MASTER EN SCIENCES MATHÉMATIQUES

Sur un algorithme de recherche de cliques dans le cadre de l'alignement des protéines

LASCHET, Ilona

Award date:
1994

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix
Namur
Faculté des Sciences

Sur un algorithme de recherche de cliques dans le cadre de l'alignement des protéines.

Mémoire présenté pour l'obtention du grade
de Licencié en Sciences
Mathématiques
par

LASCHET Ilona

Promoteur : V.H. NGUYEN
Co-promoteur : E. DEPIEREUX

Année académique 1993-1994

Je remercie toutes les personnes qui m'ont soutenu tout au long de ce mémoire et en particulier Messieurs Van Hien Nguyen et Eric Depiereux, qui ont supervisé l'élaboration de celui-ci.

Merci également à Monsieur Jean Fichefet, qui a accepté d'en être lecteur.

Mes remerciements vont également au professeur Raymond Monnat de l'Université de Washington, Seattle, Washington, d'avoir suscité notre intérêt pour des problèmes mathématiques posés en Biologie moléculaire.

Résumé :

Dans ce travail nous nous intéressons à l'élaboration et à des tests du programme *CLIQUE*. Ce programme s'intègre dans le cadre d'une recherche en Biologie moléculaire de l'Unité de Biologie quantitative des FUNDP de Namur. Il a été écrit pour résoudre un problème rencontré dans une méthode d'alignement de séquences de protéines (*Match-Box*) qui a pour objectif de prédire des régions structurellement conservées au sein des protéines. Ce problème correspond à un problème d'optimisation combinatoire bien connu en théorie des graphes : *La recherche de toutes les cliques maximales d'un graphe donné.*

Abstract :

In this work we are interested in the elaboration and in the testing of the computer program *CLIQUE*. This program is integrated in a research project on molecular Biology of the Quantitative Biology Unit of the Namur FUNDP. It has been conceived for solving a problem encountered in a protein sequence alignment methodology (*Match-Box*) which is aimed to predict structurally conserved regions of protein sequences. This problem corresponds to a very known combinatorial optimization problem in the Graph Theory : *The search for all of the maximal cliques of a given graph.*

Table des matières

Introduction

L'idée de départ de mon mémoire était de travailler sur divers problèmes mathématiques en Biologie, plus particulièrement des problèmes surgissant lors de la prédiction de la structure tridimensionnelle de l'ADN et des protéines.

Au cours de l'année, Monsieur Eric Depiereux, membre de l'Unité de Biologie quantitative des Facultés Universitaires Notre-Dame de la Paix, m'a proposé de tester les performances d'un programme écrit par Florence Badoux ([2], 1992-93) en vue de l'obtention du diplôme de Licenciée et Maître en Informatique.

Ce programme s'intègre dans le cadre d'une recherche en Biologie moléculaire. E. Depiereux et E. Feytmans ont développé une méthode d'alignement de séquences de protéines (chapitre 2) qui a pour objectif de prédire les régions structurellement conservées au sein des protéines. Elle met en oeuvre un ensemble de procédures informatiques qui permettent d'aligner plusieurs séquences de protéines.

A la section 2.4, on s'aperçoit que cette méthode a ses limites. Une idée d'amélioration de la méthode est de prendre plus qu'un "match" similaire par séquence pour chaque fenêtre initiale, ce qui permettra de détecter plusieurs "matches" complets par séquence initiale lors du "screