 2 mots croissants suivants : $\bar{N}_G(v)\uparrow$ et $[1,v-1]\uparrow\ [v+1,n]\uparrow$. Le point (*i*) du lemme 3.11 p. 78 garantit alors que t est une sous-séquence de $x_{G,v}$.

On ainsi montré que t est une sous-séquence commune strictement croissante de X_G. **Q.E.D.**

Rappelons que $\ell(X) = \min_{x \in X} |x|$ pour tout langage fini et non vide X.

Lemme 3.13 *Soit G un graphe admettant $[1,|G|]$ pour ensemble de sommets.*

(*i*). *L'application qui à G associe X_G est calculable en temps polynomial.*

(*ii*). $\ell(X_G) = \sigma(X_G) = \#X_G = |G|$.

(*iii*). *Soit I un indépendant de G. Alors, $I\uparrow$ est une sous-séquence commune de X_G de longueur $\#I$.*

(*iv*). *Soit s une U-sous-séquence commune de X_G de longueur au moins 3. Alors, l'ensemble $\{s[1], s[2], \ldots, s[|s|]\}$ des lettres apparaissant dans s est un indépendant de G de cardinal $|s|$.*

Preuve. Le point (*i*) est évidemment vrai. Posons $n := |G|$.

(*ii*). Tous les mots appartenant à X_G sont sur l'alphabet $[1,n]$ et $x_{G,1} = [1,n]\uparrow$ est un plus court mot de X_G. On a ainsi $\ell(X_G) = \sigma(X_G) = n$.

Par ailleurs, considérons un élément $v \in [1,n]$. On remarque que v est la seule lettre appartenant à $[1,n]$ qui n'apparaît pas dans le suffixe de $x_{G,v}$ de longueur $n-1$. Ainsi, les $x_{G,v}$ ($v \in [1,n]$) sont deux à deux distincts. On en déduit $\#X_G = n$.

(*iii*). Trivialement, $I\uparrow$ est une sous-séquence de $x_{G,1} = [1,n]\uparrow$ (remarque 3.12 p. 79). De plus, soit $v \in [2,n]$.

1. Supposons $v \notin I$. Alors, $I\uparrow$ est une sous-séquence du suffixe $[1,v-1]\uparrow\ [v+1,n]\uparrow$ de $x_{G,v}$.
2. Supposons $v \in I$. Alors, I ne contient aucun sommet adjacent à v dans G, donc on a $I \subseteq \bar{N}_G(v)$. D'après la remarque 3.12 p. 79, $I\uparrow$ est ainsi une sous-séquence du préfixe $\bar{N}_G(v)\uparrow$ de $x_{G,v}$.

Dans les deux cas, $I\uparrow$ est une sous-séquence de $x_{G,v}$. On a ainsi montré que $I\uparrow$ est une sous-séquence commune de X_G.

(*iv*). Quitte à changer s en $\tilde{s}$, on peut, par le lemme 3.12 p. 79, supposer que s est une sous-séquence commune strictement croissante de X_G.

En particulier, s ne contient pas deux occurrences d'une même lettre. Par suite, l'ensemble des lettres apparaissant dans s est bien de cardinal $|s|$.

Montrons maintenant que l'ensemble des lettres apparaissant dans s est un indépendant de G. Soient deux lettres-sommets v, v' distinctes apparaissant dans s. On va vérifier que $\{v,v'\} \notin \mathcal{E}$ ou encore que $v' \in \bar{N}_G(v)$. Quitte à échanger v et v', on peut supposer que $v'v$ est une sous-séquence de s. Comme s est une sous-séquence de $x_{G,v}$, il en est de même pour $v'v$. Or, la seule occurrence de v dans $x_{G,v}$ se situe dans son préfixe $\bar{N}_G(v)\uparrow$. Ainsi, v' apparaît également dans ce préfixe. Il en résulte $v' \in \bar{N}_G(v)$.

Q.E.D.

Proposition 3.6
Les problèmes LCS_D *et* LCUS_D *sont* W[1]*-difficiles pour le paramètre* k *(c'est-à-dire la longueur minimum acceptable pour une plus longue sous-séquence commune de* X*).*

Preuve. Considérons l'application qui, à toute instance (G,k) de 2-MISH_D, associe l'instance (X_G,k) de LCS_D (resp. LCUS_D). Cette application est calculable en temps polynomial (point (*i*) du lemme 3.13 p. 79) et conserve les seuils d'acceptation : le seuil d'acceptation de l'instance retournée est le même que celui de l'instance prise en argument. De plus, si on restreint 2-MISH_D aux instances (G,k) pour lesquelles on a $k \geq 3$, notre application devient une M-réduction (points (*iii*) et (*iv*) du lemme 3.13 p. 79). On peut ainsi appliquer le lemme 3.7 p. 74 et conclure. **Q.E.D.**

Théorème 3.4
Soit une application b *de* $\mathbb{N}\times\mathbb{N}\times\mathbb{N}$ *vers l'ensemble des réels supérieurs ou égaux à* 1.

On suppose que l'un des problèmes LCS *ou* LCUS *admet un algorithme d'approximation de borne* $b(\ell(X),\sigma(X),\#X)$*. Alors,* 2-MISH *admet un algorithme d'approximation de borne* $b(|G|,|G|,|G|)$*.*

Preuve. Soit A un algorithme d'approximation pour LCS (resp. LCUS) de borne $b(\ell(X),\sigma(X),\#X)$. Étant donné un graphe G, on cherche à construire un algorithme d'approximation pour 2-MISH de borne $b(|G|,|G|,|G|)$. Le lemme 3.9 p. 75, appliqué avec $r=2$ et $k_0=3$ garantit que l'on peut se restreindre au cas où $\alpha(G) \,/\, b(|G|,|G|,|G|)$ est au moins 3, ce que l'on fera désormais.

On procède de la manière suivante.

1. On calcule le langage X_G.
2. En simulant une application de l'algorithme A sur X_G, on calcule une sous-séquence (resp. une U-sous-séquence) commune s de X_G vérifiant :

$$|s| \geq \frac{\mathrm{lcs}(X_G)}{b(\ell(X_G),\sigma(X_G),\#X_G)} \quad \left(\text{resp. } |s| \geq \frac{\mathrm{lcus}(X_G)}{b(\ell(X_G),\sigma(X_G),\#X_G)}\right).$$

3. On retourne l'ensemble I des lettres apparaissant dans s.

Cet algorithme s'implante en temps polynomial :
– l'étape 1 d'après le point (*i*) du lemme 3.13 p. 79,

– l'étape 2 car l'algorithme A est polynomial et
– l'étape 3 trivialement.

Montrons maintenant que notre algorithme approxime bien 2-MISH avec la borne souhaitée.

Les points (ii) et (iii) du lemme 3.13 p. 79, garantissent respectivement

$$b\left(\ell(X_G), \sigma(X_G), \#X_G\right) = b(|G|, |G|, |G|)$$

et

$$\operatorname{lcus}(X_G) \geq \operatorname{lcs}(X_G) \geq \alpha(G)\,.$$

On en déduit :

$$|s| \geq \frac{\operatorname{lcs}(X_G)}{b\left(\ell(X_G), \sigma(X_G), \#X_G\right)} = \frac{\operatorname{lcs}(X_G)}{b(|G|, |G|, |G|)} \geq \frac{\alpha(G)}{b(|G|, |G|, |G|)} \geq 3\,.$$

Ainsi, d'après le point (iv) du lemme 3.13 p. 79, l'ensemble I retourné par notre algorithme est un indépendant de G de cardinal $\#I = |s| \geq \alpha(G) \,/\, b(|G|, |G|, |G|)$. **Q.E.D.**

3.6.2.3 Résultats concernant LCCS et LCUCS

La section précédente traite des problèmes LCS et LCUS. La section présente est son pendant pour les problèmes LCCS et LCUCS. Elle est organisée de la même façon. Le tableau suivant montre comme les résultats démontrés dans ces deux sections se correspondent.

Problèmes	LCS et LCUS	LCCS et LCUCS
Section	sec. 3.6.2.2	sec. 3.6.2.3
Définition du gadget	déf. 3.12 p. 79	déf. 3.13 p. 81
Gadget et (non) orientation	lem. 3.12 p. 79	lem. 3.14 p. 81
Propriétés du gadget	lem. 3.13 p. 79	lem. 3.15 p. 82
Complexité paramétrique	prop. 3.6 p. 80	prop. 3.7 p. 83
Inapproximabilité	th. 3.4 p. 80	th. 3.5 p. 83

Définition 3.13 (Le gadget)
Soit $H = (V, \mathcal{E})$ un hypergraphe 3-uniforme avec $V \subseteq \mathbb{N}$.

Pour chaque hyperarête $E \in \mathcal{E}$, on définit un mot

$$y_{H,E} := (V \setminus E)\!\uparrow\ E\!\uparrow[3]\ \left((V \setminus E)\!\uparrow\right)^2\ E\!\uparrow[2]\ \left((V \setminus E)\!\uparrow\right)^2\ E\!\uparrow[1]\ (V \setminus E)\!\uparrow$$

et on pose

$$Y_H := \{y_{H,E} : E \in \mathcal{E}\} \cup \{(V\!\uparrow)^p : p \in [1, m_H]\}$$

où m_H est un entier non nul, ajusté pour que $\#Y_H = |H|^3$ (cet entier existe car on a $\#\{y_{H,E} : E \in \mathcal{E}\} \leq \#\mathcal{E} \leq \binom{|H|}{3} < |H|^3$).

Le lemme 3.15 ci-dessous résume les propriétés de notre gadget. Décrivons succinctement le fonctionnement de ce dernier. Le mot $V\!\uparrow$ force toute C-sous-séquence commune de Y_H à admettre un conjugué strictement croissant. Les mots $(V\!\uparrow)^p$ pour $p \in [2, m_H]$ servent simplement à ajuster le cardinal de Y_H. Pour chaque hyperarête $E = \{E\!\uparrow[1], E\!\uparrow[2], E\!\uparrow[3]\} \in \mathcal{E}$, $y_{H,E}$ empêche les C-sous-séquences communes strictement croissantes de Y_H de contenir simultanément les trois lettres $E\!\uparrow[1]$, $E\!\uparrow[2]$ et $E\!\uparrow[3]$. Un mot donné est ainsi une C-sous-séquence de Y_H si et seulement s'il admet un conjugué de la forme $I\!\uparrow$ avec I indépendant de H (voir les points (iii) et (iv) du lemme 3.15 p. 82). Le lemme 3.14 ci-dessous permet de montrer que le gadget Y_H fonctionne indifféremment pour LCCS et LCUCS.

Lemme 3.14 *Soit $H = (V, \mathcal{E})$ un hypergraphe 3-uniforme avec $V \subseteq \mathbb{N}$ et soit s une UC-sous-séquence commune de Y_H de longueur au moins 11.*

Alors, un conjugué de s ou de $\tilde{s}$ est une C-sous-séquence strictement croissante de Y_H.

Preuve. Notons t celui des mots s ou $\tilde{s}$ qui est une C-sous-séquence de $V\uparrow$. Notons u celui des conjugués de t qui est une sous-séquence de $V\uparrow$. Alors, u est strictement croissant. Il reste à vérifier que u est une C-sous-séquence commune de Y_H.

D'une part, comme u est une sous-séquence de $V\uparrow$, u est également une sous-séquence de chacun des $(V\uparrow)^p$ ($p \in [1, m_H]$).

D'autre part, soit une hyperarête de $E \in \mathcal{E}$: u est une UC-sous-séquence strictement croissante de longueur au moins 11 de $y_{H,E}$. Or, ce mot s'écrit comme la concaténation des 9 mots croissants suivants : $(V \setminus E)\big\uparrow$, $E\uparrow[3]$, $(V \setminus E)\big\uparrow$, $(V \setminus E)\big\uparrow$, $E\uparrow[2]$, $(V \setminus E)\big\uparrow$, $(V \setminus E)\big\uparrow$, $E\uparrow[1]$ et $(V \setminus E)\big\uparrow$. Le point (ii) du lemme 3.11 p. 78 garantit alors que u est une C-sous-séquence de $y_{H,E}$.

On a ainsi montré que u est une C-sous-séquence de chacun des mots appartenant à Y_H. **Q.E.D.**

Lemme 3.15 *Soit $H = (V, \mathcal{E})$ un hypergraphe 3-uniforme avec $V \subseteq \mathbb{N}$.*

(i). L'application qui à H associe Y_H est calculable en temps polynomial.

(ii). $\ell(Y_H) = \sigma(Y_H) = |H|$ et $\#Y_H = |H|^3$.

(iii). Soit I un indépendant de H. Alors, $I\uparrow$ est une C-sous-séquence commune de Y_H de longueur $\#I$.

(iv). Soit s une UC-sous-séquence commune de Y_H de longueur au moins 11. Alors, l'ensemble des lettres apparaissant dans s est un indépendant de H de cardinal $|s|$.

Preuve.

(i). On calcule facilement, à partir de H, les $y_{H,E}$ ($E \in \mathcal{E}$) en temps polynomial. On ajoute ensuite successivement les mots $V\uparrow$, $(V\uparrow)^2$, $(V\uparrow)^3$, ... au langage $\{y_{H,E} : E \in \mathcal{E}\}$ jusqu'à obtenir un ensemble de cardinal $|H|^3$ qui n'est autre que Y_H.

(ii). Le cardinal de Y_H est, par définition, égal à $|H|^3$. De plus, tous les mots appartenant à Y_H sont sur l'alphabet V et $V\uparrow$ est le mot le plus court de Y_H. On a ainsi $\ell(Y_H) = \sigma(Y_H) = |H|$.

(iii). Tout d'abord, $I\uparrow$ est une sous-séquence de $(V\uparrow)^p$ pour tout entier $p \geq 1$.

De plus, soit $E \in \mathcal{E}$. $y_{H,E}$ est conjugué avec chacun des deux mots

$$y^{132}_{H,E} := (V \setminus E)\big\uparrow\ E\uparrow[1]\ \big((V \setminus E)\big\uparrow\big)^2\ E\uparrow[3]\ \big((V \setminus E)\big\uparrow\big)^2\ E\uparrow[2]\ (V \setminus E)\big\uparrow$$

et

$$y^{213}_{H,E} := (V \setminus E)\big\uparrow\ E\uparrow[2]\ \big((V \setminus E)\big\uparrow\big)^2\ E\uparrow[1]\ \big((V \setminus E)\big\uparrow\big)^2\ E\uparrow[3]\ (V \setminus E)\big\uparrow\ .$$

On voit alors que :

- $(V \setminus \{E\uparrow[3]\})\big\uparrow$ est une sous-séquence de $y^{132}_{H,E}$,
- $(V \setminus \{E\uparrow[1]\})\big\uparrow$ est une sous-séquence de $y^{213}_{H,E}$, et que
- $(V \setminus \{E\uparrow[2]\})\big\uparrow$ est une sous-séquence de $y^{132}_{H,E}$ et de $y^{213}_{H,E}$.

On en déduit que $y_{H,E}$ admet pour C-sous-séquences :

$$(V \setminus \{E\uparrow[1]\})\big\uparrow,\ (V \setminus \{E\uparrow[2]\})\big\uparrow \text{ et } (V \setminus \{E\uparrow[3]\})\big\uparrow.$$

Or, $I\uparrow$ est une sous-séquence de l'un de ces trois mots puisque I est une partie de V n'admettant pas simultanément pour éléments les trois extrémités $E\uparrow[1]$, $E\uparrow[2]$ et $E\uparrow[3]$ de E. On en déduit que $I\uparrow$ est une C-sous-séquence de $y_{H,E}$.

(iv). Quitte à changer s en un conjugué de s ou de $\tilde{s}$, on peut, par le lemme 3.14 p. 81, supposer que s est une C-sous-séquence strictement croissante de Y_H.

En particulier, l'ensemble des lettres apparaissant dans s est bien de cardinal $|s|$.

Montrons maintenant que l'ensemble des lettres apparaissant dans s est un indépendant de H.

Soit $E \in \mathcal{E}$. Soit s' un conjugué de s qui est une sous-séquence de $y_{H,E}$. Supposons (absurde) que les 3 lettres-sommets appartenant à E apparaissent simultanément dans s. Elles apparaissent alors dans s dans l'ordre croissant $E\uparrow[1]\ E\uparrow[2]\ E\uparrow[3]$. Par suite, elles apparaissent dans s' dans l'un des trois ordres suivants : $E\uparrow[1]\ E\uparrow[2]\ E\uparrow[3]$, $E\uparrow[2]\ E\uparrow[3]\ E\uparrow[1]$ ou $E\uparrow[3]\ E\uparrow[1]\ E\uparrow[2]$. Or, les éléments de E n'apparaissent dans $y_{H,E}$ que dans l'ordre $E\uparrow[3]\ E\uparrow[2]\ E\uparrow[1]$. Ceci contredit le fait que s' est une sous-séquence de $y_{H,E}$.

Q.E.D.

Proposition 3.7
Les problèmes LCCS_D *et* LCUCS_D *sont* W[1]*-difficiles pour leur paramètre* k *(c'est-à-dire la longueur minimum acceptable pour une plus longue sous-séquence commune de* X*).*

Preuve. Considérons l'application qui, à toute instance (H, k) de 3-MISH_D avec $k \geq 11$, associe l'instance (Y_H, k) de LCCS_D (resp. LCUCS_D).

Si l'on restreint 3-MISH_D à ses instances possédant un seuil d'acceptation supérieur ou égal à 11, notre application devient une M-réduction (points (iii) et (iv) du lemme 3.15 p. 82), calculable en temps polynomial (point (i) du lemme 3.15 p. 82). On peut ainsi appliquer le théorème 3.2 p. 76 et conclure. **Q.E.D.**

Théorème 3.5
Soit une application b *de* $\mathbb{N} \times \mathbb{N} \times \mathbb{N}$ *vers l'ensemble des réels supérieurs ou égaux à* 1*. On suppose que l'un des problèmes* LCCS *ou* LCUCS *admet un algorithme d'approximation de borne* $b(\ell(X), \sigma(X), \#X)$*. Alors,* 3-MISH *admet un algorithme d'approximation de borne* $b\left(|H|, |H|, |H|^3\right)$*.*

Preuve. Soit A un algorithme d'approximation pour LCCS ou LCUCS de borne $b(\ell(X), \sigma(X), \#X)$.

Étant donné un hypergraphe 3-uniforme H, on cherche à construire un algorithme d'approximation pour 3-MISH de borne $b\left(|H|, |H|, |H|^3\right)$. Sans perte de généralité, on peut supposer $\alpha(H) \,/\, b\left(|H|, |H|, |H|^3\right) \geq 11$ (lemme 3.9 p. 75). On construit alors le langage Y_H, puis, à l'aide de A, on calcule une UC-sous-séquence commune s de Y_H de longueur au moins $\text{lccs}(Y_H) \,/\, b(\ell(Y_H), \sigma(Y_H), \#Y_H)$. On retourne ensuite l'ensemble I des lettres apparaissant dans s.

On déduit du lemme 3.15 p. 82 que cet algorithme réalise une approximation de 3-MISH de borne $b\left(|H|, |H|, |H|^3\right)$: la démarche est analogue à celle de la preuve du théorème 3.4 p. 80. **Q.E.D.**

3.6.2.4 Conclusion

Sous l'hypothèse admise, mais non encore démontrée, ZPP $\subsetneq$ NP, nous pouvons montrer des bornes fines concernant l'approximabilité de LCS, LCUS, LCCS et LCUCS : il suffit de combiner les divers résultats démontrés dans les sections 3.5 et 3.6.

Corollaire 3.2
Soit un réel δ *avec* $0 \leq \delta \leq 1$*. Sous l'hypothèse* ZPP $\subsetneq$ NP*, on a les deux équivalences suivantes.*

- (i). *Il existe un algorithme d'approximation de borne* $(\sigma(X))^\delta$ *pour l'un des quatre problèmes* LCS, LCUS, LCCS *ou* LCUCS *si et seulement si* $\delta = 1$.
- (ii). *Il existe un algorithme d'approximation de borne* $(\ell(X))^\delta$ *pour l'un des quatre problèmes* LCS, LCUS, LCCS *ou* LCUCS *si et seulement si* $\delta = 1$.

Preuve. La proposition 3.4 p. 71 garantit que chacun des quatre problèmes LCS, LCUS, LCCS et LCUCS admet un algorithme d'approximation de borne $\sigma(X)$. De plus, nous avons remarqué juste après la preuve de la proposition 3.4 p. 71 que nos quatre problèmes admettaient $\ell(X)$ comme borne d'approximation.

Supposons maintenant que, pour un certain $\delta < 1$, l'un des deux problèmes LCS ou LCUS (resp. LCCS ou LCUCS) admette un algorithme d'approximation de borne $(\sigma(X))^\delta$ ou un algorithme d'approximation de borne $(\ell(X))^\delta$. Le théorème 3.4 p. 80 (resp. 3.5 p. 83) garantit alors que le problème 2-MISH (resp. 3-MISH) admet un algorithme d'approximation de borne $|H|^\delta$. On en déduit que NP = ZPP par le corollaire 3.1 p. 77, contre notre hypothèse. **Q.E.D.**

Corollaire 3.3
Sous l'hypothèse ZPP $\subsetneq$ NP*, on a les deux bornes suivantes.*

- (i). *Soit un réel* δ *avec* $0 \leq \delta < 1$*. Alors, il n'existe pas d'algorithme d'approximation de borne* $(\#X)^\delta$ *pour* LCS*, ni pour* LCUS.

(*ii*). *Soit un réel* δ *avec* $0 \leq \delta < 1/3$. *Alors, il n'existe pas d'algorithme d'approximation de borne* $(\#X)^{\delta}$ *pour* LCCS, *ni pour* LCUCS.

Preuve.

(*i*). Supposons (absurde) que l'un des problèmes LCS ou LCUS admette un algorithme d'approximation de borne $(\#X)^{\delta}$ avec $0 \leq \delta < 1$. Par le théorème 3.4 p. 80, 2-MISH admet alors un algorithme d'approximation de borne $|G|^{\delta}$. On tire alors de [47] (voir aussi corollaire 3.1 p. 77) que NP = ZPP. Ceci contredit notre hypothèse.

(*ii*). Supposons (absurde) que l'un des problèmes LCCS ou LCUCS admette un algorithme d'approximation de borne $(\#X)^{\delta}$ avec $0 \leq \delta < 1/3$.Alors, par le théorème 3.5 p. 83, 3-MISH admet un algorithme d'approximation de borne $\left(|H|^{\delta}\right)^3 = |H|^{3\delta}$. Comme 3δ est strictement inférieur à 1, le corollaire 3.1 p. 77 garantit que NP = ZPP, contre notre hypothèse.

Q.E.D.

3.7 Questions ouvertes

3.7.1 Algorithmes polynomiaux

Pour justifier que LCUS, LCCS et LCUCS sont dans NPO (remarque 3.6 p. 67, propriété ($NPO3$)), ainsi que pour montrer la remarque 3.7 p. 68 et le lemme 3.6 p. 72, nous avons implicitement utilisé le fait suivant.

Remarque 3.13 *Étant donnés deux mots* s *et* x *avec* $|s| \leq |x|$, *on peut tester si* s *est une sous-séquence (resp. une U-sous-séquence, resp. une C-sous-séquence, resp. une UC-sous-séquence) de* x *en temps polynomial.*

Détaillons un peu.

- À l'aide d'un algorithme glouton, on peut décider si s est une sous-séquence de x en temps linéaire [78, chapitre 1]. En appliquant cet algorithme une première fois à s et x et une seconde fois à $\tilde{s}$ et x, on peut également décider en temps linéaire si s est une U-sous-séquence de x.
- D'autre part, on peut décider en temps quadratique $O(|s|\,|x|)$, si s est une C-sous-séquence (resp. une UC-sous-séquence) de x. En effet, il suffit de tester, pour chaque mot s' conjugué avec s, si s' est une sous-séquence (resp. une U-sous-séquence) de x (voir la remarque 3.3 p. 63).

Il est naturel de se demander si l'on peut faire mieux que la complexité quadratique énoncée dans le second point, par exemple la rendre linéaire. En effet, on peut calculer en temps $O\big(|x| \times \log\big(\sigma(\{x\})\big)\big)$ la position de toutes les sous-chaînes de x qui sont conjuguées avec s, en indexant s^2 à l'aide de son *automate des suffixes* [26].

3.7.2 Complexité paramétrique

3.7.2.1 r-MISH pour $r \geq 3$

Pour tout entier $r \geq 2$, le problème de décision associé au problème de l'indépendant maximum pour les hypergraphes r-uniformes est W[1]-difficile pour son paramètre "seuil d'acceptation" (théorème 3.2 p. 76). En revanche, reste ouverte la question de l'existence d'un entier $r \geq 3$ et d'un entier $t \geq 1$ tels que r-MISH$_D$, paramétré par k, soit dans W[t].

3.7.2.2 LCUS, LCCS et LCUCS

On trouve dans la section 1.4.3, un tableau résumant les résultats connus sur la complexité paramétrique de LCS$_D$. Nous avons généralisé presque tous ces résultats aux problèmes LCUS, LCCS et LCUCS comme le montre le tableau exhibé dans la section 3.3.3. Néanmoins, alors qu'on sait que LCS$_D$ est W[1]-complet pour le paramètre $(\#X, k)$ [14], les complexités de LCUS$_D$, LCCS$_D$ et LCUCS$_D$ pour ce paramètre restent inconnues. Il est naturel de conjecturer que ces trois derniers problèmes sont, comme LCS$_D$, W[1]-difficiles pour le paramètre $(\#X, k)$.

3.7.3 Approximabilité

3.7.3.1 Existence d'un P.T.A.S. pour LCS, LCUS, LCCS et LCUCS lorsque l'alphabet d'entrée est borné

Fixons un entier $\sigma \geq 2$ et considérons les restrictions de LCS, LCUS, LCCS et LCUCS aux instances X qui sont des langages finis et non vides sur des alphabets à σ lettres (c'est-à-dire telles que $\sigma(X) \leq \sigma$). Chacune de ces quatre restrictions admet un algorithme d'approximation de borne constante σ (proposition 3.4 p. 71), mais la question de l'existence d'un P.T.A.S. reste ouverte.

3.7.3.2 Variations sur une conjecture de Jiang et Li

Considérons la borne inférieure δ_{LCS} de l'ensemble des réels $\delta \geq 0$ pour lesquels il existe un algorithme d'approximation pour LCS de borne $(\#X)^{\delta}$. Dans leur article datant de 1995 [55], Jiang et Li montrent, par réduction depuis 2-MISH, que δ_{LCS} est strictement positif sauf si NP = P. Ils conjecturent de plus δ_{LCS} est au moins égal à 1. En combinant la réduction de Jiang et Li avec les résultats postérieurs de Håstad concernant 2-MISH [47], nous avons montré que δ_{LCS} est bien au moins égal à 1 sauf si NP = ZPP (point (i) du corollaire 3.3 p. 83). Plus généralement, nous conjecturons que, quel que soit le réel $\delta > 0$, il n'existe pas d'algorithme d'approximation de borne $(\#X)^{\delta}$ pour LCS, LCUS, LCCS ou LCUCS (en particulier, $\delta_{\mathrm{LCS}} = +\infty$).

Chapitre 4

Sur la difficulté de la conception de bonnes graines

4.1 Introduction

Parmi les principaux problèmes étudiés en bio-informatique on compte :

1. la *recherche d'homologies locales* et
2. la *recherche de motifs approchés*.

Dans les deux cas, on cherche à localiser des égalités approximatives entre des fragments de séquences biologiques (ADN ou protéine).

4.1.1 Importance pratique et théorique de la recherche d'homologies

4.1.1.1 Utilité pratique des homologies locales

La découverte d'homologies entre séquences est une tâche d'importance cruciale pour les biologistes. Supposons, par exemple, que l'on vienne de séquencer un gène et que l'on cherche à identifier :

- sa fonction,
- ses origines,
- la manière dont il est régulé ou
- la stucture tridimensionnelle de la protéine codée.

Une bonne méthode consiste à comparer notre séquence avec les séquences ou les génomes déjà annotés. Supposons que l'on puisse aligner avec un bon score une région assez longue de notre séquence contre une portion de séquence stockée dans une banque de donnée. On parle alors d'*homologie locale* et on a une bonne probabilité pour que les deux fragments alignés aient des propriétés communes héritées d'un ancêtre commun.

Logiciels. Beaucoup de logiciels dédiés à la recherche d'homologies ont été développés. Parmi ceux-ci, on peut citer FASTA [95], BLAST [2] (le plus connu), QUASAR [18], FLASH [20], YASS [92], PatternHunter I et II [79, 71], *etc.* Ces logiciels sont utilisés de manière intensive. Ils ont pour point commun de commencer le traitement de leurs données par une phase de *filtration* dont le principe est décrit dans la section 4.1.3. Le chapitre présent traite de la complexité de problèmes posés par l'optimisation de cette phase.

4.1.1.2 Importance théorique de la recherche de motifs approchés

Le problème de la recherche de motifs approchés est une variante simplifiée du problème de la recherche d'homologies locales. Il a été très largement étudié d'un point de vue théorique [89, 25]. Il a donné lieu à la fabrication de nombreux algorithmes toujours plus astucieux et plus rapides (sur le papier). Décrivons ce

problème. Soient un mot x appelé *motif* et un mot y appelé *texte*. Pour fixer les idées on peut toujours supposer $|x| \ll |y|$. On cherche la position de chacune des sous-chaînes de y qui sont égales ou approximativement égales à x.

4.1.2 Définitions formelles et algorithmes

Les logiciels cités dans la section 4.1.1.1 sont fabriqués en combinant des algorithmes de recherche de motifs et/ou d'homologies locales avec des heuristiques de manière à diminuer les temps de calculs. Dans cette section nous présentons les performances théoriques de ces algorithmes.

Dans tout le reste de la section 4.1,

- Σ désigne un alphabet quelconque, éventuellement infini,
- $d : \Sigma^\star \times \Sigma^\star \to \mathbb{Z} \cup \{\infty\}$ est une application quelconque, appelée *mesure de dissimilarité* et
- e désigne un entier appelé *seuil de dissimilarité*.

On dit que deux mots u et v sur Σ sont *similaires (pour d, au seuil e)* lorsqu'on a $d(u, v) \leq e$.

Soient x_1, $x_2 \in \Sigma^\star$. on appelle *homologie locale (pour d, au seuil e)* entre x_1 et x_2 tout quadruplet d'entiers (i_1, j_1, i_2, j_2) avec $0 \leq i_h \leq j_h \leq |x_h|$ pour chaque $h \in \{1, 2\}$, tel que $x_1(i_1\,;\,j_1]$ et $x_2(i_2\,;\,j_2]$ soient similaires.

Soient x, $y \in \Sigma^\star$. On appelle *occurrence approchée (pour d, au seuil e)* de x dans y tout couple d'entiers (i, j) avec avec $0 \leq i \leq j \leq |y|$, tel que x et $y(i\,;\,j]$ soient similaires (pour d au seuil e).

4.1.2.1 Algorithmes identifiant des homologies

Décider si deux mots sont similaires. En général, on peut décider si deux mots u et v sur Σ sont similaires pour d au seuil e, en calculant $d(u, v)$ et en comparant la quantité obtenue avec e. Dans le cas où d est la distance de Levenshtein d_{L}, il est possible de faire mieux : étant donnés deux mots u, v quelconques et un entier e, on peut décider si u et v sont similaires pour d_{L} au seuil e en temps $O((|u| + |v|)e)$ [88]. Ceci est vrai en dépit du fait que le calcul de $d_{\mathrm{L}}(u, v)$ prend un temps $O(|u|\,|v|)$ (ou presque [84]).

Meilleure homologie locale. La "version optimisation" du problème de la recherche d'homologies locales a été étudiée. Ce problème, dit de la meilleure homologie locale pour d, s'énonce de la manière suivante : "*Étant donnés deux mots x_1, $x_2 \in \Sigma^\star$, trouver un quadruplet d'entiers (i_1, j_1, i_2, j_2) avec $0 \leq i_h \leq j_h \leq |x_h|$ pour chaque $h \in \{1, 2\}$, tel que $d(x_1(i_1\,;\,j_1]\,, x_2(i_2\,;\,j_2])$ soit minimum*".

Soit une fonction de pondération $\pi : (\Sigma \cup \{\varepsilon\}) \times (\Sigma \cup \{\varepsilon\}) \to \mathbb{Z}$ calculable en temps constant. Le problème de la meilleure homologie locale pour d^π_{E} admet un algorithme exact de complexité quadratique $O(|x_1|\,|x_2|)$ [106, 45].

Choix de la fonction de pondération. Supposons que π soit à valeurs positives et considérons une instance (x_1, x_2) du problème de la meilleure homologie locale pour d^π_{E}. Alors, il suffit de prendre (i_1, j_1, i_2, j_2) avec $i_1 = j_1 \in [0, |x_1|]$ et $i_2 = j_2 \in [0, |x_2|]$ pour obtenir une solution optimale car $d^\pi_{\mathrm{E}}(x_1(i_1\,;\,j_1]\,, x_2(i_2\,;\,j_2]) = d^\pi_{\mathrm{E}}(\varepsilon, \varepsilon) = 0$.

En pratique, on privilégie la recherche de longues homologie locales (deux courts fragments de séquences peuvent n'être similaires que du fait du hasard). Il est donc naturel que π vérifie :

- $\pi(a, a) < 0$ pour tout $a \in \Sigma$ et
- $\pi(\alpha, \beta) > 0$ pour tous α, $\beta \in \Sigma \cup \{\varepsilon\}$ avec $\alpha \neq \beta$.

Dans ce contexte, les appariements exacts font strictement décroître le score des alignements alors que les différences (insertions, délétions et substitutions) le font augmenter.

4.1.2.2 Algorithmes de recherche de motifs approchés

Le problème de la recherche de motifs approchés pour d s'énonce : "*Étant donnés deux mots x, $y \in \Sigma^\star$ et un entier e, déterminer toutes les occurrences approchées pour d, au seuil e, de x dans y*". Ce problème a été étudié pour diverses fonctions d. Le tableau ci-dessous résume les résultats obtenus.

La mesure de dissimilarité est	La complexité est de
la distance “tout ou rien”	$O(\|y\|)$ [61]
la distance de Hamming	$O\big(\|y\|\sqrt{e\log e}\big)$ [3]
la distance de Levenshtein	$O(\|y\|\,e)$ pour Σ fini [69]
une distance d'édition pondérée	$O(\|y\|\,\|x\|)$ [45]

Rappelons que la distance “tout ou rien” vaut 0 ou ∞ suivant que ses arguments sont égaux ou non (voir section 1.1.1). Ainsi, lorsque d est la distance “tout ou rien”, le problème de la recherche de motifs approchés se ramène au problème de la recherche de motifs exacts : on cherche les positions des sous-chaînes de y égales à x. Notons également que l'on a convenu que $d_{\mathrm{H}}(u,v) = \infty$ si u et v sont de longueurs distinctes (voir section 1.1.3.3).

4.1.3 Filtration

Comme on vient de le voir, trouver une meilleure homologie locale entre deux mots donnés ou résoudre le problème de la recherche de motifs approchés pour une distance d'édition pondérée prend un temps quadratique. Or, la taille des banques de séquences biologiques croît exponentiellement avec le temps donc il est nécessaire de recourir à des heuristiques. La méthode la plus utilisée est la *recherche par filtration*. Elle est mise en œuvre par de nombreux logiciels, notamment par ceux cités dans la section 4.1.1.1, ainsi que par Tandem Repeat Finder dont la fonction est de rechercher des *répétitions en tandem* dans les séquences biologiques [11].

4.1.3.1 Principe général de la filtration

Plaçons nous dans le cadre général suivant : dans un *espace de recherche* E de grande taille, on cherche à identifier un maximum d'éléments appartenant à un petit sous-ensemble $S \subseteq E$. On appelle *candidats* les éléments de E et *solutions* les éléments de S. Par exemple, si l'on recherche les homologies locales entre deux mots x_1 et x_2 sur Σ, alors l'espace de recherche E est l'ensemble des quadruplets d'entiers (i_1, j_1, i_2, j_2) vérifiant $0 \leq i_h \leq j_h \leq |x_h|$ pour chaque $h \in \{1,2\}$. Les solutions sont les homologies locales entre x_1 et x_2. La cardinal de E est de l'ordre de $|x_1|^2\,|x_2|^2$, ce qui rend une exploration exhaustive impossible en temps raisonnable.

Filtre. On appellera *filtre* une heuristique rapide permettant de réduire l'espace de recherche E à un sous-ensemble F. Les éléments de F sont les *candidats retenus (par le filtre)* et les éléments de $E \setminus F$ sont appelé les *candidats rejetés (par le filtre)*. Un filtre est d'autant meilleur que l'ensemble F des candidats qu'il retient est peu différent de l'ensemble S des solutions recherchées.

Algorithmes de recherche par filtration. Grossièrement on peut associer à chaque filtre l'algorithme de recherche suivant.

1. *Filtration* : À l'aide du filtre on retient un petit sous-ensemble F des candidats de l'espace de recherche E.
2. *Validation des candidats retenus* : On calcule $S \cap F$ en explorant l'espace F des candidats retenus par le filtre et on retourne les éléments de $S \cap F$.

4.1.3.2 Filtration et recherche d'homologies locales

Soient x_1, $x_2 \in \Sigma^\star$. On cherche des homologies locales entre x_1 et x_2.

Méthodes de filtrations courantes. En général les filtres utilisés par les logiciels dédiés à la recherche d'homologies locales retiennent les candidats $(i_1, j_1, i_2, j_2) \in E$ tels que $x_1(i_1\,;j_1]$ et $x_2(i_2\,;j_2]$ partagent un certain nombre de (petites) sous-chaînes communes. Autrement dit, on soupçonne la similarité de deux portions de séquences lorsqu'elles sont exactement semblables au moins en un petit nombre d'endroits.

L'exemple du logiciel BLAST. Fixons un entier $q \geq 0$ et expliquons le principe de la méthode utilisée par le logiciel BLASTN pour identifier des homologies locales entre deux séquences d'ADN représentées par les mots x_1 et x_2.

Pour BLASTN on a $q = 11$ et $\Sigma = \{\mathtt{A}, \mathtt{T}, \mathtt{G}, \mathtt{C}\}$. Le filtre utilisé par ce logiciel est construit à partir du critère de similarité grossier suivant : pour que deux mots soient similaires, il faut qu'ils aient (au moins) une sous-chaîne commune de longueur (au moins) q.

On appelle *touche (de longueur q dans (x_1, x_2))* tout couple $(t_1, t_2) \in [0, |x_1| - q] \times [0, |x_2| - q]$ tel que $x_1(t_1\,;t_1 + q] = x_2(t_2\,;t_2 + q]$. Les touches repèrent les positions des sous-chaînes de longueur q communes à x_1 et x_2.

Exemple 4.1 *Supposons que* $x_1 = \mathtt{abcdpifef}$ *et que* $x_2 = \mathtt{fpifde}$. *Alors,* $(4, 1)$ *est une touche de longueur* 3 *dans* (x_1, x_2) *car* $x_1(4\,;4 + 3] = \mathtt{pif} = x_2(1\,;1 + 3]$.

Utilisons les touches de longueur q pour filtrer.

1. On calcule l'ensemble H de toutes les touches de longueur q dans (x_1, x_2). En théorie, cette étape est implantable en temps linéaire à l'aide de structures d'index adéquates. En pratique on utilise des tables de hachages, moins gourmandes en mémoire.
2. On *étend* chaque touche $(t_1, t_2) \in H$. Informellement, cela signifie que l'on cherche une homologie locale entre x_1 et x_2 autour de la touche (t_1, t_2). Plus précisément, on cherche un élément $(i_1, j_1, i_2, j_2) \in S$ avec $i_h \leq t_h \leq t_h + q \leq j_h$ pour chaque $h \in \{1, 2\}$.

Notons que l'étape 1 réduit l'espace recherche E à l'ensemble F des $(i_1, j_1, i_2, j_2) \in E$ tels que $x_1(i_1\,;j_1]$ et $x_2(i_2\,;j_2]$ aient a une sous-chaîne commune de longueur q.

4.1.3.3 Sensibilité et sélectivité

Performances d'un filtre. Du fait de son imperfection, un filtre partitionne l'espace de recherche en quatre classes.

1. Un *vrai positif* est un élément de $S \cap F$, c'est-à-dire un candidat retenu à raison.
2. Un *vrai négatif* est un élément de $(E \setminus S) \setminus F$, c'est-à-dire un candidat rejeté à raison.
3. Un *faux positif* est un élément de $(E \setminus S) \cap F$, c'est-à-dire un candidat retenu à tort.
4. Un *faux négatif* est un élément de $S \setminus F$, c'est-à-dire un candidat rejeté à tort.

• Les vrais positifs sont les solutions retournées par l'algorithme de recherche associé au filtre alors que les faux négatifs sont les solutions qu'il omet de retourner. Ainsi, la qualité de la sortie diminue quand le nombre de faux négatifs augmente. Ceci pousse à définir la *sensibilité* du filtre comme la valeur du rapport :

$$\frac{\text{nombre de vrais positifs}}{\text{nombre de vrais positifs} + \text{nombre de faux négatifs}}\,.$$

Un filtre de sensibilité 1, c'est-à-dire un filtre n'engendrant pas de faux négatifs, est dit *sans perte* : la sortie de l'algorithme de recherche associé est parfaite.

• Les faux positifs ralentissent inutilement la phase de validation de l'algorithme de recherche associé au filtre. En effet, ces candidats, initialement retenus par le filtre, seront finalement exclus après examen. Ceci nous pousse à définir la *sélectivité* du filtre comme la valeur du rapport :

$$\frac{\text{nombre de vrais négatifs}}{\text{nombre de vrais négatifs} + \text{nombre de faux positifs}}\,.$$

La sensibilité et la sélectivité sont des propriétés antagonistes : on est toujours obligé de faire un compromis entre la qualité de la sortie et le temps de calcul.

Ajustement des performances de la filtration de type BLAST. Replaçons nous dans le cas où l'on cherche des homologies locales entre deux mots x_1 et x_2 sur Σ. Examinons le filtre par sous-chaînes communes décrit dans la section 4.1.3.2. On restreint l'espace de recherche E à l'ensemble F des $(i_1, j_1, i_2, j_2) \in E$ tels que $x_1(i_1 ; j_1]$ et $x_2(i_2 ; j_2]$ aient une sous-chaîne commune de longueur q.

Exemple 4.2 *On suppose que* $\Sigma = \{\mathtt{a}, \mathtt{X}\}$*, que* $e = 1$ *et que* d *est la distance Hamming.*
Soit $(i_1, j_1, i_2, j_2) \in E$ *tel que* $x_1(i_1 ; j_1] = \mathtt{aaaaaaa}$.
Si $x_2(i_2 ; j_2] = \mathtt{aXXaaaa}$ *et* $q = 4$ *alors :*
– $(i_1, j_1, i_2, j_2) \notin S$ *car* $d_{\mathrm{H}}(x_1(i_1 ; j_1], x_2(i_2 ; j_2]) = 2$ *et*
– $(i_1, j_1, i_2, j_2) \in F$ *car* $x_1(i_1 ; j_1]$ *et* $x_2(i_2 ; j_2]$ *admettent* `aaaa` *pour sous-chaîne commune de longueur* q.
Ainsi, (i_1, j_1, i_2, j_2) *est un élément de* $(E \setminus S) \cap F$*, c'est-à-dire un faux positif.*

Dans les trois premières colonnes du tableau suivant, on trouve le statut de (i_1, j_1, i_2, j_2) *pour chaque* $q \in \{3, 4\}$ *et diverses valeurs de* $x_1(i_2 ; j_2]$.

$x_2(i_2 ; j_2]$	$q = 3$	$q = 4$	`##-##`
`aaXaaaa`	*vrai positif*	*vrai positif*	*vrai positif*
`aaaXaaa`	*vrai positif*	*faux négatif*	*vrai positif*
`aaaaaXX`	*faux positif*	*faux positif*	*faux positif*
`XaaXaaa`	*faux positif*	*vrai négatif*	*faux positif*
`aaXaaXa`	*vrai négatif*	*vrai négatif*	*faux positif*
`aXXaaaa`	*faux positif*	*faux positif*	*vrai négatif*
`aaaXXaa`	*faux positif*	*vrai négatif*	*vrai négatif*
`aaXaXaa`	*vrai négatif*	*vrai négatif*	*vrai négatif*

Expliquons en quoi la longueur q des touches influe sur la sensibilité et sur la sélectivité du filtre.

• Si q diminue alors le nombre de touche augmente et l'ensemble des candidats retenus grossit. Par suite, le nombre de vrais positifs augmente et le nombre de faux négatifs diminue : la sensibilité augmente.

• Réciproquement, si q augmente alors le nombre de touches diminue et l'ensemble des candidats aussi. Par suite, le nombre vrais négatifs augmente et le nombre de faux positifs diminue : la sélectivité augmente. En pratique, on peut supposer que les touches engendrant des faux positifs (c'est-à-dire les touches autour desquelles, on ne trouve pas d'homologie) sont dues au hasard. Ainsi, augmenter q de 1 divise approximativement le nombre de ces touches par $\#\Sigma$ [37].

4.2 Position des problèmes

Comme l'ont remarqué Ma, Tromp et Li [79], il suffit de modifier légèrement le filtre utilisé par le logiciel BLASTN (voir section 4.1.3.2) pour le rendre à la fois plus sensible et plus sélectif. L'idée mise en œuvre dans le logiciel PatternHunter est d'augmenter la longueur des touches (elle passe de 11 à 18) en leur ajoutant quelques "trous" (7 en l'occurrence) à des endroits judicieux. Les résultats nouveaux démontrés dans ce chapitre concernent la complexité algorithmique de trois problèmes liés au bon positionnement de ces "trous" (on parle aussi de "jokers"). Ces trois problèmes sont présentés, avec leur contextes, dans la section 4.2.2. Auparavant, nous avons besoin de quelques définitions formelles supplémentaires.

4.2.1 Graine, similarité, détection

Dans les sections 4.2.1.1 et 4.2.1.2 ci-après, nous introduisons les trois principaux concepts (graine, similarité et détection) étudiés tout au long du chapitre présent. Nous motivons ensuite leur introduction dans les sections 4.2.1.3 à 4.2.1.6 suivantes, en particulier l'utilité des trous.

4.2.1.1 Graine, similarité, détection

Graine. Soit $w \in \{\texttt{\#}, \texttt{-}\}^\star$. Le *poids* de w, noté $\|w\|$, est le nombre d'occurrences de `#` dans w : $\|w\| = |w|_{\texttt{\#}}$. La lettre `-`, appelée *trou*, ne contribue pas au poids. Une *graine* est un mot sur $\{\texttt{\#}, \texttt{-}\}$ dont la première et la dernière lettre sont des `#`, c'est-à-dire un élément de $\texttt{\#}\{\texttt{\#}, \texttt{-}\}^\star\texttt{\#} \cup \{\texttt{\#}, \varepsilon\}$.

Exemple 4.3 $g := \texttt{\#-\#\#--\#-\#\#}$ *est une graine de poids* 6.

Similarité. Une *similarité* est un mot sur $\{\texttt{0}, \texttt{1}\}$, c'est-à-dire un élément de $\{\texttt{0}, \texttt{1}\}^\star$. Soient deux entiers m, k avec $0 \leq k \leq m$. Une (m, k)-*similarité* est une similarité de longueur m dans laquelle apparaissent k occurrences de la lettre `0` et $m - k$ occurrences de la lettre `1`. Autrement dit, une (m, k)-similarité est un élément de $\{s \in \{\texttt{0}, \texttt{1}\}^m : |s|_{\texttt{0}} = k\}$.

Exemple 4.4 $s := \texttt{101101101101100}$ *est une* $(15, 6)$*-similarité.*

Détection. Soient s une similarité, $w \in \{\texttt{\#}, \texttt{-}\}^\star$ et $p \in [0, |s| - |w|]$. On dit que w *détecte* s *à la position* p lorsque pour tout $i \in [1, |w|]$ tel que $w[i] = \texttt{\#}$, on a $s[p + i] = \texttt{1}$. On dit que w *détecte* s lorsqu'il existe $p \in [0, |s| - |w|]$ tel que w détecte s à la position p. Soit $W \subseteq \{\texttt{\#}, \texttt{-}\}^\star$. On dit que W détecte s lorsqu'il existe $w \in W$ tel que w détecte s.

Exemple 4.5 (Suite des exemples 4.3 et 4.4) *La graine* g *de l'exemple 4.3 p. 91 détecte la similarité* s *de l'exemple 4.4 p. 91 aux positions* 0 *et* 3*. Le mot* $\texttt{\#}(\texttt{-})^{11}\texttt{\#}(\texttt{-})^2$ *détecte* s *uniquement à la position* 0*.*

4.2.1.2 Observations en vrac

Exemple 4.6 *Soit* s *une similarité quelconque dont on note* m *la longueur.*
- *La graine* `#` *détecte* s *aux positions* $i \in [0, m-1]$ *telles que* $s[i+1] = \texttt{1}$*.*
- *Pour tout* $j \in [1, m]$ *tel que* $s[i] = \texttt{1}$*, le mot* $(\texttt{-})^{j-1}\texttt{\#}(\texttt{-})^{m-j}$ *détecte* s *à la position* 0*.*
- *Pour tout entier* $j \geq 0$*, le mot* $(\texttt{-})^j$ *détecte* s *en toute position comprise entre* 0 *et* $m - j$*.*

Divers points des deux remarques suivantes seront exploités dans les preuves des résultats de ce chapitre.

Remarque 4.1 *Soient* $w \in \{\texttt{\#}, \texttt{-}\}^\star$ *et* $s \in \{\texttt{0}, \texttt{1}\}^\star$*.*

(*i*) *Pour que le mot* w *détecte la similarité* s*, il faut* $|w| \leq |s|$*.*

(*ii*) *Dans le cas particulier où* $|w| = |s|$*,* w *détecte* s *si et seulement si* w *détecte* s *à la position* 0*.*

(*iii*) *Si* w *détecte* s *alors toute sous-chaîne de* w *détecte* s*.*

(*iv*) *Si* w *détecte une sous-chaîne de* s *alors* w *détecte* s*.*

Remarque 4.2 *Soit un entier* $m \geq 0$*.*

(*i*) *Un mot sur* $\{\texttt{\#}, \texttt{-}\}$ *détecte la similarité* $\texttt{1}^m$ *si et seulement s'il est de longueur au plus* m*.*

(*ii*) *Un mot sur* $\{\texttt{\#}, \texttt{-}\}$ *détecte la similarité* $\texttt{0}^m$ *si et seulement s'il est de la forme* $(\texttt{-})^j$ *avec* $j \in [0, m]$*.*

(*iii*) *La graine vide détecte toutes les similarités.*

4.2.1.3 Touche à gabarit et filtration

Les trous des graines permettent de généraliser la notion de touche définie dans la section 4.1.3.2. Soit g une graine et soient x_1 et x_2 deux mots quelconques. On appelle *touche de gabarit* g *(dans* (x_1, x_2)*)* tout couple $(t_1, t_2) \in [0, |x_1| - |g|] \times [0, |x_2| - |g|]$ tel que :

$$\forall i \in [1, |g|] \qquad g[i] = \texttt{\#} \Rightarrow x_1[t_1 + i] = x_2[t_2 + i]\,.$$

D'après cette définition, une touche de longueur q est également une touche de gabarit $\texttt{\#}^q$ (sans trou).

Exemple 4.7 *Reprenons la graine* $g =$ `#-##--#-##` *de l'exemple 4.3 p. 91 et posons* $x_1 :=$ `apaifpfpaafii` *et* $x_2 :=$ `ipifpfifpppafa`. *Alors* $(1, 4)$ *est une touche de gabarit* g *dans* (x_1, x_2) *comme le montre de le diagramme suivant :*

```
          a p a i f p f p a a f i i
            p - i f - - p - a f       .
i p i f p f i f p p p p a f a
```

La méthode mise en œuvre par le logiciel PatternHunter généralise celle de BLASTN : au lieu de rechercher des homologies locales autour des touches de longueur 11, PatternHunter les recherche autour des touches de gabarit g_{PH}, où g_{PH} est un graine, par défaut égale à `###-#--#-#--##-###`.

La sélectivité du filtre augmente clairement avec le poids de g_{PH} (mais aussi un peu avec sa longueur [79]). À la fin de la section 4.2.1.5, nous justifions par un exemple en quoi le bon usage des trous permet également d'augmenter la sensibilité.

4.2.1.4 Rôle des similarités

Une similarité modélise la manière dont deux mots de même longueur peuvent différer. Soient deux mots u et v de même longueur m. On appelle *similarité entre* u *et* v la similarité s de longueur m donnée par :

$$\forall i \in [1, m] \qquad s[i] = \begin{cases} \mathtt{1} & \text{si } u[i] = v[i] \\ \mathtt{0} & \text{sinon} \end{cases} .$$

Exemple 4.8 *La similarité entre* $u :=$ `ACCGATG` *et* $v :=$ `ATCGGTC` *est* `1011010`*, comme le montre le diagramme suivant :*

```
u = A C C G A T G
v = A T C G G T C .
    1 0 1 1 0 1 0
```

Les (m, k)-similarités modélisent la manière dont deux mots de longueur m séparés par une distance de Hamming égale à k peuvent différer.

Le concept de similarité permet de gagner un niveau d'abstraction : à partir de la section 4.2.2 nous n'auront plus à considérer directement les mots que l'on cherche à comparer mais seulement leurs similarités.

4.2.1.5 Rôle du concept de détection

Dans le cadre d'une filtration du type de celle utilisée par BLAST ou PatternHunter, il est préférable que les couples de mots de longueur m séparés par une distance de Hamming $k \ll m$ provoquent une touche. Sinon, on augmente le risque d'engendrer des faux négatifs.

Proposition 4.1
Soient deux mots u, v *de même longueur* m, *une graine* g *et* $p \in [0, m - |g|]$.

Alors, g *détecte la similarité entre* u *et* v *à la position* p *si et seulement si* (p, p) *est une touche de gabarit* g *dans* (u, v).

Ainsi, le nombre de touches de gabarit g croît avec le nombre de (m, k)-similarités détectés par g. Par suite, pour m et k bien choisis, la sensibilité du filtre fabriqué à partir de g croît avec le nombre de (m, k)-similarités détectées par g.

4.2.1.6 Utilité des graines à trous

Plus y a de gruyère, plus y a de trous.
Mais plus y a de trous, moins y a de gruyère.
Coluche (?)

En résumé, on a vu que :

1. la sélectivité du filtre fabriqué à partir d'une graine pour rechercher des homologies augmente avec son poids et que
2. la sensibilité de ce même filtre augmente avec le nombre de (m, k)-similarités détectées.

On peut maintenant justifier par l'exemple l'utilité des graines à trous.

Exemple 4.9 ([37]) *Soit un entier $m \geq 0$. Par commodité on va supposer que 6 divise $m - 1$.*
- *La graine sans trou de poids maximum détectant toutes les $(m, 1)$-similarités est $\#^{(m-1)/2}$.*
- *La graine à un trou $\#^{(m-1)/3}\texttt{-}\#^{(m-1)/3}$, détecte également toutes les $(m, 1)$-similarités et a un poids de $2(m - 1) / 3$.*

Dans le cas particulier où $m = 7$, le comportement des graines $\#^{(m-1)/2} = \texttt{###}$, $\texttt{####}$ et $\#^{(m-1)/3}\texttt{-}\#^{(m-1)/3} = \texttt{\#\#-\#\#}$ est abordé dans le tableau de l'exemple 4.2 p. 90.

4.2.2 Description de nos résultats

Le but du chapitre en cours est d'étudier la complexité et l'approximabilité de trois problèmes liés à la conception de graines. Chacune des sections 4.2.2.1, 4.2.2.2 et 4.2.2.3 ci-après est consacrée à la présentation de l'un ces problèmes : nous énonçons les résultats déjà connus ainsi que notre propre contribution.

4.2.2.1 Le problème Non Detection

Le problème Non Detection s'énonce de la manière suivante : "*Étant donnés une graine g et deux entiers m, k vérifiant $0 \leq k \leq m$, décider s'il existe une (m, k)-similarité non détectée par g*". Ce problème est introduit formellement par Kucherov, Noé et Roytberg [65]. Nous montrons dans la section 4.3 que Non Detection est NP-complet, en supposant que, dans toute instance (g, m, k) de Non Detection, les entiers m et k sont codés en unaire. Ainsi, l'instance (g, m, k) est de taille $O(|g| + m)$ et donc nous montrons en fait que Non Detection est *fortement* NP-complet[1]. Ce résultat assure que les algorithmes exponentiels résolvant les deux problèmes décrits ci-dessous sont en quelque sorte optimaux.

Deux caractéristiques importantes pour un ensemble de graine. Supposons donnés deux entiers m, k vérifiant $0 \leq k \leq m$ et un ensemble de graines G vérifiant $|g| \leq m$ pour tout $g \in G$.

1. Notons $U(G, m, k)$ le nombre de (m, k)-similarités non détectées par G.
2. Notons $T(G, m, k)$ le plus grand entier $t \geq 0$ tel que G détecte toute (m, k)-similarité en t positions.

Définissons précisément cette dernière notion. Étant donnés un entier $t \geq 0$ et une similarité s, on dit que *G détecte s en t positions* lorsqu'il existe t paires distinctes (g_1, p_1), (g_2, p_2), $\ldots$, (g_t, p_t) vérifiant, pour tout $i \in [1, t]$, $g_i \in G$, $p_i \in [0, |s| - |g_i|]$ et g_i détecte s à la position p_i.

Algorithmes. Il existe des algorithmes de programmation dynamique calculant $U(G, m, k)$ et $T(G, m, k)$ en temps au plus de l'ordre de

$$m \times \sum_{k'=0}^{k} \binom{M(G)}{k'}(k - k' + 1) + (\#G) \times \sum_{k'=0}^{k} \binom{M(G)}{k'} \tag{4.1}$$

où $M(G) = \max_{g \in G} |g|$ [66, 19]. Donnons une idée du fonctionnement de ces algorithmes.

Pour tous entiers m', k' avec $0 \leq k' \leq m'$ et toute similarité s, on appelle (m', k', s)-similarité toute (m', k')-similarité admettant s pour suffixe. On note

1. $U'_G[m'][k'][s]$ le nombre de (m', k', s)-similarités non détectées par G et
2. $T'_G[m'][k'][s]$ le plus grand entier $t \geq 0$ tel que G détecte toute (m', k', s)-similarité en t positions.

[1] La notion de NP-*complétude au sens fort* est compliquée (impossible ?) à définir formellement. On trouvera un commentaire intéressant à ce sujet dans le livre de Papadimitriou [94, note 9.5.31].

Une (m, k, ε)-similarité est une (m, k)-similarité et réciproquement. Il en résulte que les deux quantités que l'on cherche à calculer sont des entrées des tables définies ci-dessus : $U(G, m, k) = U'_G[m][k][\varepsilon]$ et $T(G, m, k) = T'_G[m][k][\varepsilon]$.

Les algorithmes de Kucherov, Noé et Roytberg [66] calculent à l'aide d'une relation de récurrence convenable $U'_G[m'][k'][s]$ (resp. $T'_G[m'][k'][s]$) pour tout triplet (m', k', s) avec

- $m' \in [0, m]$ (resp. $m' \in [M(G), m]$),
- $k' \in [0, k]$ et
- $s \in \{\mathtt{0}, \mathtt{1}\}^\star$ avec $|s| \leq M(G)$ et $|s|_{\mathtt{0}} \leq k'$.

Liens avec le problème Non Detection. Résoudre le problème Non Detection sur une instance (g, m, k) donnée revient à décider si $U(\{g\}, m, k)$ est strictement positif ou si $T(\{g\}, m, k)$ est nul. Comme Non Detection est NP-complet (théorème 4.2 p. 100), il est peu probable que l'on puisse trouver un algorithme calculant l'une des quantités $U(G, m, k)$ ou $T(G, m, k)$ en temps sous-exponentiel.

Par ailleurs, l'inégalité

$$\sum_{k'=0}^{k} \tbinom{M(G)}{k'} \leq 2^{M(G)}$$

permet de majorer la quantité (4.1) : $U(G, m, k)$ et $T(G, m, k)$ sont calculables en temps exponentiel $O\big(2^{M(G)}(mk + \#G)\big)$. Ainsi, Non Detection est F.P.T. pour le paramètre $|g|$ $(= M(\{g\}))$.

4.2.2.2 Le problème Maximum Weight Lossless Seed (MWLS)

Introduisons un nouveau problème naturel concernant la conception de graines :

Définition 4.1
On appelle Maximum Weight Lossless Seed *(*MWLS*) le problème de maximisation suivant :* "Étant donné un ensemble S de similarités, trouver une graine de poids maximum détectant toutes les similarités appartenant à S".

En quelque sorte, résoudre MWLS consiste à rechercher un graine de sélectivité maximale dont le filtre associé est sans perte. Dans la section 4.4 nous montrons que MWLS est au moins aussi difficile à approximer que 2-MISH. En particulier, MWLS n'admet pas d'algorithme polynomial exact, sauf si NP = P.

Notons qu'une variante du problème de décision associé à MWLS a été montrée NP-complète [71] : "*Étant donnés un ensemble S de similarités et deux entiers ℓ et ϖ avec $1 \leq \varpi \leq \ell$, décider s'il existe une graine de longueur ℓ et de poids ϖ détectant toutes les similarités appartenant à S*". Il semble que le paramètre de longueur ℓ soit artificiel. De notre point de vue, ce paramètre n'a été introduit que pour faciliter la preuve de NP-difficulté du problème considéré. Dans la définition de MWLS, nous avons délibérément omis toute contrainte sur la longueur de la graine recherchée.

4.2.2.3 Le problème Region Specific Optimal Seed(s) (RSOS(s))

Considérons le problème Region Specific Optimal Seeds (RSOSs), introduit par Li, Ma, Kisman et Tromp [71] : "*On suppose donnés trois entiers naturels d, ℓ, ϖ avec $1 \leq \varpi \leq \ell$ et un ensemble S de similarités. On cherche d graines g_1, g_2, ..., g_d, chacune de poids ϖ et de longueur au plus ℓ, maximisant le nombre des similarités appartenant à S et détectées par $\{g_1, g_2, \ldots, g_d\}$*". Deux résultats sont connus sur la complexité de ce problème [71].

1. Le problème RSOSs est au moins aussi difficile à approximer que Max k-Cover. (La définition et les principaux résultats connus sur l'approximabilité de ce dernier problème sont rappelés dans la section 4.5.1.) Ainsi, sous l'hypothèse P $\subsetneq$ NP, RSOSs n'admet pas d'algorithme d'approximation de borne constante ρ, quel que soit le réel ρ vérifiant $1 \leq \rho < \frac{e}{e-1}$. Nous désignons ici par e base du logarithme népérien : $e \simeq 2.718$ et $\frac{e}{e-1} \simeq 1.582$.
2. Le problème de décision associé à RSOSs reste NP-difficile même si l'on omet de borner la longueur des graines recherchées, c'est-à-dire même si l'on restreint RSOSs aux instances (d, ℓ, ϖ, S) vérifiant $\ell \geq \max_{s \in S} |s|$.

Nous améliorons ces deux résultats dans la section 4.5.

Nous considérons la restriction suivante de RSOSs, notée RSOS : "*On suppose donnés un entier $\varpi \geq 1$ et un ensemble S de similarités. On cherche une graine de poids ϖ détectant un maximum de similarités appartenant à S*". Informellement, résoudre RSOS consiste à rechercher une graine de sensibilité maximale avec une sélectivité donnée. Notons que RSOS correspond à la restriction de RSOSs aux instances (d, ℓ, ϖ, S) vérifiant $d = 1$ et $\ell \geq \max_{s \in S} |s|$: dans RSOS on ne cherche à construire qu'une seule graine dont la longueur n'est pas contrainte.

Nous montrons (théorème 4.4 p. 110) que, sous l'hypothèse P $\subsetneq$ NP, il n'existe pas d'algorithme d'approximation pour RSOS de borne constante ρ, quel que soit le réel ρ vérifiant $1 \leq \rho < \frac{e}{e-1}$.

4.3 Difficulté du problème Non Detection

Dans cette section, nous montrons que le problème Non Detection, défini dans la section 4.2.2.1, est NP-complet (théorème 4.2 p. 100). Pour cela, nous introduisons le problème intermédiaire suivant.

Définition 4.2 (Soapy Set Cover)
On nomme Soapy Set Cover *(*SSC*) le problème de décision suivant :* "On suppose donnés un ensemble fini $\mathcal{P}$ de parties finies et non vides de $\mathbb{Z}$ et deux entiers n, $k \geq 1$. Existe-t-il X_1, X_2, ..., $X_k \in \mathcal{P}$ et t_1, t_2, ..., $t_k \in \mathbb{Z}$ tels que $(X_1 + t_1) \cup (X_2 + t_2) \cup \ldots \cup (X_k + t_k)$ contienne n entiers consécutifs ?".

On pourra comparer SSC avec le problème MSC_D défini dans l'exemple 1.16 p. 14 ainsi qu'avec les problèmes de pavages définis dans la section 4.6.1. On supposera que tous les entiers intervenant dans une instance $(\mathcal{P}, n, k)$ de SSC, c'est-à-dire n, k et les éléments des éléments de $\mathcal{P}$, sont codés en unaire.

Nous prouvons que Non Detection est NP-difficile en trois temps.

1. Dans la preuve du théorème 4.1 p. 95, nous réduisons à SSC le problème Exact Cover by 3-Sets. Ce dernier problème est notoirement NP-complet [40] et sa définition est rappelée ci-dessous.
2. Dans la preuve du corollaire 4.1 p. 98, nous réduisons SSC à sa restriction aux instances $(\mathcal{P}, n, k)$ telles que $\#\mathcal{P} = 1$.
3. Dans la preuve du théorème 4.2 p. 100, nous réduisons cette restriction de SSC à Non Detection.

Notons que dans l'article où la NP-complétude de Non Detection est prouvée pour la première fois [91], les étapes 1 et 2 sont contenues dans la preuve d'un seul et même théorème.

Rappelons l'énoncé du problème Exact Cover : "*Étant donné un hypergraphe $H = (V, \mathcal{E})$, décider si $\mathcal{E}$ admet pour sous-ensemble une partition de V*". Le problème Exact Cover by 3-Sets (X3C en abrégé) est la restriction de Exact Cover aux instances H qui sont des hypergraphes 3-uniformes.

Théorème 4.1
Le problème SSC *est* NP*-complet.*

Preuve. Montrons tout d'abord que SSC est dans NP.

On construit facilement un algorithme polynomial C, retournant `oui` si et seulement si son entrée est de la forme

$$((\mathcal{P}, n, k), (X_1, X_2, \ldots, X_k, t_1, t_2, \ldots, t_k))$$

avec $(\mathcal{P}, n, k)$ instance de SSC, X_1, X_2, ... $X_k \in \mathcal{P}$, et t_1, t_2, ... $t_k \in \mathbb{Z}$ vérifiant :

$$[1, n] \subseteq \bigcup_{h=1}^{k} (X_h + t_h)\,. \tag{4.2}$$

Notons que, si $(\mathcal{P}, n, k)$ est une instance positive de SSC, alors l'équation (4.2) est vérifiée pour un certain objet $(X_1, X_2, \ldots, X_k, t_1, t_2, \ldots, t_k)$. De plus, pour chaque $h \in [1, k]$, on peut toujours supposer $1 - \max X_h \leq t_h \leq n - \min X_h$ car sinon, $X_h + t_h$ n'intersecte pas $[1, n]$. En faisant ces hypothèses, on rend $(X_1, X_2, \ldots, X_k, t_1, t_2, \ldots, t_k)$ de taille polynomiale en la taille de $(\mathcal{P}, n, k)$.

On déduit des arguments ci-dessus que C est un algorithme de validation polynomial pour SSC et donc que SSC est dans NP. Il reste à montrer que SSC est NP-difficile. Pour cela, nous exhibons une réduction de Karp de X3C à SSC.

Construction d'une instance de Non Detection à partir d'une instance de X3C. Supposons donnée une instance $H = (V, \mathcal{E})$ de X3C : H est un hypergraphe 3-uniforme. On construit une instance $(\mathcal{P}, n, k)$ de SSC de la manière suivante.

Posons :

$$k := \frac{\#V}{3} \qquad \text{et} \qquad n := 2k^2 + 4k\,.$$

Si 3 ne divise pas le cardinal de V ou si $\mathcal{E}$ est de cardinal inférieur à k alors H est une instance négative de X3C qu'il suffit de transformer en une instance négative triviale de SSC, par exemple $(\{\{0\}\}, 2, 1)$. On peut donc désormais supposer que k et n sont entiers, que $\mathcal{E}$ est de cardinal au moins k et que

$$V = [k+1, 4k]\ ,$$

quitte à numéroter de $k+1$ à $4k$ les $3k$ éléments de V. Numérotons également les éléments de $\mathcal{E}$:

$$m := \#\mathcal{E} \qquad \text{et} \qquad \mathcal{E} = \{E_1, E_2, \ldots, E_m\}\,.$$

Posons :

- $F_j := [(j-1)(2k-1) + 4k + 1, j(2k-1) + 4k]$ pour tout $j \in [1, k]$,
- $X_{i,j} := \{j\} \cup E_i \cup F_j \cup \{n - j + 1\}$ pour tout $(i, j) \in [1, m] \times [1, k]$ et
- $\mathcal{P} := \{X_{i,j} : (i, j) \in [1, m] \times [1, k]\}$.

Notre réduction consiste en la transformation de l'instance H de X3C en l'instance $(\mathcal{P}, n, k)$ de SSC définie ci-dessus.

Décrivons succinctement la forme de notre gadget. Posons $W := \big[4k + 1, 2k^2 + 3k\big]$. Tout d'abord, $\{F_1, F_2, \ldots, F_k\}$ est l'unique partition de W en segments de longueur $2k - 1$. De plus, les quatre segments (discrets)

$$[1, k]\ , \quad V = [k+1, 4k]\ , \quad W = [4k+1, n-k]\ , \quad [n-k+1, n]$$

sont de longueurs k, $3k$, $2k^2 - k$ et k, respectivement. Ils forment une partition de $[1, n]$. Chacun contribue à la fabrication de $X_{i,j}$. En effet, on a :

$$\{j\} \subseteq [1, k] \quad E_i \subseteq V\,, \quad F_j \subseteq W\,, \quad \{n - j + 1\} \subseteq [n - k + 1, n]\ .$$

Plus schématiquement, on peut représenter ces inclusions de la manière suivante :

$$\begin{array}{ccccccccc} [1,n] & = & [1,k] & \cup & V & \cup & W & \cup & [n-k+1,n] \\ X_{i,j} & = & \{j\} & \cup & E_i & \cup & F_j & \cup & \{n-j+1\} \end{array}\ ,$$

chacun des cinq ensembles apparaissant sur la ligne inférieure étant une partie de l'ensemble situé immédiatement au dessus de lui.

Montrons que notre transformation est bien une réduction de Karp de X3C à SSC. Comme on s'est ramené au cas où $\mathcal{E}$ est de cardinal au moins k, la taille de H est au moins k. Par suite $(\mathcal{P}, n, k)$, qui est de taille polynomiale en k, est de taille polynomiale en la taille de H. On vérifie de plus facilement que l'on peut construire $(\mathcal{P}, n, k)$ à partir de H en temps polynomial. Donc, il ne reste plus qu'à vérifier que $(\mathcal{P}, n, k)$ est une instance positive de SSC si et seulement si H est une instance positive de X3C.

• Supposons que H soit une instance positive de X3C.

Alors, il existe i_1, i_2, …, $i_k \in [1, m]$ tels que $\{E_{i_1}, E_{i_2}, \ldots, E_{i_k}\}$ soit une partition de V. Par suite, $\{X_{i_1,1}, X_{i_2,2}, \ldots, X_{i_k,k}\}$ est une partition de $[1, n]$. En effet, on a

$$\begin{aligned} \bigcup_{j=1}^{k} X_{i_j,j} &= \bigcup_{j=1}^{k} \{j\} \cup \bigcup_{j=1}^{k} E_{i_j} \cup \bigcup_{j=1}^{k} F_j \cup \bigcup_{j=1}^{k} \{n-j+1\} \\ &= [1,k] \cup V \cup W \cup [n-k+1, n] \\ &= [1, n] \end{aligned}$$

donc, posant $t_j := 0$ pour tout $j \in [1,k]$, on peut écrire :

$$[1,n] = \bigcup_{j=1}^{k} (X_{i_j,j} + t_j)\,.$$

Par conséquent, $(\mathcal{P}, n, k)$ est une instance positive de SSC, ce qu'on voulait à ce stade.

• Réciproquement, supposons que $(\mathcal{P}, n, k)$ soit une instance positive de SSC.
Alors, il existe i_1, i_2, …, $i_k \in [1,m]$, j_1, j_2, …, $j_k \in [1,k]$ et t_1, t_2, …, $t_k \in \mathbb{Z}$ vérifiant :

$$[1,n] \subseteq \bigcup_{h=1}^{k} (X_{i_h,j_h} + t_h)\,. \tag{4.3}$$

Si l'on arrive à montrer que

$$t_1 = t_2 = \cdots = t_k = 0 \tag{4.4}$$

alors, on pourra en déduire que les hyperarêtes E_{i_1}, E_{i_2}, …, E_{i_k} couvrent V. Comme ces k hyperarêtes sont chacune de cardinal 3 et comme V est de cardinal $3k$, on obtiendra que $\{E_{i_1}, E_{i_2}, \ldots, E_{i_k}\}$ est une partition de V, et donc que H une instance positive de X3C.

On procède en plusieurs étapes.

Lemme 4.1 *L'ensemble d'ensembles*

$$\{X_{i_1,j_1} + t_1, X_{i_2,j_2} + t_2, \ldots, X_{i_k,j_k} + t_k\}$$

est une partition de $[1,n]$.

Preuve. Quels que soient $i \in [1,m]$ et $j \in [1,k]$, l'ensemble $X_{i,j}$ est de cardinal $2k+4$ car il s'écrit comme la réunion disjointe
- de l'hyperarête E_i qui est de cardinal 3,
- du segment F_j qui est de cardinal $2k-1$, et
- des 2 singletons $\{j\}$ et $\{n-j+1\}$.

Par suite, la réunion des k ensembles

$$X_{i_1,j_1} + t_1,\ X_{i_2,j_2} + t_2,\ \ldots,\ X_{i_k,j_k} + t_k$$

est de cardinal au plus $k \cdot (2k+4) = n = \#\,[1,n]$. Il en résulte que l'inclusion (4.3) ne peut pas être stricte et que nos k ensembles sont deux à deux disjoints. Ceci termine la preuve de notre lemme 4.1. **Q.E.D.**

Lemme 4.2 *Les* k *entiers* t_1, t_2, …, t_k *sont strictement compris entre* $-k$ *et* $+k$.

Preuve. Soit $h \in [1,k]$. Comme j_h et $n - j_h + 1$ sont deux éléments de X_{i_h,j_h}, leurs translatés $j_h + t_h$ et $n - j_h + 1 + t_h$ sont deux éléments de $X_{i_h,j_h} + t_h$. Or, le lemme 4.1 p. 97 garantit $X_{i_h,j_h} + t_h \subseteq [1,n]$. On en déduit les inégalités $1 \leq j_h + t_h$ et $n - j_h + 1 + t_h \leq n$, d'où $1 - j_h \leq t_h \leq j_h - 1$. Comme $j_h \in [1,k]$, ce dernier encadrement implique $1 - k \leq t_h \leq k - 1$, ce qui finit la preuve de notre lemme 4.2. **Q.E.D.**

Lemme 4.3 $\{j_1, j_2, \ldots, j_k\} = [1,k]$.

Preuve. Par définition, les entiers j_1, j_2, …, j_k sont des éléments de $[1,k]$. Il reste à montrer que ces k entiers sont deux à deux distincts. La preuve de ce fait repose essentiellement sur la remarque triviale suivante :

Remarque 4.3 *Soit* F *un segment de longueur* $2k-1$ *et* t *un entier strictement compris entre* $-k$ *et* $+k$. *Alors,* $F + t$ *admet pour élément le milieu de* F *:* $\frac{1}{2}(\max F + \min F) \in F + t$.

Supposons (absurde) qu'il existe h, $h' \in [1,k]$ avec $h \neq h'$ et $j_h = j_{h'}$. Par le lemme 4.2 p. 97, t_h et $t_{h'}$ sont strictement compris entre $-k$ et $+k$ donc la remarque 4.3 p. 97 garantit que $F_{j_h} + t_h$ et $F_{j_{h'}} + t_{h'}$ admettent tous les deux pour élément le milieu du segment $F_{j_h} = F_{j_{h'}}$. Or, ceci est en contradiction avec le lemme 4.1 p. 97 qui implique que $X_{i_h,j_h} + t_h$ et $X_{i_{h'},j_{h'}} + t_{h'}$ sont disjoints. On a ainsi démontré le lemme 4.3. **Q.E.D.**

Le lemme 4.3 p. 97 permet de renuméroter les triplets (i_h, j_h, t_h) de manière à ce que, pour tout $h \in [1,k]$, on ait $j_h = h$. Supposons que l'ensemble $Z := \{h \in [1,k] : t_h \neq 0\}$ soit non vide. La remarque suivante va nous permettre de déboucher sur une contradiction.

Remarque 4.4 *Soit une partie finie $X \subseteq \mathbb{Z}$. Alors, l'unique translaté de X contenu dans $[\min X, \max X]$ est X lui-même : pour tout $t \in \mathbb{Z}$, $X + t \subseteq [\min X, \max X]$ implique $t = 0$.*

Soit $\eta := \min Z$. Pour tout $j \in [1, \eta - 1]$, on a à la fois :
- $(X_{i_j,j} + t_j) \cap (X_{i_\eta,\eta} + t_\eta) = \emptyset$ par le lemme 4.1 p. 97 et
- $j \notin Z$ donc $t_j = 0$.

Ainsi, pour tout $j \in [1, \eta - 1]$, on a $X_{i_j,j} \cap (X_{i_\eta,\eta} + t_\eta) = \emptyset$ donc, en particulier, les éléments j et $n - j + 1$ de $X_{i_j,j}$ n'appartiennent pas à $X_{i_\eta,\eta} + t_\eta$. Par conséquent, $X_{i_\eta,\eta} + t_\eta$, qui est inclus dans $[1,n]$ par le lemme 4.1 p. 97, est en fait une partie de $[\eta, n - \eta + 1]$. Ceci permet d'appliquer la remarque 4.4 p. 98 avec $X = X_{i_\eta,\eta}$. Il en résulte $t_\eta = 0$, d'où $\eta \notin Z$: contradiction.

On en déduit que $Z = \emptyset$, donc on a démontré (4.4) et, par suite, H est bien une instance positive de X3C.

Q.E.D.

Le théorème 4.1 p. 95 se raffine en le corollaire suivant qui nous sera utile pour prouver la NP-complétude de Non Detection.

Corollaire 4.1
Le problème SSC *reste* NP*-complet même si on le restreint aux instances $(\mathcal{P}, n, k)$ pour lesquelles $\mathcal{P}$ est un singleton.*

Preuve. Il suffit de réduire le problème SSC dans sa généralité à la restriction que nous considérons.

Supposons donnée une instance quelconque $(\mathcal{P}, n, k)$ de SSC. Posons $m := \#\mathcal{P}$ et numérotons les éléments de $\mathcal{P}$: $\mathcal{P} = \{X_1, X_2, \ldots, X_m\}$. On calcule les entiers τ_1, τ_2, …, τ_m donnés par la récurrence : $\tau_1 = -\min X_1$ et

$$\forall i \in [1, m-1] \qquad \tau_{i+1} = -\min X_{i+1} + \max X_i + \tau_i + n\,. \tag{4.5}$$

Ensuite, pour chaque $i \in [1,m]$, on calcule :

$$\hat{X}_i := X_i + \tau_i\,.$$

On construit enfin l'instance $(\{\hat{X}\}, n, k)$ de SSC avec :

$$\hat{X} := \hat{X}_1 \cup \hat{X}_2 \cup \ldots \cup \hat{X}_m\,.$$

Cette construction est facilement réalisable en temps polynomial à partir de $(\mathcal{P}, n, k)$, compte tenu des hypothèses faites sur l'encodage des instances de SSC.

Décrivons succinctement la forme de notre gadget. Pour tout $i \in [1,m]$, on a $\max \hat{X}_i = \max X_i + \tau_i$ et $\min \hat{X}_i = \min X_i + \tau_i$ donc (4.5) équivaut à :

$$\forall i \in [1, m-1] \qquad \min \hat{X}_{i+1} = \max \hat{X}_i + n\,. \tag{4.6}$$

Ainsi, les ensembles $\hat{X}_1$, $\hat{X}_2$, …, $\hat{X}_m$ apparaissent successivement et dans cet ordre sur la droite réelle. De plus, deux ensembles consécutifs sont séparés par une "brèche" de taille $n - 1$: pour tout $i \in [1, m-1]$, l'élément le plus à droite de $\hat{X}_i$ est situé n positions à gauche de l'élément de plus à gauche de $\hat{X}_{i+1}$.

Exemple 4.10 *Soient* $n := 5$, $m := 3$,

$$X_1 := \{4, 8, 12\}\,, \quad X_2 := \{1, 5, 6, 7\}\,, \quad X_3 := \{3, 10\}$$

et $\mathcal{P} := \{X_1, X_2, X_3\}$. *Dans ce cas, on a* $\tau_1 = -4$, $\tau_2 = 12$ *et* $\tau_3 = 21$, *d'où*

$$\hat{X} = \underbrace{\{0, 4, 8\}}_{\hat{X}_1} \cup \underbrace{\{13, 17, 18, 19\}}_{\hat{X}_2} \cup \underbrace{\{24, 31\}}_{\hat{X}_3}\,.$$

Montrons que notre transformation est bien une M-réduction de SSC à lui-même.

• Supposons que $(\mathcal{P}, n, k)$ soit une instance positive de SSC.
Alors, il existe i_1, i_2, …, $i_k \in [1, m]$ et t_1, t_2, …, $t_k \in \mathbb{Z}$ tels que :

$$[1, n] \subseteq \bigcup_{h=1}^{k} (X_{i_h} + t_h)\,.$$

Pour tout $h \in [1, k]$, posons $\hat{t}_h := t_h - \tau_{i_h}$: on a alors $X_{i_h} + t_h = \hat{X}_{i_h} + \hat{t}_h \subseteq \hat{X} + \hat{t}_h$. Par suite, il vient

$$[1, n] \subseteq \bigcup_{h=1}^{k} (X_{i_h} + t_h) \subseteq \bigcup_{h=1}^{k} \left(\hat{X} + \hat{t}_h\right)$$

donc $(\{\hat{X}\}, n, k)$ est une instance positive de SSC, ce qu'on voulait à ce stade.

• Réciproquement, supposons que $(\{\hat{X}\}, n, k)$ soit une instance positive de SSC.
On dit que deux sous-ensembles $U, V \subseteq \mathbb{Z}$ sont *distants d'au moins* n lorsque pour tous $(u, v) \in U \times V$, on a $|u - v| \geq n$. Les trois lemmes suivants sont indépendants du statut (positivité ou négativité) de l'instance $(\{\hat{X}\}, n, k)$.

Lemme 4.4 *Les ensembles* $\hat{X}_1$, $\hat{X}_2$, …, $\hat{X}_m$ *sont deux à deux distants d'au moins* n.

Preuve. Pour tout $i \in [1, m-1]$, on a d'après (4.6),

$$\min \hat{X}_{i+1} = \max \hat{X}_i + n \geq \max \hat{X}_i \geq \min \hat{X}_i\,,$$

donc la suite $(\min \hat{X}_1, \min \hat{X}_2, \ldots, \min \hat{X}_m)$ est croissante.
Soient deux indices i, j avec $1 \leq i < j \leq m$. On a alors $\max \hat{X}_i + n = \min \hat{X}_{i+1} \leq \min \hat{X}_j$, ou encore

$$\min \hat{X}_j - \max \hat{X}_i \geq n\,.$$

Soit $(u, v) \in \hat{X}_i \times \hat{X}_j$. On a $u \leq \max \hat{X}_i$ et $\min \hat{X}_j \leq v$ d'où :

$$|u - v| \geq v - u \geq \min \hat{X}_j - \max \hat{X}_i \geq n\,.$$

On a ainsi montré que $\hat{X}_i$ et $\hat{X}_j$ étaient distants d'au moins n. Ceci termine la preuve de notre lemme 4.4. **Q.E.D.**

Remarquons que deux sous-ensembles de $\mathbb{Z}$ distants d'au moins n ne peuvent avoir tous les deux une intersection non vide avec un même segment de longueur n. On déduit le lemme suivant du lemme précédent en formalisant cette idée.

Lemme 4.5 *Pour tout* $\hat{t} \in \mathbb{Z}$, *il existe au plus un indice* $i \in [1, m]$ *tel que* $\left(\hat{X}_i + \hat{t}\right) \cap [1, n]$ *soit non vide.*

Preuve. On procède par l'absurde. Supposons qu'il existe deux indices distincts i, j avec $1 \leq i < j \leq m$ tels que $[1, n]$ intersecte à la fois $\hat{X}_i + \hat{t}$ et $\hat{X}_j + \hat{t}$. On peut alors considérer

– un élément u dans $\left(\hat{X}_i + \hat{t}\right) \cap [1, n]$ et
– un élément v dans $\left(\hat{X}_j + \hat{t}\right) \cap [1, n]$.

Comme u et v appartiennent respectivement à $\hat{X}_i + \hat{t}$ et à $\hat{X}_j + \hat{t}$, on a :

$$\left(u - \hat{t}, v - \hat{t}\right) \in \hat{X}_i \times \hat{X}_j \,.$$

Le lemme 4.4 p. 99 permet alors d'écrire $n \leq \left|\left(u - \hat{t}\right) - \left(v - \hat{t}\right)\right| = |u - v|$. Or, comme u et v sont deux éléments de $[1, n]$, on a également $|u - v| < n$: contradiction.

On a ainsi démontré notre lemme 4.5. **Q.E.D.**

Lemme 4.6 *Pour tout $\hat{t} \in \mathbb{Z}$, il existe $i \in [1, m]$ tel que $\left(\hat{X} + \hat{t}\right) \cap [1, n] \subseteq \hat{X}_i + \hat{t}$.*

Preuve. D'après le lemme 4.5 p. 99, il existe un indice $i \in [1, m]$ tel que $\left(\hat{X}_j + \hat{t}\right) \cap [1, n] = \emptyset$ pour tout $j \in [1, n]$ avec $i \neq j$.

Or, on a :

$$\hat{X} + \hat{t} = \left(\bigcup_{j=1}^{m} \hat{X}_j\right) + \hat{t} = \bigcup_{j=1}^{m} \left(\hat{X}_j + \hat{t}\right) \,,$$

d'où :

$$\begin{aligned}\left(\hat{X} + \hat{t}\right) \cap [1, n] &= \bigcup_{j=1}^{m} \left(\left(\hat{X}_j + \hat{t}\right) \cap [1, n]\right) \\ &= \left(\hat{X}_i + \hat{t}\right) \cap [1, n] \\ &\subseteq \hat{X}_i + \hat{t} \,.\end{aligned}$$

Ceci termine la preuve de notre lemme 4.6. **Q.E.D.**

Nous pouvons maintenant achever la preuve du corollaire 4.1. Par hypothèse, il existe $\hat{t}_1$, $\hat{t}_2$, …, $\hat{t}_k \in \mathbb{Z}$ vérifiant :

$$[1, n] \subseteq \bigcup_{h=1}^{k} \left(\hat{X} + \hat{t}_h\right) \,.$$

Pour chaque $h \in [1, k]$, le lemme 4.6 p. 100 permet de trouver un élément $i_h \in [1, m]$ tel que $\left(\hat{X} + \hat{t}_h\right) \cap [1, n] \subseteq \hat{X}_{i_h} + \hat{t}_h$; posant $t_h := \tau_{i_h} + \hat{t}_h$, on a $\hat{X}_{i_h} + \hat{t}_h = X_{i_h} + t_h$. On en déduit :

$$[1, n] = \bigcup_{h=1}^{k} \left(\hat{X} + \hat{t}_h\right) \cap [1, n] \subseteq \bigcup_{h=1}^{k} (X_{i_h} + t_h)$$

donc $(\mathcal{P}, n, k)$ est bien une instance positive de SSC.

Q.E.D.

Théorème 4.2
Le problème Non Detection *est* NP*-complet.*

Preuve. On peut facilement construire un algorithme polynomial qui, étant donnés une instance (g, m, k) de Non Detection et un objet s, retourne `oui` si et seulement si s est une (m, k)-similarité non détectée par g. Cet algorithme est un algorithme de validation polynomial pour Non Detection, donc Non Detection est dans NP. Montrons maintenant que Non Detection est NP-difficile. Grâce au théorème 4.1 p. 95, il suffit d'exhiber une réduction de Karp de SSC à Non Detection .

Supposons donnée une instance $(\mathcal{P}, n, k)$ de SSC. Par le corollaire 4.1 p. 98, on peut supposer qu'il existe $X \subseteq \mathbb{Z}$ tel que $\mathcal{P} = \{X\}$. Sans perte de généralité, on peut également supposer que :

– 0 est le plus petit élément de X et que
– k est inférieur à n car, si l'on a $k \geq n$, alors $(X+1) \cup (X+2) \cup \ldots \cup (X+k)$ contient n entiers consécutifs, donc $(\{X\}, n, k)$ est une instance positive de SSC.

Soit g le mot sur $\{\texttt{\#}, \texttt{-}\}$ de longueur $\max X + 1$ donné par :

$$\forall i \in [1, |g|] \qquad g[i] = \begin{cases} \texttt{\#} & \text{si } |g| - i \in X \\ \texttt{-} & \text{sinon} \end{cases} .$$

Exemple 4.11 *Si* $X = \{0, 1, 3, 6, 7, 9\}$ *alors* $g =$ `#-##--#-##` *:*

$$\begin{array}{rcccccccccccc} [\min X, \max X] & = & \{ & 9, & 8, & 7, & 6, & 5, & 4, & 3, & 2, & 1, & 0 & \} \\ X & = & \{ & 9, & & 7, & 6, & & & 3, & & 1, & 0 & \} \\ g & = & & \texttt{\#} & \texttt{-} & \texttt{\#} & \texttt{\#} & \texttt{-} & \texttt{-} & \texttt{\#} & \texttt{-} & \texttt{\#} & \texttt{\#} & \end{array}$$

Posons en outre $m := n - 1 + |g|$, alors

$$n - 1 = m - |g| . \tag{4.7}$$

L'ensemble X admet pour éléments $|g|-1 = \max X$ et $|g|-|g| = 0 = \min X$ donc $g[1] = g[|g|] = \texttt{\#}$. Autrement dit, la première et la dernière lettre de g sont des `#`, donc g est une graine. On a de plus $0 \leq k < n \leq m$ donc (g, m, k) est bien une instance de Non Detection, que l'on construit en temps polynomial à partir de l'instance $(\{X\}, n, k)$ de SSC. Il reste à vérifier que notre transformation est une M-réduction de SSC à Non Detection.

• Supposons que (g, m, k) soit une instance positive de Non Detection.

Alors, il existe une (m, k)-similarité s qui n'est pas détectée par g. Soit $T := \{i \in [1, m] : s[i] = \texttt{0}\} - |g|$. L'ensemble T est de cardinal k comme translaté d'un ensemble de cardinal $|s|_{\texttt{0}} = k$. Ainsi, pour montrer que $(\{X\}, n, k)$ est une instance positive de SSC, il suffit de vérifier l'inclusion :

$$[0, n-1] \subseteq \bigcup_{t \in T} (X + t) . \tag{4.8}$$

Soit $p \in [0, n-1]$. L'égalité (4.7) garantit que $p \in [0, m - |g|]$. Donc, comme g ne détecte pas s à la position p, il existe $i \in [1, |g|]$ vérifiant $g[i] = \texttt{\#}$ et $s[p+i] = \texttt{0}$. Posant $x := |g| - i$ et $t := p + i - |g|$, on a $x \in X$, $t \in T$ et $p = x + t \in X + t$. On a ainsi démontré (4.8), ce qu'on voulait à ce stade.

• Réciproquement, supposons que $(\{X\}, n, k)$ soit une instance positive de SSC.

Alors, il existe t_1, t_2, …, $t_k \in \mathbb{Z}$ vérifiant :

$$[0, n-1] \subseteq \bigcup_{h=1}^{k} (X + t_h) . \tag{4.9}$$

Soit $s \in \{\texttt{0}, \texttt{1}\}^m$ la similarité donnée par :

$$\forall i \in [1, m] \qquad s[i] = \begin{cases} \texttt{0} & \text{s'il existe } h \in [1, k] \text{ tel que } i = t_h + |g| \\ \texttt{1} & \text{sinon} \end{cases} .$$

Pour montrer que (g, m, k) est une instance positive de Non Detection, il suffit de vérifier que g ne détecte pas s. En effet, s n'est pas forcément une (m, k)-similarité mais $|s|_{\texttt{0}}$ est au plus k, donc on peut changer $k - |s|_{\texttt{0}}$ occurrences de `1` dans s en `0`.

Soit $p \in [0, m - |g|]$. De l'égalité (4.7), on déduit que $p \in [0, n-1]$ donc l'inclusion (4.9) garantit qu'il existe $h \in [1, k]$ et $x \in X$ tels que $p = x + t_h$. Posons $i := |g| - x$. D'une part, on a $|g| - i = x \in X$ d'où $g[i] = \texttt{\#}$. D'autre part, on a aussi $p + i = (x + t_h) + (|g| - x) = t_h + |g|$ d'où $s[p+i] = \texttt{0}$. On en déduit que g ne détecte pas s à la position p. Comme p peut être choisi arbitrairement, g ne détecte pas s et, par suite, (g, m, k) est une instance positive de Non Detection.

Q.E.D.

4.4 Inapproximabilité de MWLS

Dans cette section, nous démontrons que le problème MWLS, introduit dans la section 4.2.2.2, est au moins aussi difficile à approximer que le problème 2-MISH (théorème 4.3 p. 107 et corollaire 4.2).

4.4.1 Gadgets "recyclables"

Dans la section 4.4.2, nous réduisons 2-MISH à MWLS à l'aide d'un gadget dont plusieurs composantes sont également utilisées dans la section 4.5 pour réduire Max k-Cover à RSOS. Dans la section présente, nous décrivons la nature et le fonctionnement de ces composantes de gadgets "recyclables".

Soit un entier $n \geq 1$.

4.4.1.1 Règle de Golomb

Les divers gadgets mis en jeu dans la suite de ce chapitre sont construits à partir de règles de Golomb :

Définition 4.3 (Règle de Golomb [6])
On appelle règle de Golomb *tout ensemble $R \subseteq \mathbb{Z}$ satisfaisant la propriété suivante : pour tous i_1, j_1, i_2, $j_2 \in R$ vérifiant $i_1 \neq j_1$ et $i_2 \neq j_2$, on a $|i_1 - j_1| = |i_2 - j_2|$ si et seulement si $\{i_1, j_1\} = \{i_2, j_2\}$.*

Autrement dit, un ensemble fini d'entiers naturels R est une règle de Golomb si et seulement si la distance entre deux éléments distincts de R caractérise ces deux éléments.

Exemple 4.12 *Les ensembles $Q_3 := \{0, 1, 3\}$, $Q_4 := \{0, 1, 4, 6\}$, et $Q_5 := \{201, 202, 205, 210, 212\}$ sont des règles de Golomb. En effet, dressons les tableaux de distances suivants :*

Q_3	0	1	3
0	0	1	3
1		0	2
3			0

,

Q_4	0	1	4	6
0	0	1	4	6
1		0	3	5
4			0	2
6				0

,

Q_5	201	202	205	210	212
201	0	1	4	9	11
202		0	3	8	10
205			0	5	7
210				0	2
212					0

.

On vérifie facilement que, dans chacune de ces trois tables, les entrées situées au dessus de la diagonale sont deux à deux distinctes.

Pour chaque entier $n \geq 1$ donné, nous avons besoin d'une règle de Golomb de cardinal n, calculable en temps polynomial en fonction de n.

Définition 4.4
On note respectivement $\mathfrak{r}_n$ et $\mathfrak{R}_n$ l'application et l'ensemble d'entiers suivants :

$$\begin{array}{rccc} \mathfrak{r}_n : & \mathbb{Z} & \longrightarrow & \mathbb{Z} \\ & x & \longmapsto & (x-1)n^2 + x^2 \end{array}$$

et

$$\mathfrak{R}_n := \{\mathfrak{r}_n(1), \mathfrak{r}_n(2), \ldots, \mathfrak{r}_n(n)\} .$$

Remarquons que : $\mathfrak{r}_n$ est strictement croissante, $\mathfrak{r}_n(1) = 1$, $\mathfrak{r}_n(n) = n^3$, $\#\mathfrak{R}_n = n$ et $\mathfrak{R}_n \subseteq \left[1, n^3\right]$.

Exemple 4.13 $\mathfrak{R}_5 = \{1, 29, 59, 91, 125\}$.

Les fonctions $\varphi_n(\cdot)$ et $\psi_n(\cdot)$ définies ci-dessous permettent de retrouver deux éléments distincts de $\mathfrak{R}_n$ uniquement à partir de la distance les séparant (voir lemme 4.7 p. 103).

Définition 4.5
Pour tout entier $x \geq n^2$ on pose :

$$\varphi_n(x) := \mathfrak{r}_n\left(\frac{1}{2}\left(\frac{x \bmod n^2}{\lfloor x \,/\, n^2 \rfloor} - \lfloor x \,/\, n^2 \rfloor\right)\right)$$

et

$$\psi_n(x) := \mathfrak{r}_n\left(\frac{1}{2}\left(\frac{x \bmod n^2}{\lfloor x \,/\, n^2 \rfloor} + \lfloor x \,/\, n^2 \rfloor\right)\right) .$$

Notons que pour x vérifiant $0 \leq x < n^2$, on a $\lfloor x \,/\, n^2 \rfloor = 0$ donc les quantités $\varphi_n(x)$ et $\psi_n(x)$ ci-dessus ne sont pas définies pour de tels x.

Lemme 4.7 *Soient i, $j \in \mathfrak{R}_n$ avec $i < j$.*
Alors, on a $i = \varphi_n(j-i)$ et $j = \psi_n(j-i)$.

Preuve. Soient x, $y \in [1, n]$ tels que

$$i = \mathfrak{r}_n(x) \quad \text{et} \quad j = \mathfrak{r}_n(y) \; . \tag{4.10}$$

Posons $\chi := y - x$ et $\varrho := y^2 - x^2$.

– D'une part, on vérifie facilement que

$$x = \frac{1}{2}\left(\frac{\varrho}{\chi} - \chi\right) \quad \text{et} \quad y = \frac{1}{2}\left(\frac{\varrho}{\chi} + \chi\right) . \tag{4.11}$$

– D'autre part, on a

$$j - i = \big((y-1)n^2 + y^2\big) - \big((x-1)n^2 + x^2\big) = \chi n^2 + \varrho \, .$$

De plus, comme la fonction $\mathfrak{r}_n$ est strictement croissante, $i < j$ force $x < y$, et donc l'entier ϱ est positif. Il vérifie également $\varrho < y^2 \leq n^2$. Par suite, χ et ϱ sont respectivement le quotient et le reste de la division euclidienne de $j - i$ par n^2 :

$$\left\{ \begin{array}{rcl} \chi & = & \lfloor (j-i) \,/\, n^2 \rfloor \\ \varrho & = & (j-i) \bmod n^2 \end{array} \right. . \tag{4.12}$$

Il suffit alors de combiner (4.10), (4.11) et (4.12) pour obtenir :

$$i = \varphi_n(j-i) \quad \text{et} \quad j = \psi_n(j-i) \, .$$

Q.E.D.

On déduit immédiatement du lemme 4.7 p. 103 que $\mathfrak{R}_n$ est bien une règle de Golomb.

Règles de Golomb optimales (digression). Nous terminons cette section par une digression : nous exposons le problème le plus étudié concernant les règles de Golomb.

Tout d'abord, notons que, pour tout entier $n \geq 1$, il y a une infinité de règles de Golomb de cardinal n.

Exemple 4.14 *Les ensembles d'entiers $\{\mathfrak{r}_n(1), \mathfrak{r}_n(2), \ldots, \mathfrak{r}_n(n-1), j\}$ ($j \in \mathbb{Z}$) sont deux à deux distincts et sont tous des règles de Golomb de cardinal n lorsque j est assez grand. En effet, pour j assez grand, la distance de j à tout point de $\{\mathfrak{r}_n(1), \mathfrak{r}_n(2), \ldots, \mathfrak{r}_n(n-1)\}$ est strictement supérieure à la distance maximale entre deux points de $\{\mathfrak{r}_n(1), \mathfrak{r}_n(2), \ldots, \mathfrak{r}_n(n-1)\}$.*

On dit qu'une règle de Golomb R_* est *optimale* lorsque, pour toute règle de Golomb R vérifiant $\#R = \#R_*$, on a $\max R - \min R \geq \max R_* - \min R_*$. Autrement dit la règle de Golomb R_* est optimale si et seulement si elle est d'*étendue* minimale parmi toutes les règles de Golomb de même cardinal.

Exemple 4.15 (Suite de l'exemple 4.12 p. 102) *Les règles de Golomb Q_3, $Q_4 = \{0, 1, 3, 6\}$ et Q_5 sont optimales contrairement à $Q'_4 := \{0, 1, 3, 7\}$.*

Le problème suivant fait encore aujourd'hui l'objet de nombreuses investigations (voir `http://www.distributed.net/ogr/index.php.fr`) : "*Étant donné un entier $n \geq 1$, trouver une règle de Golomb optimale de cardinal n*". On connaît aujourd'hui des règles de Golomb optimales de cardinal n pour tout $n \leq 24$.

4.4.1.2 Encodage d'ensembles à l'aide de graines et de similarités

Les deux gadgets définis ci-dessous (définitions 4.6 p. 104 et 4.7 p. 105) servent dans les sections 4.4.2 et 4.5.2 à encoder des sommets, des arêtes et des ensembles finis.

Définition 4.6 (Gadget)
Soit une partie $X \subseteq [1,n]$.

On note $\mathfrak{w}_X^n$ le mot sur $\{\mathtt{\#},\mathtt{-}\}$ de longueur n^3 donné par :

$$\forall i \in \left[1,n^3\right] \qquad \mathfrak{w}_X^n[i] = \begin{cases} \mathtt{\#} & \text{si } \exists x \in X,\ i = \mathfrak{r}_n(x) \\ \mathtt{-} & \text{sinon} \end{cases} .$$

On note $\mathfrak{g}_X^n$ la plus longue sous-chaîne de $\mathfrak{w}_X^n$ qui est une graine.

Lorsque X est non vide alors $\mathfrak{g}_X^n$ est la graine obtenue à partir de $\mathfrak{w}_X^n$ en effaçant les $\mathtt{-}$ éventuellement situés au début et à la fin : $\mathfrak{w}_X^n = (\mathtt{-})^p \mathfrak{g}_X^n (\mathtt{-})^q$ avec p, $q \in \mathbb{N}$. Dans le cas dégénéré où $X = \emptyset$, on a $\mathfrak{w}_X^n = (\mathtt{-})^{n^3}$ et la graine $\mathfrak{g}_X^n$ est réduite au mot vide.

Remarque 4.5 *Pour tout $X \subseteq [1,n]$, $\mathfrak{g}_X^n$ et $\mathfrak{w}_X^n$ sont de poids $\#X$.*

Exemple 4.16 (Suite de l'exemple 4.13 p. 102) *Prenons $n := 5$ et $X := \{2,3,5\}$. On a alors $\#X = 3$, $n^3 = 125$, $\mathfrak{R}_n = \mathfrak{R}_5 = \{1,29,59,91,125\}$,*

$$\mathfrak{w}_X^n = \mathfrak{w}_{\{2,3,5\}}^5 = (\mathtt{-})^{28}\mathtt{\#}(\mathtt{-})^{29}\mathtt{\#}(\mathtt{-})^{65}\mathtt{\#}$$

et

$$\mathfrak{g}_X^n = \mathfrak{g}_{\{2,3,5\}}^5 = \mathtt{\#}(\mathtt{-})^{29}\mathtt{\#}(\mathtt{-})^{65}\mathtt{\#} .$$

Le point (i) du lemme suivant permet d'exprimer X en fonction de $\mathfrak{g}_X^n$ et de n, pour toute partie $X \subseteq [1,n]$ de cardinal au moins 2. Il garantit en particulier que les $\mathfrak{g}_X^n$ (pour $X \subseteq [1,n]$ de cardinal au moins 2) sont deux à deux distincts.

Lemme 4.8 *Soient deux parties X, $Y \subseteq [1,n]$ de cardinal au moins 2.*

Alors, on a

(i). $X = \{x \in [1,n] : \mathfrak{g}_X^n[\mathfrak{r}_n(x) - \varphi_n(|\mathfrak{g}_X^n| - 1) + 1] = \mathtt{\#}\}$, et

(ii). $\mathfrak{g}_X^n = \mathfrak{g}_Y^n$ si et seulement $X = Y$.

Preuve. Le point (ii) est une conséquence immédiate du point (i). Il reste donc à montrer le point (i).

On a

$$X = \{x \in [1,n] : \mathfrak{w}_X^n[\mathfrak{r}_n(x)] = \mathtt{\#}\} \quad \text{et} \quad \mathfrak{w}_X^n = (\mathtt{-})^p \mathfrak{g}_X^n (\mathtt{-})^q$$

pour certains p, $q \in \mathbb{N}$, d'où

$$X = \{x \in [1,n] : \mathfrak{g}_X^n[\mathfrak{r}_n(x) - p] = \mathtt{\#}\} .$$

Pour montrer le point (i), il suffit de vérifier que $-p = -\varphi_n(\ell - 1) + 1$ avec $\ell := |\mathfrak{g}_X^n|$.

Comme X est de cardinal au moins 2, $\mathfrak{w}_X^n$ contient au moins 2 occurrences distinctes de la lettre $\mathtt{\#}$ (remarque 4.5 p. 104). L'occurrence la plus à gauche est située à la position p et correspond à la première lettre de $\mathfrak{g}_X^n$. La plus à droite est située à la position $p + \ell - 1$ et correspond à la dernière lettre de $\mathfrak{g}_X^n$. On a ainsi[2] $\mathfrak{w}_X^n[p+1] = \mathfrak{w}_X^n[p+\ell] = \mathtt{\#}$, donc $p+1$ et $p+\ell$ sont deux éléments de $\mathfrak{R}_n$. Le lemme 4.7 p. 103 garantit alors que $p + 1 = \varphi_n((p+\ell) - (p+1)) = \varphi_n(\ell - 1)$. On a ainsi démontré le point (i). **Q.E.D.**

[2] Rappelons que, pour tout mot w, nous avons convenu que la lettre de w située à la position i est $w[i+1]$ (voir section 1.1.3.2).

Notons que $\mathfrak{g}^n_{\{1\}} = \mathfrak{g}^n_{\{2\}} = \cdots = \mathfrak{g}^n_{\{n\}} = \texttt{\#}$. Il est donc bien utile de se restreindre à des parties X, $Y \subseteq [1,n]$ de cardinal au moins 2 dans le point (ii) du lemme ci-dessus.

Définition 4.7 (Re-gadget)
Soit $v \in [1,n]$.
On note $\mathfrak{t}^n_v$ la similarité de longueur n^3 donnée par :

$$\forall i \in \left[1, n^3\right] \qquad \mathfrak{t}^n_v[i] = \begin{cases} \texttt{1} & \text{si } \exists x \in [1,n],\ x \neq v \text{ et } i = \mathfrak{r}_n(x) \\ \texttt{0} & \text{sinon} \end{cases}.$$

Exemple 4.17 (Suite de l'exemple 4.13 p. 102) *Pour $n = 5$ et $v = 2$, on a $\mathfrak{R}_n = \mathfrak{R}_5 = \{1, 29, 59, 91, 125\}$ et $\mathfrak{t}^n_v = \mathfrak{t}^5_2 = \texttt{10}^{57}\texttt{10}^{31}\texttt{10}^{33}\texttt{1}$.*

Les similarités $\mathfrak{t}^n_v$ ($v \in [1,n]$) possèdent naturellement la propriété suivante :

Lemme 4.9 *Soient $v \in [1,n]$ et $w \in \{\texttt{\#}, \texttt{-}\}^{n^3}$.*
Alors, w détecte $\mathfrak{t}^n_v$ si et seulement s'il existe $X \subseteq [1,n]$ tel que $v \notin X$ et $w = \mathfrak{w}^n_X$.

Preuve.

• Supposons que $w \in \{\texttt{\#}, \texttt{-}\}^{n^3}$ détecte $\mathfrak{t}^n_v$.
Comme $|w| = n^3 = |\mathfrak{t}^n_v|$, w détecte $\mathfrak{t}^n_v$ à la position 0 (point (ii) de la remarque 4.1 p. 91) :

$$\forall i \in \left[1, n^3\right] \qquad (\mathfrak{t}^n_v[i] = \texttt{0}) \Rightarrow (w[i] = \texttt{-}). \tag{4.13}$$

Posons $X := \{x \in [1,n] : w[\mathfrak{r}_n(x)] = \texttt{\#}\}$. Tout d'abord, on a $\mathfrak{t}^n_v[\mathfrak{r}_n(v)] = \texttt{0}$, donc $w[\mathfrak{r}_n(v)] = \texttt{-}$ d'après (4.13), et par suite $v \notin X$. Il reste à vérifier que $w = \mathfrak{w}^n_X$ ou encore que, pour chaque $i \in \left[1, n^3\right]$, $w[i] = \mathfrak{w}^n_X[i]$.
Soit $i \in \left[1, n^3\right]$.
– S'il existe $x \in X$ tel que $i = \mathfrak{r}_n(x)$ alors on a $w[i] = \texttt{\#} = \mathfrak{w}^n_X[i]$.
– Supposons maintenant le contraire, c'est-à-dire $i \notin \{\mathfrak{r}_n(x) : x \in X\}$. On a alors $\mathfrak{w}^n_X[i] = \texttt{-}$. Pour montrer que $w[i] = \texttt{-}$ on se place successivement dans chacun des deux cas complémentaires (a) et (b) ci-dessous.
  (a) Si $i \in \mathfrak{R}_n$ alors il existe $x \in [1,n]$ tel que $i = \mathfrak{r}_n(x)$. Comme on s'est placé sous l'hypothèse $x \notin X$, on a $w[i] = w[\mathfrak{r}_n(x)] = \texttt{-}$.
  (b) Réciproquement, si $i \notin \mathfrak{R}_n$ alors on a $\mathfrak{t}^n_v[i] = \texttt{0}$ donc $w[i] = \texttt{-}$ d'après (4.13).

On a ainsi prouvé que $w = \mathfrak{w}^n_X$, ce qu'on voulait à ce stade.

• Soit $X \subseteq [1,n]$ avec $v \notin X$. Montrons que $\mathfrak{w}^n_X$ détecte $\mathfrak{t}^n_v$ à la position 0.
Soit $i \in \left[1, n^3\right]$ tel que $\mathfrak{w}^n_X[i] = \texttt{\#}$. Alors, il existe $x \in X$ tel que $i = \mathfrak{r}_n(x)$. Or, v n'est pas dans X donc on a $x \neq v$. Il en résulte $\mathfrak{t}^n_v[i] = \texttt{1}$.

Q.E.D.

On déduit de ce lemme la propriété fondamentale des $\mathfrak{t}^n_v$ (pour $v \in [1,n]$) :

Lemme 4.10 *Soit $v \in [1,n]$ et soit g une graine.*
Alors, g détecte $\mathfrak{t}^n_v$ si et seulement s'il existe $X \subseteq [1,n]$ tel que $v \notin X$ et $g = \mathfrak{g}^n_X$.

Preuve.

• Supposons qu'il existe $X \subseteq [1,n]$ vérifiant $v \notin X$ et $g = \mathfrak{g}^n_X$. Comme $v \notin X$, $\mathfrak{w}^n_X$ détecte $\mathfrak{t}^n_v$ (lemme 4.9 p. 105). Comme g est une sous-chaîne de $\mathfrak{w}^n_X$, g détecte également $\mathfrak{t}^n_v$ d'après le point (iii) de la remarque 4.1 p. 91 : ce qu'on voulait à ce stade.

• Réciproquement, supposons que g détecte $\mathfrak{t}^n_v$. Soit $p \in \left[0, n^3 - |g|\right]$ tel que g détecte $\mathfrak{t}^n_v$ à la position p. Alors, $w := (\texttt{-})^p g (\texttt{-})^{n^3-|g|-p}$ est un mot sur $\{\texttt{\#}, \texttt{-}\}$ de longueur n^3 détectant $\mathfrak{t}^n_v$ à la position 0. On en déduit (lemme 4.9 p. 105) que w est de la forme $w = \mathfrak{w}^n_X$ avec $X \subseteq [1,n]$ tel que $v \notin X$. Comme g est, par construction de w, la plus longue sous-chaîne de w qui est une graine, on a aussi $g = \mathfrak{g}^n_X$.

Q.E.D.

4.4.2 Gadget spécifique à notre réduction de 2-MISH à MWLS

Nous avons maintenant tous les outils pour construire un gadget permettant de réduire 2-MISH à MWLS.

Définition 4.8
Soit un graphe $G = (V, \mathcal{E})$. On pose $n := |G|$ et on suppose que $V = [1, n]$.

Pour tout $E \in \mathcal{E}$, on pose

$$s_E^n := \mathfrak{t}_x^n \mathtt{0}^{n^3} \mathfrak{t}_y^n \text{ où } x \text{ et } y \text{ sont donnés par } E = \{x, y\} \text{ et } 1 \leq x < y \leq n.$$

On note :

$$S_G := \left\{ \mathtt{1}^{n^3} \right\} \cup \{ s_E^n : E \in \mathcal{E} \} \cup \left\{ \mathtt{1}^{3n^3} \mathtt{0}^p : p \in \left[1, n^2 - \#\mathcal{E} - 1\right] \right\}.$$

Décrivons succinctement le rôle joué par chaque élément de S_G. Soit g une graine de poids au moins 2 détectant toutes les similarités appartenant à S_G.

1. Les $\mathtt{1}^{3n^3}\mathtt{0}^p$ ($1 \leq p < n^2 - \#\mathcal{E}$) ne servent qu'à ajuster le cardinal de S_G (voir preuve du point (i) du lemme 4.11 p. 106).
2. La similarité $\mathtt{1}^{n^3}$ force g à être de longueur au plus n^3 (voir preuve du point (iii) du lemme 4.11 p. 106).
3. Pour tout $E \in \mathcal{E}$, s_E^n force alors g à être de la forme $g = \mathfrak{g}_X^n$ avec $X \subseteq [1, n]$ n'admettant pas E pour sous-ensemble (voir preuve du point (iii) du lemme 4.11 p. 106).

Par ailleurs, pour chaque indépendant I de G, $\mathfrak{g}_I^n$ détecte toutes les similarités appartenant à S_G (voir preuve du point (ii) du lemme 4.11 p. 106).

Prouvons de manière plus formelle ces propriétés.

Lemme 4.11 *Soit un graphe G admettant $[1, n]$ pour ensemble de sommets.*

(i). S_G est de cardinal n^2.

(ii). Soit I un indépendant de G. Alors, il existe une graine de poids $\#I$ détectant toutes les similarités appartenant à S_G.

(iii). Supposons donnée une graine g détectant toutes les similarités appartenant à S_G. Alors, G admet un indépendant de cardinal $\|g\|$ calculable en temps polynomial à partir de g et de n.

Preuve.

(i). Comme les similarités s_E^n (pour $E \in \mathcal{E}$) sont toutes de longueur $3n^3$, l'ensemble

$$\left\{ \mathtt{1}^{n^3} \right\} \cup \left\{ \mathtt{1}^{3n^3} \mathtt{0}^p : p \in \left[1, n^2 - \#\mathcal{E} - 1\right] \right\}$$

qui est de cardinal $1 + (n^2 - \#\mathcal{E} - 1)$, est disjoint de l'ensemble $\{s_E^n : E \in \mathcal{E}\}$ qui est de cardinal $\#\mathcal{E}$. On a ainsi $\#S_G = (1 + (n^2 - \#\mathcal{E} - 1)) + \#\mathcal{E} = n^2$.

(ii). La graine $\mathfrak{g}_I^n$ est de poids $\#I$ (remarque 4.5 p. 104). De plus, comme $\mathfrak{g}_I^n$ est de longueur au plus n^3, $\mathfrak{g}_I^n$ détecte $\mathtt{1}^{n^3}$ et les $\mathtt{1}^{3n^3}\mathtt{0}^p$ pour $p \geq 1$ (voir le point (i) de la remarque 4.2 p. 91 et le point (iv) de la remarque 4.1 p. 91). Enfin, soit une arête E de G. Il existe une extrémité $v \in E$ telle que $v \notin I$ donc, par le lemme 4.10 p. 105, $\mathfrak{g}_I^n$ détecte $\mathfrak{t}_v^n$. Comme $\mathfrak{t}_v^n$ est préfixe ou suffixe de s_E^n, $\mathfrak{g}_I^n$ détecte s_E^n.

On a ainsi montré que $\mathfrak{g}_I^n$ détecte toutes les similarités appartenant à S_G.

(iii). Si g est de poids 0 (resp. 1) alors on peut considérer l'indépendant $\emptyset$ (resp. $\{1\}$) de G. Supposons désormais $\|g\| \geq 2$.

Comme g détecte $\mathtt{1}^{n^3} \in S_G$, g est de longueur au plus n^3 d'après le point (i) de la remarque 4.2 p. 91.

Soit une arête $E = \{x, y\}$ de G. Comme g détecte $s_E^n \in S_G$, il existe une sous-chaîne f_E de s_E^n, de même longueur que g, telle que g détecte f_E (à la position 0).

- D'une part, g commence et finit par la lettre `#` donc f_E commence et finit par la lettre `1`.
- D'autre part, on a $|f_E| = |g| \leq n^3$ donc le bloc $\mathtt{0}^{n^3}$ séparant $\mathfrak{t}_x^n$ et $\mathfrak{t}_y^n$ dans s_E^n est plus long que f_E.

Il en résulte que f_E est entièrement contenu dans $\mathfrak{t}^n_x$ ou dans $\mathfrak{t}^n_y$. Ainsi, il existe $v_E \in E$ tel que g détecte $\mathfrak{t}^n_{v_E}$. Par suite, le lemme 4.10 p. 105 garantit l'existence d'une partie $X_E \subseteq [1,n]$ vérifiant $v_E \notin X_E$ et $g = \mathfrak{g}^n_{X_E}$. Or, on a $\#X_E = \left\|\mathfrak{g}^n_{X_E}\right\| = \|g\| \geq 2$ (remarque 4.5 p. 104) donc, par le point (ii) du lemme 4.8 p. 104, tous les X_E (pour $E \in \mathcal{E}$) sont égaux à un même sous-ensemble $I \subseteq [1,n]$. On a par conséquent $g = \mathfrak{g}^n_I$ et l'ensemble I est un indépendant de G de cardinal $\|g\|$. De plus, I est calculable en temps polynomial à partir de g et de n, grâce à la formule donnée dans le point (i) du lemme 4.8 p. 104.

Q.E.D.

Théorème 4.3
Soit une fonction b associant, à chaque entier $n \geq 0$, un réel $b(n) \geq 1$.

On suppose qu'il existe un algorithme d'approximation pour MWLS *de borne $b(\#S)$.*

Alors, il existe un algorithme d'approximation pour 2-MISH *de borne $b(|G|^2)$.*

Preuve. Pour tout $S \subseteq \{\mathtt{0},\mathtt{1}\}^\star$, on note $\mathrm{opt}(S)$ le poids maximum pour une graine détectant toutes les similarités appartenant à S.

On suppose qu'il existe un algorithme d'approximation A pour MWLS de borne $b(\#S)$. Exhibons un algorithme d'approximation pour 2-MISH de borne $b(|G|^2)$.

Supposons donnée une instance G de 2-MISH. On peut supposer que le graphe G admet $[1,n]$ pour ensemble de sommets.

1. On calcule S_G.
2. En simulant une application de l'algorithme A sur S_G, on calcule une graine g, détectant toutes les similarités appartenant à S_G, avec $\|g\| \geq \mathrm{opt}(S_G) \,/\, b(\#S_G)$.
3. On retourne un indépendant de G de cardinal $\|g\|$.

Cet algorithme s'implante en temps polynomial :
- l'étape 1 d'après la forme de notre gadget (voir les définitions 4.4 p. 102, 4.6 p. 104, 4.7 p. 105 et 4.8 p. 106),
- l'étape 2 car l'algorithme A est polynomial, et
- l'étape 3 par le point (iii) du lemme 4.11 p. 106.

Bornons ses performances. Les points (i) et (ii) du lemme 4.11 p. 106 garantissent respectivement $\#S_G = |G|^2$ et $\mathrm{opt}(S_G) \geq \alpha(G)$, d'où :

$$\|g\| \geq \frac{\mathrm{opt}(S_G)}{b(\#S_G)} = \frac{\mathrm{opt}(S_G)}{b(|G|^2)} \geq \frac{\alpha(G)}{b(|G|^2)}.$$

Q.E.D.

En combinant le théorème 4.3 ci-avant et les résultats de Håstad [47] (voir section 1.3.3) on obtient :

Corollaire 4.2
Supposons que, pour un certain un réel $\delta \geq 0$, il existe un algorithme d'approximation pour MWLS *de borne $(\#S)^\delta$.*
- *Si δ est strictement inférieur à 0.5 alors* NP = ZPP.
- *Si δ est strictement inférieur à 0.25 alors* NP = P.

4.5 Inapproximabilité de RSOS

Dans cette section, nous démontrons que le problème RSOS, défini dans la section 4.2.2.3, est au moins aussi difficile à approximer que le problème Max k-Cover introduit dans la section 4.5.1 ci-dessous. Ainsi, sous l'hypothèse P $\subsetneq$ NP, il n'existe pas d'algorithme d'approximation pour RSOS de borne constante ρ, quel que soit le réel ρ vérifiant $1 \leq \rho < \frac{e}{e-1}$ (thèorème 4.4 p. 110).

4.5.1 Le problème Maximum k-Cover

On nomme MAX k-COVER le problème d'optimisation suivant : "*On suppose donnés un hypergraphe $H = (V, \mathcal{E})$ et un entier $k \geq 0$. On cherche une partie $\mathcal{C} \subseteq \mathcal{E}$ de cardinal au plus k telle que l'ensemble $\bigcup \mathcal{C}$ des sommets de H couverts par $\mathcal{C}$ soit de cardinal maximum*". Notons que, dans une instance (H, k) de MAX k-COVER, k ne désigne pas un seuil d'acceptation : il ne faut pas confondre MAX k-COVER et MINIMUM SET COVER (voir exemple 1.21 p. 16).

Dans la section 4.5.1.1 ci-dessous nous présentons les résultats connus sur l'approximabilité de MAX k-COVER. Ensuite, dans la section 4.5.1.2, nous introduisons le problème MAX k-HITTING SET (MAX k-HS en abrégé), qui est simplement un formulation alternative de MAX k-COVER. C'est par réduction depuis MAX k-HS que nous montrons notre résultat principal concernant RSOS dans la section 4.5.2.

4.5.1.1 Approximabilité de Max k-Cover

L'approximabilité de MAX k-COVER a été exhaustivement étudiée.

1. MAX k-COVER admet un algorithme d'approximation glouton de borne $\frac{1}{1-\left(1-\frac{1}{k}\right)^k} < \frac{e}{e-1}$ [49].
2. Mais, quel que soit le réel ρ vérifiant $1 \leq \rho < \frac{e}{e-1}$, le problème d'évaluation associé à MAX k-COVER n'admet pas d'algorithme d'approximation de borne constante ρ, sauf si NP = P [38].

Rappelons que l'on a défini la notion de problème d'évaluation associé à un problème d'optimisation dans la section 1.2.5.2.

4.5.1.2 Une formulation alternative de Max k-Cover : Max k-Hitting Set

On nomme MAXIMUM k-HITTING SET (MAX k-HS en abrégé) le problème suivant : "*Supposons donnés un n-uplet $(F_1, F_2, \ldots, F_n)$ d'ensembles finis et un entier $k \geq 0$. On cherche un ensemble C de cardinal k maximisant le nombre des indices $j \in [1, n]$ tels que $C \cap F_j \neq \emptyset$*". Dans cette section, nous montrons que MAX k-HS est exactement aussi difficile à approximer que MAX k-COVER comme suggéré dans [49, chapitre 3].

Proposition 4.2
Soit un réel ρ avec $1 \leq \rho < \frac{e}{e-1}$.

Alors, sous l'hypothèse P $\subsetneq$ NP*, il n'existe pas d'algorithme d'approximation de borne constante ρ pour le problème d'évaluation associé à* MAX k-HS.

Preuve. Supposons (absurde) que MAX k-HS$_E$ admette un algorithme d'approximation de borne ρ. Nous allons montrer, à l'aide d'une réduction, que MAX k-COVER$_E$ admet un algorithme d'approximation de même borne, de manière à pouvoir appliquer le résultat de Feige [38] cité dans la section 4.5.1.1.

Supposons donnée une instance (H, k) de MAX k-COVER.

Construction d'une instance de Max k-HS. On note $n := |H|$, $\mathcal{E}$ l'ensemble des hyperarêtes de H et $m := \#\mathcal{E}$. Sans perte de généralité, on peut supposer que H admet $[1, n]$ pour ensemble de sommets. On numérote également les éléments de $\mathcal{E}$: $\mathcal{E} = \{E_1, E_2, \ldots, E_m\}$. Dans la suite de cette preuve, j désignera toujours un sommet de H et i indexera une hyperarête de H.

Pour chaque $j \in [1, n]$, on calcule l'ensemble F_j des indices $i \in [1, m]$ tels que $j \in E_i$. Ainsi, les E_i (pour $i \in F_j$) sont les hyperarêtes de H contenant j et on a :

$$\forall (i, j) \in [1, m] \times [1, n] \qquad i \in F_j \iff j \in E_i \,.$$

On construit enfin l'instance $((F_1, F_2, \ldots, F_n), k)$ de MAX k-HS.

Correction de notre réduction. Soit un entier $N \geq 0$ quelconque.

Pour tout ensemble C, posons $J_C := \{j \in [1, n] : C \cap F_j \neq \emptyset\}$. Résoudre le problème MAX k-HS sur l'instance $((F_1, F_2, \ldots, F_n), k)$ consiste à trouver un ensemble C de cardinal k maximisant le cardinal de J_C.

Pour terminer notre preuve, il suffit de montrer que les deux assertions suivantes sont équivalentes.

(i). Il existe un ensemble C de cardinal k vérifiant $\#J_C \geq N$.

(ii). Il existe une partie $\mathcal{C} \subseteq \mathcal{E}$ de cardinal au plus k vérifiant $\#\left(\bigcup \mathcal{C}\right) \geq N$.

On va tout d'abord vérifier l'assertion :

$$\forall C \subseteq [1,m] \qquad \bigcup_{c \in C} E_c = J_C \,. \tag{4.14}$$

• Montrons $J_C \subseteq \bigcup_{c\in C} E_c$.

Soit $j \in J_C$. Alors, $C \cap F_j$ est non vide donc cette intersection contient un certain élément c. Comme $c \in F_j$, on a $j \in E_c$ et, donc, comme $c \in C$, on a $j \in \bigcup_{c\in C} E_c$.

• Réciproquement, montrons $\bigcup_{c\in C} E_c \subseteq J_C$.

Soit $j \in \bigcup_{c\in C} E_c$. Il existe alors $c \in C$ tel que $j \in E_c$. Ainsi, on a $c \in C \cap F_j$ donc $C \cap F_j \neq \emptyset$ et, par suite, $j \in J_C$.

On a ainsi démontré (4.14). Nous pouvons maintenant montrer facilement l'équivalence entre (i) et (ii).

• Montrons que $(i) \Rightarrow (ii)$. Supposons (i) : il existe un ensemble C de cardinal au plus k tel que l'ensemble J_C soit de cardinal au moins N.

Comme tous les ensembles F_j (pour $j \in [1,n]$) sont des parties de $[1,m]$, on ne change pas J_C en remplaçant C par $C \cap [1,m]$. Ainsi, on peut supposer $C \subseteq [1,m]$ et poser $\mathcal{C} := \{E_c : c \in C\}$. Comme C est de cardinal au plus k, il en est de même pour $\mathcal{C}$. De plus, (4.14) garantit $\bigcup \mathcal{C} = J_C$ donc $\bigcup \mathcal{C}$ est de cardinal au moins N.

On a ainsi démontré (ii).

• Réciproquement, montrons que $(ii) \Rightarrow (i)$. Supposons (ii) : il existe une partie $\mathcal{C} \subseteq \mathcal{E}$ de cardinal au plus k avec $\#\left(\bigcup \mathcal{C}\right) \geq N$.

On écrit $\mathcal{C}$ sous la forme $\mathcal{C} = \{E_{c_1}, E_{c_2}, \ldots, E_{c_k}\}$ où c_1, c_2, …, c_k sont des éléments de $[1,m]$ (non nécessairement distincts deux à deux). Posant $C := \{c_1, c_2, \ldots, c_k\}$, on a $\#C \leq k$. De plus, on déduit de (4.14) que $J_C = \bigcup_{c\in C} E_c = \bigcup \mathcal{C}$ donc J_C est de cardinal au moins N. Quitte à rajouter des éléments dans C, on peut, sans faire décroître le cardinal de J_C, supposer que C est de cardinal exactement k.

On a ainsi démontré (i).

Q.E.D.

La proposition suivante garantit que tout algorithme d'approximation pour Max k-Cover peut être transformé en un algorithme d'approximation pour Max k-HS. Ce résultat n'est d'aucune utilité pour la suite. Nous le donnons simplement par souci d'exhaustivité. Aussi, sa preuve est simplement esquissée. Elle est de toute façon du même genre que celle de la proposition 4.2 précédente.

Proposition 4.3
Soit une fonction b associant, à chaque entier $k \geq 0$, un réel $b(k) \geq 1$. On suppose qu'il existe un algorithme d'approximation pour Max k-Cover *de borne $b(k)$.*

Alors, il existe également un algorithme d'approximation pour Max k-HS *de borne $b(k)$.*

Preuve. Par hypothèse, Max k-Cover admet un algorithme d'approximation A de borne $b(k)$. Construisons un algorithme d'approximation pour Max k-HS de même borne.

Supposons donnée une instance $((F_1, F_2, \ldots, F_n), k)$ de Max k-HS. Posons $U := F_1 \cup F_2 \cup \ldots \cup F_n$. Pour chaque $u \in U$, on calcule l'ensemble E_u des indices $j \in [1,n]$ tels que $u \in F_j$. On construit ensuite l'ensemble $\mathcal{E} := \{E_u : u \in U\}$ puis l'instance (H,k) de Max k-Cover où $H := ([1,n], \mathcal{E})$. Cette construction prend évidemment un temps polynomial et on vérifie facilement qu'elle possède la propriété suivante :

$$\forall C \subseteq U \qquad \bigcup_{c \in C} E_c = \{j \in [1,n] : C \cap F_j \neq \emptyset\} \,. \tag{4.15}$$

Appliquons maintenant l'algorithme A sur l'entrée (H,k) : on obtient, en temps polynomial, une partie $\mathcal{C} \subseteq \mathcal{E}$ de cardinal au plus k telle que $\#\left(\bigcup \mathcal{C}\right)$ soit au moins égal à $1 \,/\, b(k)$ fois l'optimum. Notant $h := \#\mathcal{C}$, on calcule u_1, u_2, …, $u_h \in U$ tels que $\mathcal{C} = \{E_{u_1}, E_{u_2}, \ldots, E_{u_h}\}$. Enfin, on retourne l'ensemble $\{u_1, u_2, \ldots, u_h\}$, augmenté artificiellement de $k-h$ éléments distincts quelconques de manière à obtenir une solution de Max k-HS. La correction de notre algorithme se déduit de l'assertion (4.15). **Q.E.D.**

4.5.2 Preuve de notre résultat principal concernant RSOS

Cette section est dévolue à la preuve du théorème 4.4 ci-dessous.

Théorème 4.4
Soit un réel ρ avec $1 \leq \rho < \frac{e}{e-1}$.

Alors, sous l'hypothèse P $\subsetneq$ NP, *le problème d'évaluation associé à* RSOS *n'admet pas d'algorithme d'approximation de borne constante ρ.*

Preuve. Étant donnée une instance $((F_1, F_2, \ldots, F_n), k)$ de Max k-HS, on cherche à construire en temps polynomial une instance (ϖ, S) de RSOS telle que, pour tout entier N assez grand, les deux assertions suivantes soient équivalentes.

(A_N). Il existe une graine de poids ϖ détectant au moins N similarités appartenant à S.

(B_N). Il existe un ensemble C de cardinal k vérifiant :

$$\#\{j \in [1,n] : C \cap F_j \neq \emptyset\} \geq N\,.$$

Si l'on parvient à exhiber une telle réduction, alors approximer RSOS_E sur l'instance réduite (ϖ, S) reviendra à approximer Max k-HS_E sur l'instance initiale $((F_1, F_2, \ldots, F_n), k)$. Ainsi, tout algorithme d'approximation de borne constante pour RSOS_E induira un algorithme d'approximation de même borne pour Max k-HS_E. La proposition 4.2 p. 108 permettra alors de conclure la preuve du théorème 4.4.

Dans la section 4.5.2.1, on décrit un algorithme de réduction polynomial construisant une instance (ϖ, S) de RSOS à partir de chaque instance $((F_1, F_2, \ldots, F_n), k)$ de Max k-HS. Ensuite, dans la section 4.5.2.2, nous montrons l'équivalence des assertions (A_N) et (B_N).

4.5.2.1 Construction d'une instance de RSOS à partir d'une instance de Max k-HS

Soit $((F_1, F_2, \ldots, F_n), k)$ une instance de Max k-HS. Posons $m := \#(F_1 \cup F_2 \cup \ldots \cup F_n)$. Quitte à numéroter les éléments de $F_1 \cup F_2 \cup \ldots \cup F_n$, on peut, sans perte de généralité, supposer

$$F_1 \cup F_2 \cup \ldots \cup F_n = [1, m]$$

mais aussi $k < m$. En effet, si l'on a $k \geq m$ alors on construit une solution optimale C^* de Max k-HS pour l'instance $((F_1, F_2, \ldots, F_n), k)$ en prenant pour C^* un ensemble de cardinal k contenant $[1, m]$.

Pour chaque $j \in [1, n]$, on encode l'ensemble F_j à l'aide de la similarité s_j construite de la manière décrite ci-après.

• Notons $q_j(u) := \mathfrak{r}_{mn}((j-1)m + u)$ pour tout $u \in \mathbb{Z}$ (voir la définition 4.4 p. 102). On calcule

$$Q_j := \{q_j(u) : u \in F_j\}\ .$$

Remarque 4.6

(*i*). *Considérons la fonction qui, à tout $(u, j) \in \mathbb{Z} \times \mathbb{Z}$, associe $q_j(u)$. Cette fonction induit une bijection de $[1, m] \times [1, n]$ dans la règle de Golomb $\mathfrak{R}_{mn}$.*

(*ii*). *On déduit du point précédent que les ensembles Q_1, Q_2, …, Q_n sont des parties deux à deux disjointes de $\mathfrak{R}_{mn}$.*

• Posant $M := m^3 n^3$ on a $Q_j \subseteq \mathfrak{R}_{mn} \subseteq [1, M]$. On construit les M similarités de longueur m^3

$$s_{j,1},\ s_{j,2},\ s_{j,3},\ \ldots,\ s_{j,M}$$

données par :

1. pour chaque $u \in F_j$, $s_{j,q_j(u)} := \mathfrak{t}_u^m$ (voir la définition 4.7 p. 105), et
2. pour chaque $i \in [1, M] \setminus Q_j$, $s_{j,i} := \mathsf{0}^{m^3}$.

Notons que, s'il existait u, $u' \in [1,m]$ avec $u \neq u'$ et $q_j(u) = q_j(u')$, alors il faudrait affecter simultanément $\mathfrak{t}_u^m$ et $\mathfrak{t}_{u'}^m$ à $s_{j,q_j(u)} = s_{j,q_j(u')}$ d'après le premier point de la définition ci-dessus. Heureusement, d'après le point (i) de la remarque 4.6 p. 110, ce problème ne peut pas se présenter.

• On <u>encode</u> l'ensemble F_j par la similarité

$$s_j := s_{j,1}s_{j,2}s_{j,3}\dots s_{j,M}\mathtt{0}^{j-1}\,.$$

Les mots $\mathfrak{t}_u^m$ (pour $u \in F_j$) apparaissent dans s_j, dans l'ordre des u croissants, aux positions de la forme $m^3(i-1)$ avec $i \in Q_j$. Ils sont séparés par des blocs de la forme $\mathtt{0}^{m^3 l}$ avec $l \geq 0$ (en fait $l \geq 1$ comme le garantit la remarque 4.8 p. 112). La similarité s_j est de longueur $m^3M + j - 1$. L'adjonction du bloc $\mathtt{0}^{j-1}$ à la fin de s_j n'a pour fonction que de permettre la remarque suivante.

Remarque 4.7 *Les similarités s_1, s_2, …, s_n sont (de longueurs) deux à deux distinctes.*

• Enfin, on <u>construit</u> l'instance (ϖ, S) de RSOS avec :

$$\varpi := m - k \qquad \text{et} \qquad S := \{s_1, s_2, \dots, s_n\}\,.$$

Notons qu'on a bien $\varpi \geq 1$ car on a supposé $k < m$.

Cette construction est réalisable en temps polynomial à partir de l'instance $((F_1, F_2, \dots, F_n), k)$ de Max k-HS.

Le tableau suivant résume comment les instances de Max k-HS sont encodés par des instances de RSOS :

Max k-HS	RSOS
l'instance $((F_1, F_2, \dots, F_n), k)$	l'instance (ϖ, S)
l'ensemble F_j	la similarité $s_j \in S$
u est un élément de F_j	$\mathfrak{t}_u^m$ est une sous-chaîne de s_j
l'ensemble C intersecte F_j	la graine $\mathfrak{g}_{[1,m]\setminus C}^m$ détecte s_j
C est de cardinal k	$\mathfrak{g}_{[1,m]\setminus C}^m$ est de poids au moins ϖ

.

Notons que les deux dernières lignes du tableau ci-dessus, c'est-à-dire la correspondance qu'il existe entre les solutions de RSOS et celles de Max k-HS, sont explicitées dans la section suivante.

4.5.2.2 Correction de notre réduction

Dans cette section, nous cherchons à montrer que les assertions (A_N) et (B_N) sont équivalentes. Nous commençons par montrer que, pour tout entier $N \geq 0$, (B_N) implique (A_N). Ensuite, contrairement à ce que nous avions laissé entendre au début de la preuve du théorème 4.4, nous prouvons seulement que (A_N) implique (B_N) pour tout entier N au moins égal à 3. Nous laissons au lecteur le soin de vérifier que cet "effet de bord" de notre gadget n'affecte en rien la validité de notre preuve du théorème 4.4.

Montrons que (B_N) implique (A_N) pour tout $N \geq 0$. Supposons (B_N) : il existe un ensemble C de cardinal k tel que l'ensemble $J := \{j \in [1,n] : C \cap F_j \neq \emptyset\}$ soit de cardinal au moins N.

Posons $g := \mathfrak{g}_{[1,m]\setminus C}^m$ (voir définition 4.6 p. 104).

• Considérons un élément $j \in J$. On peut alors trouver un élément u dans l'intersection de C et F_j.
  – D'une part, on a $u \notin [1,m] \setminus C$ donc, par le lemme 4.10 p. 105, g détecte $\mathfrak{t}_u^m$.
  – D'autre part, $u \in F_j$ donc $\mathfrak{t}_u^m = s_{j,q_j(u)}$ est un sous-chaîne de s_j.

On en déduit que g détecte s_j d'après le point (iv) de la remarque 4.1 p. 91.

Comme $\{s_j : j \in J\}$ est de même cardinal que J (remarque 4.7 p. 111), g détecte au moins N similarités appartenant à S.

• De la remarque 4.5 p. 104, on déduit que g est de poids $\#([1,m] \setminus C) \geq m - k = \varpi$. Quitte à remplacer suffisamment de `#` dans g par des `-`, on peut supposer que g est une graine de poids exactement ϖ.

On a ainsi démontré (A_N), ce qu'on voulait à ce stade.

Montrons que (A_N) implique (B_N) pour tout $N \geq 3$. La preuve de cette implication s'appuie sur la remarque et les deux lemmes (4.12 et 4.13) ci-après.

Remarque 4.8 *L'ensemble $\mathfrak{R}_{mn}$ ne contient pas deux entiers consécutifs.*
En effet, pour tout entier $x \geq 1$, on a

$$\mathfrak{r}_{mn}(x+1) = \mathfrak{r}_{mn}(x) + (mn)^2 + 2x + 1 > \mathfrak{r}_{mn}(x) + 1\,.$$

Lemme 4.12 *Soient $j \in [1,n]$ et g une graine de longueur au plus m^3 détectant s_j.*
Alors, il existe $u \in F_j$ tel que g détecte $\mathfrak{t}_u^m$.

Preuve. Il existe une sous-chaîne f de s_j avec $|f| = |g|$ telle que g détecte f. Comme f et g sont de longueur au plus m^3 et comme les blocs $s_{j,i}$ (pour $i \in [1,M]$) constituant s_j sont de longueur m^3, il existe $i \in [2,M]$ tel que f soit une sous-chaîne de $s_{j,i-1}s_{j,i}$.

Supposons (absurde) que g ne détecte ni $s_{j,i-1}$, ni $s_{j,i}$. Alors, f n'est sous-chaîne d'aucun de ces deux mots donc la première et la dernière lettre de f apparaissent respectivement dans $s_{j,i-1}$ et $s_{j,i}$. Comme g commence et finit par un `#`, f commence et finit par un `1`. Par conséquent, $s_{j,i-1}$ et $s_{j,i}$ sont distincts de $\mathtt{0}^{m^3}$ et par suite, Q_j contient $i-1$ et i : contradiction avec la remarque 4.8 p. 112.

Le raisonnement par l'absurde précédent garantit qu'il existe $h \in \{i-1, i\}$ tel que g détecte $s_{j,h}$. La similarité $s_{j,h}$ est donc distincte de $\mathtt{0}^{m^3}$ ce qui force $h \in Q_j$: il existe $u \in F_j$ tel que $h = q_j(u)$ donc g détecte $\mathfrak{t}_u^m = s_{j,h}$. Ceci termine la preuve du lemme 4.12. **Q.E.D.**

La preuve du lemme suivant clarifie le rôle des sous-ensembles deux à deux disjoints Q_1, Q_2, …, $Q_n \subseteq \mathfrak{R}_{mn}$ (point (ii) de la remarque 4.6 p. 110).

Lemme 4.13 *Soit g une graine de longueur $|g| > m^3$.*
Alors, g détecte au plus 2 similarités appartenant à S.

Preuve. Posons $\chi := (|g| - 1) \,/\, m^3$.

Dans un premier temps, soit $j \in [1,n]$ tel que g détecte s_j. Il existe $p_j \in \big[0, m^3M - |g|\big]$ tel que g détecte s_j à la position p_j. Considérons la sous-chaîne de s_j de même longueur que g commençant à la position p_j : sa première et sa dernière lettre sont des `1`. Ces deux `1` apparaissent respectivement dans s_{j,a_j} et s_{j,b_j} où $a_j := \big\lceil (p_j + 1) \,/\, m^3 \big\rceil$ et $b_j := \big\lceil (p_j + |g|) \,/\, m^3 \big\rceil$. Ainsi, s_{j,a_j} et s_{j,b_j} sont distincts de $\mathtt{0}^{m^3}$ donc on a :

$$a_j \in Q_j \quad \text{et} \quad b_j \in Q_j\,. \tag{4.16}$$

Par ailleurs, on obtient facilement l'encadrement

$$\chi - 1 < b_j - a_j < \chi + 1\,. \tag{4.17}$$

L'hypothèse faite sur la longueur de g garantit $\chi \geq 1$, d'où $b_j - a_j > 0$ et, par suite, on a

$$a_j \neq b_j\,. \tag{4.18}$$

Supposons maintenant (absurde) qu'il existe j_1, j_2, $j_3 \in [1,n]$ deux à deux distincts tels que g détecte les trois similarités s_{j_1}, s_{j_2} et s_{j_3}. L'encadrement (4.17) garantit que les trois entiers $b_{j_1} - a_{j_1}$, $b_{j_2} - a_{j_2}$ et $b_{j_3} - a_{j_3}$ sont strictement compris entre $\chi - 1$ et $\chi + 1$. Ils ne peuvent donc être deux à deux distincts. Quitte à permuter j_1, j_2 et j_3, on peut se placer dans le cas où

$$b_{j_1} - a_{j_1} = b_{j_2} - a_{j_2}\,. \tag{4.19}$$

D'après le point (ii) de la remarque 4.6 p. 110, Q_{j_1} et Q_{j_2} sont des parties de $\mathfrak{R}_{mn}$ donc l'assertion (4.16) garantit que a_{j_1}, b_{j_1}, a_{j_2} et b_{j_2} sont des éléments de $\mathfrak{R}_{mn}$. De plus, on a $a_{j_1} \neq b_{j_1}$ et $a_{j_2} \neq b_{j_2}$ par (4.18). Comme $\mathfrak{R}_{mn}$ est une règle de Golomb (lemme 4.7 p. 103), l'égalité (4.19) implique $a_{j_1} = a_{j_2}$ et $b_{j_1} = b_{j_2}$. Il en résulte une contradiction avec le fait que Q_{j_1} et Q_{j_2} sont disjoints (point (ii) de la remarque 4.6 p. 110). Ceci termine la preuve du lemme 4.13. **Q.E.D.**

Nous avons maintenant tous les outils pour montrer que (A_N) implique (B_N).

Supposons (A_N) : il existe une graine g de poids ϖ et une partie $J \subseteq [1,n]$ de cardinal N telles que g détecte la similarité s_j pour tout $j \in J$.

Comme on a $N \geq 3$, le lemme 4.13 p. 112 garantit que g est de longueur au plus m^3. Appliquons le lemme 4.12 p. 112. Pour tout $j \in J$, il existe $u_j \in F_j$ tel que g détecte $\mathfrak{t}^m_{u_j}$; le lemme 4.10 p. 105 garantit alors qu'il existe $X_j \subseteq [1,m]$ vérifiant $u_j \notin X_j$ et $g = \mathfrak{g}^m_{X_j}$. Par la remarque 4.5 p. 104, les X_j $(j \in J)$ sont tous de cardinal ϖ. Par le point (ii) du lemme 4.8 p. 104, ils sont tous égaux à un même ensemble X. Posant $C := [1,m] \setminus X$, on obtient un ensemble de cardinal $m - \#X = m - \varpi = k$. De plus, pour tout $j \in J$ on a $u_j \in C \cap F_j$. On en déduit $J \subseteq \{j \in [1,n] : C \cap F_j \neq \emptyset\}$, donc ce dernier ensemble a un cardinal au moins égal à $\#J = N$.

On a ainsi démontré (B_N), ce qui finit la preuve du théorème 4.4. **Q.E.D.**

4.6 Questions ouvertes

4.6.1 Problèmes de pavage

La preuve de la NP-complétude de SSC (théorème 4.1 p. 95) soulève naturellement plusieurs questions concernant le problème du pavage d'une forme finie unidimensionnelle.

Soit un entier $d \geq 1$. Soient une partie $F \subseteq \mathbb{Z}^d$ et un ensemble $\mathcal{P}$ de parties de $\mathbb{Z}^d$. On appellera F la *forme* et les éléments de $\mathcal{P}$ des *tuiles*. On dit que $\mathcal{P}$ *pave* F *(par translation)* lorsque $\left\{P + \mathbf{t} : (P, \mathbf{t}) \in \mathcal{P} \times \mathbb{Z}^d\right\}$ admet pour sous-ensemble une partition de F. On nomme d-Dimensional Finite Region Tiling (d-DFRT) le problème suivant : "*Étant donnés une partie finie $F \subseteq \mathbb{Z}^d$ et un ensemble fini $\mathcal{P}$ de parties finies de $\mathbb{Z}^d$, décider si l'ensemble de tuiles $\mathcal{P}$ pave la forme F par translation*".

Ce problème est trivialement dans NP. De plus, lorsqu'on se restreint à des tuiles de cardinal 2, d-DFRT se réduit au problème du *couplage parfait*, donc devient polynomial [31]. En revanche, posons $H_2 := \{(0,0),(1,0)\}$, $V_3 := \{(0,0),(0,1),(0,2)\}$ et $\mathcal{P}_{23} := \{H_2, V_3\}$: les tuiles H_2 et V_3 sont appelés respectivement *domino horizontal* et *triomino vertical*. La restriction de 2-DFRT aux instances $(F, \mathcal{P})$ vérifiant $\mathcal{P} = \mathcal{P}_{23}$ est NP-difficile [9]. Ainsi, décider si 2 tuiles de cardinal au plus 3 pavent une forme finie plane est NP-difficile.

Lorsqu'on se restreint à la dimension 1, la complexité de 1-DFRT est plus ardue à étudier. Examinons attentivement la preuve de notre théorème 4.1 p. 95. Dans cette preuve, on exhibe une réduction de Karp de X3C à SSC : toute instance H de X3C est transformée en une instance $(\mathcal{P}, n, k)$ de SSC. Or, on peut vérifier que l'application transformant H en $([1,n], \mathcal{P})$ est une réduction de Karp de X3C à 1-DFRT. Ainsi, 1-DFRT est NP-complet. Néanmoins, cette réduction ne permet pas de répondre aux deux questions suivantes qui restent ouvertes.

1. Le problème 1-DFRT reste-t-il NP-difficile si l'on se restreint à un nombre borné de tuiles ?
2. Le problème 1-DFRT reste-t-il NP-difficile si l'on se restreint à des tuiles de cardinaux bornés ?

4.6.2 Maximisation du poids

Sous l'hypothèse raisonnable ZPP $\subsetneq$ NP, notre corollaire 4.3 p. 107 garantit que MWLS n'admet pas d'algorithme d'approximation de borne $(\#S)^\delta$, quel que soit le réel $\delta < 0.5$. On conjecture qu'il n'existe *aucun* réel $\delta \geq 0$ tel que MWLS admette un algorithme d'approximation de borne $(\#S)^\delta$.

Par ailleurs, on ne peut faire en général aucune hypothèse sur la forme des similarités que l'on cherche à détecter. Ainsi, la variante suivante de MWLS est plus intéressante que le problème MWLS lui-même : "*Étant donnés deux entiers m et k avec $0 \leq k \leq m$, trouver une graine de poids maximal détectant* toutes *les (m,k)-similarités*". Ce problème est abordé mais n'est pas complètement résolu dans la littérature [37, 66].

4.6.3 Approximation de RSOS

D'une part, remarquons que RSOS admet un algorithme d'approximation trivial de borne $\#S$. En effet, supposons donnée une instance (ϖ, S) de RSOS.

– S'il existe une similarité $s \in S$ avec $|s|_1 \geq \varpi$ alors on retourne une graine de poids ϖ détectant s.

– Sinon, on retourne une graine quelconque de poids ϖ.

D'autre part, d'après notre théorème 4.4 p. 110, RSOS n'admet vraisemblablement pas d'algorithme d'approximation de borne constante ρ, quel que soit le réel ρ vérifiant $1 \leq \rho < \frac{e}{e-1}$. L'existence d'un algorithme d'approximation pour RSOS de borne constante supérieure ou égale à $\frac{e}{e-1}$ reste ouverte.

Bibliographie

[1] B. Alberts, D. Bray, J. Lewis, M. Raff, K. Roberts, and J. D. Watson. *Molecular Biology of the Cell*. Garland, New York, 1983. pages 59

[2] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic Local Alignment Search Tool. *Journal of Molecular Biology*, 215(3) :403–410, 1990. pages 6, 86

[3] A. Amir, M. Lewenstein, and E. Porat. Faster algorithms for string matching with k mismatches. *Journal of Algorithms*, 50(2) :257–275, 2004. pages 88

[4] A. Apostolico and Z. Galil, editors. *Pattern Matching Algorithms*. Oxford University Press, 1997. pages 20, 24

[5] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation*. Springer-Verlag, second edition, 2003. pages 15, 16, 17

[6] W. C. Babcock. Intermodulation interference in radio systems. *Bell System Technical Journal*, 32(1) :63–73, 1953. pages 102

[7] V. Bafna, E. L. Lawler, and P. A. Pevzner. Approximation algorithms for multiple sequence alignment. *Theoretical Computer Science*, 182(1–2) :233–244, 1997. pages 36, 37

[8] A. Bateman, editor. *Database Issue*, volume 33 of *Nucleic Acids Research*. Oxford University Press, 2005. pages 25

[9] D. Beauquier, M. Nivat, E. Rémila, and M. Robson. Tiling figures of the plane with two bars. *Computational Geometry*, 5 :1–25, 1995. pages 113

[10] A. Ben-Dor, G. Lancia, J. Perone, and R. Ravi. Banishing bias from consensus sequences. In *Proceedings of the 8th Annual Symposium on Combinatorial Pattern Matching (CPM'97)*, volume 1264 of *Lecture Notes in Computer Science*, pages 247–261. Springer-Verlag, 1997. pages 29, 39, 58

[11] G. Benson. Tandem repeats finder: a program to analyze DNA sequences. *Nucleic Acids Research*, 27(2) :573–580, 1999. pages 88

[12] S. A. É. Bérard and É. Rivals. Comparison of minisatellites. *Journal of Computational Biology*, 10(3/4) :357–372, 2003. pages 24

[13] H. L. Bodlaender, R. G. Downey, M. R. Fellows, M. T. Hallett, and H. T. Wareham. Parameterized complexity analysis in computational biology. *Computer Applications in the Biosciences (CABIOS)*, 11(1) :49–57, 1995. pages 20, 70

[14] H. L. Bodlaender, R. G. Downey, M. R. Fellows, and H. T. Wareham. The parameterized complexity of sequence alignment and consensus. *Theoretical Computer Science*, 147(1–2) :31–54, 1995. pages 20, 77, 84

[15] P. Bonizzoni and G. Della Vedova. The complexity of multiple sequence alignment with SP-score that is a metric. *Theoretical Computer Science*, 259(1–2) :63–79, 2001. pages 36

[16] R. Boppana and M. M. Halldòrsson. Approximating maximum independent sets by excluding subgraphs. *BIT Numerical Mathematics*, 32(2) :180–196, 1992. pages 18

[17] H. Bunke and U. Buehler. Applications of approximate string matching to 2D shape recognition. *Pattern Recognition*, 26(12) :1797–1812, december 1993. pages 59

[18] S. Burkhardt, A. Crauser, P. Ferragina, H.-P. Lenhof, É. Rivals, and M. Vingron. q-gram Based Database Searching Using a Suffix Array (QUASAR). In *Proceedings of the 3rd Annual International Conference on Computational Molecular Biology (RECOMB'99)*, pages 77–83. ACM Press, 1999. pages 86

[19] S. Burkhardt and J. Kärkkäinen. Better filtering with gapped q-grams. *Fundamenta Informaticae*, 56(1–2) :51–70, 2003. pages 93

[20] A. Califano and I. Rigoutsos. FLASH: a fast look-up algorithm for string homology. In L. Hunter, D. Searls, and J. Shavlik, editors, *Proceedings of the 1st International Conference on Intelligent Systems for Molecular Biology (ISMB'93)*, pages 56–64. AAAI Press, 1993. pages 86

[21] J. Chen, I. A. Kanj, and W. Jia. Vertex Cover: Further observations and further improvements. *Journal of Algorithms*, 41(2) :280–301, 2001. pages 13

[22] K.-L. Chung. A fast algorithm for stereo matching. *Information Processing Letters*, 63(2) :57–61, 1997. pages 64

[23] S. A. Cook. The complexity of theorem-proving procedures. In *Conference Record of the 3rd Annual ACM Symposium on the Theory of Computing (STOC'71)*, pages 151–158, 1971. pages 6, 12

[24] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press and McGraw-Hill Book Company, second edition, 2001. pages 12, 14, 15, 17, 18, 20

[25] M. Crochemore, C. Hancart, and T. Lecroq. *Algorithmique du texte*. Vuibert, 2001. pages 24, 86

[26] M. Crochemore, G. M. Landau, and M. Ziv-Ukelson. A subquadratic sequence alignment algorithm for unrestricted scoring matrices. *SIAM Journal on Computing*, 32(6) :1654–1673, 2003. pages 24, 84

[27] C. de la Higuera and F. Casacuberta. Topology of strings: Median string is NP-complete. *Theoretical Computer Science*, 230(1–2) :39–48, 2000. pages 38, 39, 40, 43

[28] M. Dell'Amico, F. Maffioli, and S. Martello, editors. *Annotated Bibliographies in Combinatorial Optimization*. John Wiley and Sons, 1997. pages 7

[29] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999. pages 7, 13, 14, 15, 18

[30] R. C. Edgar. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, 32(5) :1792–1797, 2004. pages 25

[31] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17(3) :449–467, 1965. pages 113

[32] I. Elias. Settling the intractability of multiple alignment. In *Proceedings of The 14th Annual International Symposium on Algorithms and Computation (ISAAC'03)*, volume 2906 of *Lecture Notes in Computer Science*, pages 352–363. Springer-Verlag, 2003. pages 36, 38, 39, 43, 57, 58

[33] L. Engebretsen and J. Holmerin. Towards optimal lower bounds for clique and chromatic number. *Theoretical Computer Science*, 299(1–3) :537–584, 2003. pages 19

[34] P. A. Evans and A. D. Smith. Complexity of approximating closest substring problems. In *Proceedings of the 14th International Symposium on Foundations of Complexity Theory (FCT'2003)*, volume 2751 of *Lecture Notes in Computer Science*, pages 210–221. Springer-Verlag, 2003. pages 34

[35] P. A. Evans, A. D. Smith, and H. T. Wareham. The parameterized complexity of p-center approximate substring problems. Technical report TR01-149, Faculty of Computer Science, University of New Brunswick, 2001. pages 33

[36] P. A. Evans, A. D. Smith, and H. T. Wareham. On the complexity of finding common approximate substrings. *Theoretical Computer Science*, 306(1–3) :407–430, 2003. pages 33

[37] M. Farach-Colton, G. M. Landau, S. Cenk Sahinalp, and D. Tsur. Optimal spaced seeds for faster approximate string matching. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP'05)*, volume 3580 of *Lecture Notes in Computer Science*, pages 1251–1262. Springer-Verlag, 2005. pages 90, 93, 113

[38] U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the Association for Computing Machinery*, 45(4) :634–652, 1998. pages 108

[39] M. R. Fellows, J. Gramm, and R. Niedermeier. On the parameterized intractability of closest substring and related problems. In *Proceedings of the 19th International Symposium on Theoretical Aspects of Computer Science (STACS'02)*, volume 2285 of *Lecture Notes in Computer Science*, pages 262–273. Springer-Verlag, 2002. pages 33, 34

[40] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co., 1979. pages 12, 13, 95

[41] L. Gąsieniec, J. Jansson, and A. Lingas. Approximation algorithms for hamming clustering problems. *Journal of Discrete Algorithms*, 2(2) :289–301, 2004. pages 34

[42] T. F. González. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38 :293–306, 1985. pages 18

[43] O. Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162(3) :705–708, 1982. pages 24

[44] J. Gramm, R. Niedermeier, and P. Rossmanith. Fixed-parameter algorithms for closest string and related problems. *Algorithmica*, 37(1) :25–42, 2003. pages 33, 34, 58

[45] D. Gusfield. *Algorithms on Strings, Trees, and Sequences*. Computer Science and Computational Biology. Cambridge University Press, 1997. pages 20, 24, 28, 38, 39, 58, 64, 87, 88

[46] M. M. Halldòrsson. Approximations of weighted independent set and hereditary subset problems. *Journal of Graph Algorithms and Applications*, 4(1), 2000. pages 18, 20, 67, 71, 72

[47] J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182 :105–142, 1999. pages 19, 67, 77, 84, 85, 107

[48] D. S. Hirschberg. Algorithms for the longest common subsequence problem. *Journal of the Association for Computing Machinery*, 24(4) :664–675, 1977. pages 24

[49] D. S. Hochbaum, editor. *Approximation Algorithms for* NP*-Hard Problems*. PWS Publishing Company, 1996. pages 108

[50] D. S. Hochbaum and D. B. Shmoys. A best possible approximation algorithm for the k-center problem. *Mathematics of Operations Research*, 10(2) :180–184, 1985. pages 18

[51] T. J. Hubbard, A. M. Lesk, and A. Tramontano. Gathering them in to the fold. *Nature Structural Biology*, 3(4) :313, 1996. pages 25

[52] O. H. Ibarra and C. E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the Association for Computing Machinery*, 22(4) :463–468, 1975. pages 17

[53] S. Y. Itoga. The string merging problem. *BIT Numerical Mathematics*, 21(1) :20–30, 1981. pages 20, 68

[54] T. Jiang, E. L. Lawler, and L. Wang. Approximation algorithms for tree alignment with a given phylogeny. *Algorithmica*, 16(3) :302–315, 1996. pages 38

[55] T. Jiang and M. Li. On the approximation of shortest common supersequences and longest common subsequences. *SIAM Journal on Computing*, 24(5) :1122–1139, 1995. pages 20, 66, 67, 71, 77, 79, 85

[56] Y. Jiao, J. Xu, and M. Li. On the k-closest substring and k-consensus pattern problems. In *Proceedings of the 15th Annual Symposium on Combinatorial Pattern Matching (CPM'04)*, volume 3109 of *Lecture Notes in Computer Science*, pages 130–144. Springer-Verlag, 2004. pages 33, 34

[57] D. S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9(3) :256–278, 1974. pages 7

[58] W. Just. Computational complexity of multiple sequence alignment with SP-score. *Journal of Computational Biology*, 8(6) :615–623, 2001. pages 36, 37, 58

[59] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972. pages 6, 18

[60] J. D. Kececioglu. The maximum trace problem in multiple sequence alignment. In *Proceedings of the 4th Annual Symposium on Combinatorial Pattern Matching (CPM'93)*, volume 684 of *Lecture Notes in Computer Science*, pages 106–119. Springer-Verlag, 1993. pages 66

[61] D. E. Knuth, J. H. Morris Jr., and V. R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2) :323–350, 1977. pages 88

[62] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*, volume 21 of *Algorithms and Combinatorics*. Springer-Verlag, second edition, 2002. pages 11

[63] M. Krivelevich, R. Nathaniel, and B. Sudakov. Approximating coloring and maximum independent sets in 3-uniform hypergraphs. *Journal of Algorithms*, 41(1) :99–113, 2001. pages 18

[64] M. Krivelevich and B. Sudakov. Approximate coloring of uniform hypergraphs. *Journal of Algorithms*, 49(1) :2–12, 2003. pages 75

[65] G. Kucherov, L. Noé, and M. Roytberg. Multi-seed lossless filtration. In *Proceedings of the 15h Annual Symposium on Combinatorial Pattern Matching (CPM'04)*, volume 3109 of *Lecture Notes in Computer Science*, pages 297–310. Springer-Verlag, 2004. pages 93

[66] G. Kucherov, L. Noé, and M. Roytberg. Multiseed lossless filtration. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2(1) :51–61, 2005. pages 93, 94, 113

[67] J. Kevin Lanctot, M. Li, B. Ma, S. Wang, and L. Zhang. Distinguishing string selection problems. *Information and Computation*, 185(1) :41–55, 2003. pages 33, 34

[68] G. M. Landau, E. W. Myers, and J. P. Schmidt. Incremental string comparison. *SIAM Journal on Computing*, 27(2) :557–582, 1998. pages 64, 65

[69] G. M. Landau and U. Vishkin. Fast parallel and serial approximate string matching. *Journal of Algorithms*, 10(2) :157–169, 1989. pages 88

[70] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Cybernetics and Control Theory*, 10(8) :707–710, 1966. pages 9, 22

[71] M. Li, B. Ma, D. Kisman, and J. Tromp. PatternHunter II: Highly sensitive and fast homology search. *Journal of Bioinformatics and Computational Biology*, 2(3) :417–439, 2004. pages 86, 94

[72] M. Li, B. Ma, and L. Wang. Finding similar regions in many strings. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC'99)*, pages 473–482. ACM Press, 1999. pages 34

[73] M. Li, B. Ma, and L. Wang. Finding similar regions in many sequences. *Journal of Computer and System Sciences*, 65(1) :73–96, 2002. pages 33, 38, 58

[74] M. Li, B. Ma, and L. Wang. On the closest string and substring problems. *Journal of the Association for Computing Machinery*, 49(2) :157–171, 2002. pages 34

[75] D. J. Lipman, S. F. Altschul, and J. D. Kececioglu. A tool for multiple sequence alignment. *Proceedings of the National Academy of Sciences of the United States of America*, 86(12) :4412–4415, 1989. pages 25

[76] M. Lothaire, editor. *Combinatorics on Words*. Cambridge Mathematical Library. Cambridge University Press, second edition, 1997. pages 9, 61

[77] M. Lothaire, editor. *Algebraic Combinatorics on Words*. Number 90 in Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2002. pages 9, 47, 48, 61

[78] M. Lothaire, editor. *Applied Combinatorics on Words*. Number 105 in Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2005. pages 84

[79] B. Ma, J. Tromp, and M. Li. Patternhunter: faster and more sensitive homology search. *Bioinformatics*, 18(3) :440–445, 2002. pages 86, 90, 92

[80] M. Maes. On a cyclic string-to-string correction problem. *Information Processing Letters*, 35(2) :73–78, 1990. pages 60, 64, 65

[81] D. Maier. The complexity of some problems on subsequences and supersequences. *Journal of the Association for Computing Machinery*, 25(2) :322–336, 1978. pages 20, 41, 66, 70

[82] B. Manthey. Non-approximability of weighted multiple sequence alignment. *Theoretical Computer Science*, 296(1) :179–192, 2003. pages 36

[83] A. Marzal and G. Peris. Normalized cyclic edit distances: An efficient algorithm. In *Proceedings of the 10th Conferencia de la Asociación Española Para la Inteligencia Artificial (CAEPIA'03)*, volume 3040 of *Lecture Notes in Computer Science*, pages 435–444. Springer-Verlag, 2004. Revised Selected Paper. pages 59

[84] W. J. Masek and M. S. Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20(1) :18–31, 1980. pages 87

[85] M. Middendorf. More on the complexity of common superstring and supersequence problems. *Theoretical Computer Science*, 125(2) :205–228, 1994. pages 60, 66

[86] W. Miller and E. W. Myers. Sequence comparison with concave weighting functions. *Bulletin of Mathematical Biology*, 50(2) :97–120, 1988. pages 24

[87] R. A. Mollineda, E. Vidal, and F. Casacuberta. Cyclic sequence alignments: Approximate versus optimal techniques. *International Journal of Pattern Recognition and Artificial Intelligence*, 16(3) :291–299, 2002. pages 60

[88] E. W. Myers. An $O(nd)$ difference algorithm and its variations. *Algorithmica*, 1(2) :251–266, 1986. pages 87

[89] G. Navarro and M. Raffinot. *Flexible Pattern Matching in Strings: Practical on-Line Search Algorithms for Texts and Biological Sequences*. Cambridge University Press, 2002. pages 86

[90] F. Nicolas and É. Rivals. Complexities of the centre and median string problems. In *Proceedings of the 14th Annual Symposium on Combinatorial Pattern Matching (CPM'03)*, volume 2676 of *Lecture Notes in Computer Science*, pages 315–327. Springer-Verlag, 2003. pages 43

[91] F. Nicolas and É. Rivals. Hardness of optimal spaced seed design. In A. Apostolico, M. Crochemore, and K. Park, editors, *Proceedings of the 16th Annual Symposium on Combinatorial Pattern Matching (CPM'05)*, volume 3537 of *Lecture Notes in Computer Science*, pages 144–155. Springer-Verlag, 2005. pages 95

[92] L. Noé and G. Kucherov. YASS: enhancing the sensitivity of DNA similarity search. *Nucleic Acids Research*, 33 (web-server issue) :W540–W543, 2005. pages 86

[93] R. Ostrovsky and Y. Rabani. Polynomial-time approximation schemes for geometric min-sum median clustering. *Journal of the Association for Computing Machinery*, 49(2) :139–156, 2002. pages 33

[94] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley Professional, 1994. pages 13, 16, 19, 93

[95] W. R. Pearson and D. J. Lipman. Improved tools for biological sequence comparison. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 85, pages 2444–2448, 1988. pages 6, 86

[96] G. Peris and A. Marzal. Fast cyclic edit distance computation with weighted edit costs in classification. In *Proceedings of the 16th International Conference on Pattern Recognition (ICPR'02)*, volume IV, pages 184–187. IEEE Computer Society, 2002. pages 59, 60

[97] K. Pietrzak. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *Journal of Computer and System Sciences*, 67(4) :757–771, 2003. pages 20, 41, 43, 66, 70

[98] K.-J. Räihä and E. Ukkonen. The shortest common supersequence problem over binary alphabet is NP-complete. *Theoretical Computer Science*, 16(2) :187–198, 1981. pages 66

[99] R. Ravi and J. D. Kececioglu. Approximation algorithms for multiple sequence alignment under a fixed evolutionary tree. *Discrete Applied Mathematics*, 88(1–3) :355–366, 1998. pages 37, 38, 52

[100] D. Sankoff. Minimal mutation trees of sequences. *SIAM Journal on Applied Mathematics*, 28(1) :35–42, 1975. pages 38

[101] D. Sankoff and J. B. Kruskal, editors. *Time Warps, String Edits and Macromolecules: the Theory and Practice of Sequence Comparison*. CSLI Publications, second edition, 1999. pages 19, 20, 37, 38

[102] J. P. Schmidt. All highest scoring paths in weighted grid graphs and its application to finding all approximate repeats in strings. *SIAM Journal on Computing*, 27(4) :972–992, 1998. pages 60, 64, 65

[103] G. D. Schuler, S. F. Altschul, and D. J. Lipman. A workbench for multiple alignment construction and analysis. *Proteins: Structure, Function, and Genetics*, 9(3) :180–190, 1991. pages 25

[104] J. S. Sim and K. Park. The consensus string problem for a metric is NP-complete. In *Proceedings of the Tenth Australasian Workshop on Combinatorial Algorithms (AWOCA'99)*, pages 107–113, 1999. pages 38, 43

[105] J. S. Sim and K. Park. The consensus string problem for a metric is NP-complete. *Journal of Discrete Algorithms*, 1(1) :111–117, 2003. pages 38

[106] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1) :195–197, 1981. pages 87

[107] Y. Sortais and R. Sortais. *La géométrie du triangle*. Hermann, second edition, 1997. pages 32

[108] G. A. Stephen. *String Searching Algorithms*, volume 3 of *Lecture Notes Series On Computing*. World Scientific, 1994. pages 20

[109] P. Tauvel. *Algèbre pour l'agrégation interne*. Masson, 1996. pages 30

[110] J. D. Thompson, D. G. Higgins, and T. J. Gibson. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22(22) :4673–4680, 1994. pages 25

[111] V. G. Timkovskii. Complexity of common subsequence and supersequence problems and related problems. *Cybernetics*, 25 :565–580, 1990. pages 66

[112] A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. In *Proceedings of the London Mathematical Society*, volume 42 of *2*, pages 230–265, 1936. pages 6

[113] L. Wang and D. Gusfield. Improved approximation algorithms for tree alignment. *Journal of Algorithms*, 25(2) :255–273, 1997. pages 38, 58

[114] L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *Journal of Computational Biology*, 1(4) :337–348, 1994. pages 36, 38, 39, 58, 71

[115] L. Wang, T. Jiang, and D. Gusfield. A more efficient approximation scheme for tree alignment. *SIAM Journal on Computing*, 30(1) :283–299, 2000. pages 37, 38

Annexe A

Preuve du théorème 2.6 page 56

Rappelons l'énoncé de notre théorème 2.6. On considère les égalités :

$$B[\mathbf{p} - \mathbf{e}_j][\mathbf{d} - \pi(\varepsilon, w_j[p_j])\mathbf{e}_j] = \texttt{oui} \tag{E_j}$$

pour tout $j \in I$, et

$$B\left[\mathbf{p} - \sum_{i\in J} \mathbf{e}_i\right]\left[\mathbf{d} - \sum_{i\in[1,m]\setminus J} \pi(a,\varepsilon)\mathbf{e}_i - \sum_{i\in J} \pi(a, w_i[p_i])\mathbf{e}_i\right] = \texttt{oui} \tag{$F_{a,J}$}$$

pour tout $a \in \Sigma$ et tout $J \subseteq I$. Nous devons démontrer que $B[\mathbf{p}][\mathbf{d}] = \texttt{oui}$ si et seulement si l'une au moins des 2 assertions suivantes est vérifiée :

1. $\exists j \in I$ tel que (E_j) soit vrai
   ou
2. $\exists a \in \Sigma$ et $\exists J \subseteq I$ avec $J \neq \emptyset$ tels que $(F_{a,J})$ soit vrai.

Tout d'abord, numérotons les entrées du m-uplet $\mathbf{d}$: $\mathbf{d} = (d_1, d_2, \ldots, d_m)$. Dans les deux points suivants, nous traduisons à l'aide de la définition de $B[\,\cdot\,][\,\cdot\,]$, les égalités (E_j) et $(F_{a,J})$ en terme d'inégalités faisant intervenir la distance d'édition.

• Pour tout $g \in \Sigma^\star$ et tout $j \in I$, on considère les deux assertions

$$d_{\mathrm{E}}^{\pi}\left(g, w_j^{(p_j-1)}\right) \leq d_j - \pi(\varepsilon, w_j[p_j]) \tag{$E^+_{g,j}$}$$

et

$$\forall i \in [1,m] \qquad i \neq j \Rightarrow d_{\mathrm{E}}^{\pi}\left(g, w_i^{(p_i)}\right) \leq d_i\,. \tag{$E^-_{g,j}$}$$

L'égalité (E_j) est vraie si et seulement s'il existe $g \in \Sigma^\star$ satisfaisant à la fois $(E^+_{g,j})$ et $(E^-_{g,j})$.

• Pour tout $g \in \Sigma^\star$, tout $a \in \Sigma$ et tout $J \subseteq I$, on considère les deux assertions

$$\forall i \in J \qquad d_{\mathrm{E}}^{\pi}\left(g, w_i^{(p_i-1)}\right) \leq d_i - \pi(a, w_i[p_i]) \tag{$F^+_{g,a,J}$}$$

et

$$\forall i \in [1,m] \setminus J \qquad d_{\mathrm{E}}^{\pi}\left(g, w_i^{(p_i)}\right) \leq d_i - \pi(a,\varepsilon)\,. \tag{$F^-_{g,a,J}$}$$

L'égalité $(F_{a,J})$ est vraie si et seulement s'il existe $g \in \Sigma^\star$ satisfaisant à la fois $(F^+_{g,a,J})$ et $(F^-_{g,a,J})$.

A.1 Montrons que la condition est suffisante.

Il s'agit de montrer les deux faits suivants.

1. Si $\exists j \in I$ satisfaisant (E_j), alors $B[\mathbf{p}][\mathbf{d}] = \texttt{oui}$. Cette assertion est prouvée dans la section A.1.1.
2. Si $\exists a \in \Sigma$ et $\exists J \subseteq I$ avec $J \neq \emptyset$ satisfaisant $(F_{a,J})$, alors $B[\mathbf{p}][\mathbf{d}] = \texttt{oui}$. Cette assertion est prouvée dans la section A.1.2.

Nous commençons par montrer une propriété générale de la distance d'édition.

Lemme A.1 *Pour tous x, $y \in \Sigma^\star$ et tous a, $b \in \Sigma$, on a :*

(i). $d_{\mathrm{E}}^{\pi}(xa, yb) \leq d_{\mathrm{E}}^{\pi}(x, y) + \pi(a, b)$,

(ii). $d_{\mathrm{E}}^{\pi}(xa, y) \leq d_{\mathrm{E}}^{\pi}(x, y) + \pi(a, \varepsilon)$, *et*

(iii). $d_{\mathrm{E}}^{\pi}(x, yb) \leq d_{\mathrm{E}}^{\pi}(x, y) + \pi(\varepsilon, b)$.

Preuve. Soit :

$$\begin{bmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_n \\ \beta_1 & \beta_2 & \cdots & \beta_n \end{bmatrix}$$

un alignement de (x, y) de π-score minimum $d_{\mathrm{E}}^{\pi}(x, y)$. Alors,

$$\begin{bmatrix} \alpha_1 & \cdots & \alpha_n & a \\ \beta_1 & \cdots & \beta_n & b \end{bmatrix}, \quad \begin{bmatrix} \alpha_1 & \cdots & \alpha_n & a \\ \beta_1 & \cdots & \beta_n & \varepsilon \end{bmatrix} \quad \text{et} \quad \begin{bmatrix} \alpha_1 & \cdots & \alpha_n & \varepsilon \\ \beta_1 & \cdots & \beta_n & b \end{bmatrix}$$

sont des alignements de (xa, xb), (xa, y) et (x, yb) respectivement. Ces alignements sont de π-scores :

$$d_{\mathrm{E}}^{\pi}(x, y) + \pi(a, b)\,, \quad d_{\mathrm{E}}^{\pi}(x, y) + \pi(a, \varepsilon) \quad \text{et} \quad d_{\mathrm{E}}^{\pi}(x, y) + \pi(\varepsilon, b)\,.$$

On en déduit les trois inégalités recherchées. **Q.E.D.**

A.1.1 Supposons qu'il existe un $j \in I$ vérifiant (E_j) et montrons que $B[\mathbf{p}][\mathbf{d}] = \texttt{oui}$.

Cela signifie qu'il existe $g \in \Sigma^\star$ satisfaisant à la fois $(E^+_{g,j})$ et $(E^-_{g,j})$. On peut alors écrire :

$$d_{\mathrm{E}}^{\pi}\left(g, w_j^{(p_i)}\right) \leq d_{\mathrm{E}}^{\pi}\left(g, w_j^{(p_j-1)}\right) + \pi(\varepsilon, w_j[p_j]) \leq d_j\,,$$

la première inégalité se déduisant du point (iii) du lemme A.1 p. 122 et la seconde de $(E^+_{g,j})$. La conjonction de $(E^-_{g,j})$ et de l'inégalité $d_{\mathrm{E}}^{\pi}\left(g, w_j^{(p_i)}\right) \leq d_j$ qui vient d'être montrée fournit :

$$\forall i \in [1, m] \qquad d_{\mathrm{E}}^{\pi}\left(g, w_i^{(p_i)}\right) \leq d_i\,.$$

On a ainsi montré que $B[\mathbf{p}][\mathbf{d}] = \texttt{oui}$, ce qu'on voulait à ce stade.

A.1.2 Supposons qu'il existe $a \in \Sigma$ et $J \subseteq I$ vérifiant $(F_{a,J})$ et montrons que $B[\mathbf{p}][\mathbf{d}] = \texttt{oui}$.

Cela signifie qu'il existe $g \in \Sigma^\star$ satisfaisant à la fois $(F^+_{g,a,J})$ et $(F^-_{g,a,J})$. On va montrer :

$$\forall i \in [1, m] \qquad d_{\mathrm{E}}^{\pi}\left(ga, w_i^{(p_i)}\right) \leq d_i\,. \tag{A.1}$$

• Soit $i \in J$. On peut écrire :

$$d_{\mathrm{E}}^{\pi}\left(ga, w_i^{(p_i)}\right) \leq d_{\mathrm{E}}^{\pi}\left(g, w_i^{(p_i-1)}\right) + \pi(a, w_i[p_i]) \leq d_i\,,$$

la première inégalité se déduisant du point (i) du lemme A.1 p. 122 et la seconde de $(F^+_{g,a,J})$.

• Soit $i \in [1,m] \setminus J$. Comme dans le point précédent, on déduit du point (ii) du lemme A.1 p. 122 et de $(F^-_{g,a,J})$ les majorations :

$$d^\pi_{\mathrm{E}}\left(ga, w_i^{(p_i)}\right) \leq d^\pi_{\mathrm{E}}\left(g, w_i^{(p_i)}\right) + \pi(a,\varepsilon) \leq d_i\,.$$

Les deux points ci-dessus constituent une preuve de (A.1), d'où $B[\mathbf{p}][\mathbf{d}] = \texttt{oui}$.

A.2 Montrons que la condition est nécessaire.

Supposons que $B[\mathbf{p}][\mathbf{d}] = \texttt{oui}$. Ceci signifie qu'il existe un mot $g \in \Sigma^\star$, que l'on peut supposer de longueur minimale, vérifiant :

$$\forall i \in [1,m] \qquad d^\pi_{\mathrm{E}}\left(g, w_i^{(p_i)}\right) \leq d_i\,. \tag{A.2}$$

Pour chaque $i \in [1,m]$, on peut alors trouver un alignement A_i de $\left(g, w_i^{(p_i)}\right)$ tel que le π-score de A_i soit d'au plus d_i. Rappelons que l'on a convenu que la matrice à 2 lignes et 0 colonne est l'unique alignement de $(\varepsilon,\varepsilon)$ (voir section 1.5.1). Pour chaque $i \in [1,m]$ avec $gw_i^{(p_i)} \neq \varepsilon$, l'alignement A_i possède au moins une colonne donc on peut noter :

– $\begin{bmatrix}\alpha_i\\ \beta_i\end{bmatrix}$ la colonne la plus à droite de A_i, et
– A'_i l'alignement obtenu à partir de A_i en effaçant sa colonne la plus à droite.

α_i est égal au mot vide ou à la dernière lettre de g et β_i est égal au mot vide ou à la dernière lettre de $w_i^{(p_i)}$. Notons que, pour tout $i \in I$, on a $w_i^{(p_i)} \neq \varepsilon$ donc les objets α_i, β_i et A'_i sont bien définis.

Lemme A.2 *Pour tout $i \in [1,m]$ avec $gw_i^{(p_i)} \neq \varepsilon$, A'_i est de π-score au plus $d_i - \pi(\alpha_i,\beta_i)$.*

Preuve. Notons d'_i le π-score de A'_i. Le π-score de A_i est égal à la somme des π-scores des alignements A'_i et $\left[\begin{smallmatrix}\alpha_i\\ \beta_i\end{smallmatrix}\right]$. Il en résulte $d'_i + \pi(\alpha_i,\beta_i) \leq d_i$ ou encore, $d'_i \leq d_i - \pi(\alpha_i,\beta_i)$. **Q.E.D.**

A.2.1 Supposons qu'il existe un $j \in I$ tel que $\alpha_j = \varepsilon$ et montrons (E_j).

Comme il apparaît au moins une lettre dans toute colonne d'alignement, on a $\beta_j \in \Sigma$. Il en résulte

$$\begin{bmatrix}\alpha_j\\ \beta_j\end{bmatrix} = \begin{bmatrix}-\\ w_j[p_j]\end{bmatrix}$$

donc A'_j est un alignement de $\left(g, w_j^{(p_j-1)}\right)$. Par suite, le π-score de A'_j est au moins égal à $d^\pi_{\mathrm{E}}\left(g, w_j^{(p_j-1)}\right)$. On en déduit

$$d^\pi_{\mathrm{E}}\left(g, w_j^{(p_j-1)}\right) \leq d_j - \pi(\varepsilon, w_j[p_j])$$

d'après le lemme A.2 p. 123. On a alors prouvé l'assertion $(E^+_{g,j})$ et, par ailleurs, (A.2) implique trivialement $(E^-_{g,j})$. On a ainsi montré montré l'égalité (E_j).

A.2.2 Supposons que, pour tout $i \in I$, on ait $\alpha_i \neq \varepsilon$ et montrons qu'il existe $a \in \Sigma$ et une partie non vide $J \subseteq I$ satisfaisant $(F_{a,J})$.

Comme on a $\mathbf{p} \neq (0,0,\ldots,0)$, on peut trouver un élément i_0 dans I. Posons $a := \alpha_{i_0}$ et $J := \{i \in I : \beta_i = w_i[p_i]\}$.

D'après l'hypothèse faite dans cette section, a est une lettre, et plus précisément, a est la dernière lettre de g : il existe $g' \in \Sigma^\star$ tel que $g = g'a$. En particulier, g est non vide, donc pour tout $i \in [1,m]$, A_i admet au moins une colonne et, par suite, α_i, β_i et A'_i sont bien définis.

Montrons $(F^{+}_{g',a,J})$**.** Soit $i \in J$. Comme J est une partie de I, l'hypothèse faite dans cette section garantit $\alpha_i \neq \varepsilon$. De plus, par définition de J, on a $\beta_i = w_i[p_i]$:

$$\begin{bmatrix} \alpha_i \\ \beta_i \end{bmatrix} = \begin{bmatrix} a \\ w_i[p_i] \end{bmatrix} .$$

Il en résulte que A'_i est un alignement de $\left(g', w_i^{(p_i-1)}\right)$, d'où

$$d^{\pi}_{\mathrm{E}}\left(g', w_i^{(p_i-1)}\right) \leq d_i - \pi(a, w_i[p_i])$$

d'après le lemme A.2. On a ainsi montré $(F^{+}_{g',a,J})$.

Montrons $(F^{-}_{g',a,J})$**.** Soit $i \in [1,m] \setminus J$. Remarquons que $\beta_i = \varepsilon$.

- En effet, si $i \in I$ alors on a $\beta_i \neq w_i[p_i]$, par définition de J.
- D'autre part, si $i \notin I$ alors on a $w_i^{(p_i)} = \varepsilon$ donc toutes les entrées de A_i situées sur sa ligne inférieure sont des $-$ et, en particulier, $\beta_i = \varepsilon$.

Comme il apparaît au moins une lettre dans toute colonne d'alignement, $\beta_i = \varepsilon$ force $\alpha_i = a$:

$$\begin{bmatrix} \alpha_i \\ \beta_i \end{bmatrix} = \begin{bmatrix} a \\ \varepsilon \end{bmatrix} .$$

Il en résulte que A'_i est un alignement de $\left(g', w_i^{(p_i)}\right)$, d'où

$$d^{\pi}_{\mathrm{E}}\left(g', w_i^{(p_i)}\right) \leq d_i - \pi(a, \varepsilon) \tag{A.3}$$

d'après le lemme A.2. On a ainsi montré $(F^{-}_{g',a,J})$.

D'après les deux paragraphes précédents, l'égalité $(F_{a,J})$ est vraie. Pour terminer la preuve de notre théorème 2.6, il reste encore à vérifier que l'ensemble J est non vide.

Montrons que J **est non vide.** Supposons (absurde) que l'on ait $J = \emptyset$. Alors, l'inégalité (A.3) est vraie pour tout $i \in [1,m]$. Or, $\pi(a,\varepsilon)$ est positif ou nul (hypothèse $(**)$ p. 54) donc on a $d_i - \pi(a,\varepsilon) \leq d_i$. Il en résulte :

$$\forall i \in [1,m] \qquad d^{\pi}_{\mathrm{E}}\left(g', w_i^{(p_i-1)}\right) \leq d_i$$

ce qui contredit le fait que g a été choisi comme un mot de longueur minimale satisfaisant (A.2). Ceci termine la preuve du théorème 2.6. **Q.E.D.**

Annexe B

Glossaire des problèmes

Dans cette annexe, sont recensés les principaux problèmes dont nous traitons.

Chapitre 1

Nom: Maximum Independent Set in Hypergraphs (MISH).
Instance: Un hypergraphe H.
Solution: Un indépendant I de H.
Mesure: Le cardinal de I, à maximiser.
Page: 18.
Notes: La notion d'indépendant est définie page 18. Le cardinal maximum pour un indépendant de l'hypergraphe H est noté $\alpha(H)$. La restriction de MISH aux hypergraphes r-uniformes est notée r-MISH.

Nom: Longest Common Subsequence (LCS).
Instance: Un langage fini et non vide X.
Solution: Un mot s tel que, pour tout $x \in X$, s est une sous-séquence de x.
Mesure: La longueur de s, à maximiser.
Page: 19.
Notes: La longueur maximale pour une sous-séquence commune du langage X est notée $\mathrm{lcs}(X)$.

Chapitre 2

Nom: d-Median où $d : U \times U \to \mathbb{Z}$ est une application quelconque.
Instance: Une partie finie $W \subseteq U$.
Solution: Un élément $\mu \in U$.
Mesure: $\sum_{w \in W} d(\mu, w)$, à minimiser.
Page: 26.
Notes: La mesure d'une solution optimale est notée $\mathrm{S}_d(W)$.

Nom: d-Center où $d : U \times U \to \mathbb{Z}$ est une application quelconque.
Instance: Une partie finie $W \subseteq U$.
Solution: Un élément $g \in U$.
Mesure: $\max_{w \in W} d(g, w)$, à minimiser.
Page: 29.
Notes: La mesure d'une solution optimale est notée $\mathrm{R}_d(W)$. Cette quantité est appelée rayon de W pour d.

Nom: LCS_D^0.
Instance: Un couple (X, k) où k est un entier naturel et où X est un langage fini et non vide dont tous les mots sont de longueur $2k$.
Question: Existe-t-il un mot s de longueur k tel que tout mot de X admette s pour sous-séquence ?
Page: 40.
Notes: Ce problème est une restriction du problème de décision associé à LCS.

Nom: ℓ-Weighted Common Subsequence (ℓ-WCS) où ℓ est une longueur pondérée sur $\Sigma^\star$, Σ désignant un alphabet.
Instance: Un langage fini et non vide $X \subseteq \Sigma^\star$ et un entier $k \geq 0$.
Question: Existe-t-il un mot s avec $\ell(s) = k$ tel que, pour tout $x \in X$, s soit une sous-séquence de x ?
Page: 47.
Notes: La notion de longueur pondérée est définie page 47. On nomme ℓ-WCS^0 la restriction de ℓ-WCS aux instances (X, k) telles que, pour tout $x \in X$, on ait $\ell(x) = 2k$.

Chapitre 3

Nom: Longest Commun Unoriented Subsequence (LCUS).
Instance: Un langage fini et non vide X.
Solution: Un mot s tel que, pour tout $x \in X$, s est une U-sous-séquence de x.
Mesure: La longueur de s, à maximiser.
Page: 64.
Notes: La notion de U-sous-séquence est définie page 62. La longueur maximale pour une U-sous-séquence commune du langage X est notée $\text{lcus}(X)$.

Nom: Longest Common Cyclic Subsequence (LCCS).
Instance: Un langage fini et non vide X.
Solution: Un mot s tel que, pour tout $x \in X$, s est une C-sous-séquence de x.
Mesure: La longueur de s, à maximiser.
Page: 64.
Notes: La notion de C-sous-séquence est définie page 62. La longueur maximale pour une C-sous-séquence commune du langage X est notée $\text{lccs}(X)$.

Nom: Longest Common Unoriented Cyclic Subsequence (LCUCS).
Instance: Un langage fini et non vide X.
Solution: Un mot s tel que, pour tout $x \in X$, s est une UC-sous-séquence de x.
Mesure: La longueur de s, à maximiser.
Page: 64.
Notes: La notion de UC-sous-séquence est définie page 62. La longueur maximale pour une UC-sous-séquence commune du langage X est notée $\text{lcucs}(X)$.

Chapitre 4

Nom: Non Detection.
Instance: Une graine g et deux entiers m, k vérifiant $0 \leq k \leq m$.
Question: Existe-t-il une (m, k)-similarité non détectée par g ?
Page: 93.
Notes: La taille de l'instance est $O(|g| + m)$ ce qui signifie que l'on suppose que les entiers m et k sont codés en unaire. Les notions de graine, de similarité et de détection sont définies dans la section 4.2.1.

Nom: Maximum Weight Lossless Seed (MWLS).
Instance: Un ensemble S de similarités.
Solution: Une graine g détectant toutes les similarités appartenant à S.
Mesure: Le poids de g, à maximiser.
Page: 94.
Notes: Néant.

Nom: Region Specific Optimal Seed (RSOS).
Instance: Un entier $\varpi \geq 1$ et un ensemble S de similarités.
Solution: Une graine g de poids ϖ.
Mesure: Le nombre de similarités appartenant à S et détectées par g, à maximiser.
Page: 95.
Notes: À ne pas confondre avec sa variante RSOSs définie également dans la section 4.2.2.3.

Nom: Exact Cover by 3-Sets (X3C).
Instance: Un hypergraphe 3-uniforme $H = (V, \mathcal{E})$.
Question: L'ensemble d'hyperarêtes $\mathcal{E}$ admet-il pour sous-ensemble une partition de l'ensemble de sommets V ?
Page: 95.
Notes: On veut décider de l'existence d'un sous-ensemble $\mathcal{P} \subseteq \mathcal{E}$ vérifiant : pour chaque $v \in V$, il existe un unique $E \in \mathcal{P}$ tel que $v \in E$.

Nom: Soapy Set Cover (SSC).
Instance: Un ensemble fini $\mathcal{P}$ de parties finies et non vides de $\mathbb{Z}$ et deux entiers n, $k \geq 1$.
Question: Existe-t-il $X_1, X_2, \ldots, X_k \in \mathcal{P}$ et $t_1, t_2, \ldots, t_k \in \mathbb{Z}$ tels que $(X_1 + t_1) \cup (X_2 + t_2) \cup \ldots \cup (X_k + t_k)$ contienne n entiers consécutifs ?
Page: 95.
Notes: On suppose que les entiers n, k et les éléments des éléments de $\mathcal{P}$ sont codés en unaire.

Nom: Max k-Cover.
Instance: Un hypergraphe $H = (V, \mathcal{E})$ et un entier $k \geq 0$.
Solution: Une partie $\mathcal{C} \subseteq \mathcal{E}$ de cardinal au plus k.
Mesure: Le cardinal de $\bigcup \mathcal{C}$.
Page: 108.
Notes: $\bigcup \mathcal{C}$ est l'ensemble des sommets de H couverts par $\mathcal{C}$, c'est-à-dire l'ensemble des $v \in V$ pour lesquels il existe $C \in \mathcal{C}$ tel que $v \in C$.

Nom: Maximum k-Hitting Set (Max k-HS).
Instance: Un n-uplet $(F_1, F_2, \ldots, F_n)$ d'ensembles finis et un entier $k \geq 0$.
Solution: Tout ensemble C de cardinal k.
Mesure: Le nombre des indices $j \in [1, n]$ pour lesquels $C \cap F_j$ est non vide.
Page: 108.
Notes: Néant.

Résumé

Dans ce mémoire, nous étudions la complexité algorithmique de plusieurs problèmes combinatoires concernant la comparaison de séquences biologiques. Nous nous plaçons successivement du point de vue de chacune des trois principales théories de la complexité algorithmique : la NP-complétude, l'approximabilité et la complexité paramétrique.

Dans un premier temps, nous considérons plusieurs formes du problème de l'extraction des motifs communs à un ensemble de séquences donné. Les motifs communs permettent, en pratique, de classifier les protéines grâce à leur structure primaire, par exemple en fabriquant des séquences consensus.

En particulier, le problème de la médiane (resp. du centre) pour la distance d'édition consiste à rechercher une séquence consensus minimisant la somme (resp. le maximum) des distances d'édition la séparant de chacune des séquences prises en entrée. Nous affinons les résultats connus sur la difficulté de chacun de ces deux problèmes : nous montrons, par exemple, qu'ils sont tous les deux W[1]-difficiles lorsqu'on les paramétrise par le nombre des séquences étudiées et ce, même dans le cas d'un alphabet binaire. Nous considérons également le problème de la plus longue sous-séquence commune. Ce problème a été exhaustivement étudié dans sa forme usuelle. Or, on trouve dans la nature des séquences d'ADN et d'ARN circulaires qu'il est utile de comparer. Dans ce mémoire, nous menons à bien la première étude du problème de la plus longue sous-séquence commune à plusieurs séquences circulaires et/ou non orientées.

Dans un second temps, nous considérons plusieurs problèmes liés à la recherche de similarités approchées entre séquences biologiques. C'est dans ce domaine que l'application de l'informatique à la biologie moléculaire a été la plus fructueuse. En pratique les similarités permettent de déterminer les propriétés des molécules nouvellement séquencées à l'aide de celles des séquences déjà annotées. En effet, une similarité en séquence entraîne généralement une similarité en structure ou en fonction.

La plupart des nombreux logiciels dédiés à la détection de similarités locales, mettent en œuvre des filtres heuristiques : deux portions de séquences ne possédant pas certains motifs spécifiques en commun sont considérées d'emblée comme dissimilaires. Le choix des motifs conditionne la sensibilité et la sélectivité du filtre associé. Dans ce mémoire nous considérons un certain type de motifs appelé graine. Il s'agit en fait de sous-chaînes à trous.

Nous étudions plusieurs problèmes algorithmiques liés à la conception de bonnes graines. En particulier, nous montrons que le problème suivant est NP-difficile : étant donnés deux entiers naturels k, m et une graine, décider si le filtre associé est sans perte lorsque l'on restreint la notion de similarité aux paires de mots de même longueur m, séparés par une distance de Hamming au plus k. Notons que plusieurs algorithmes exponentiels ont été proposés pour des généralisations de ce problème.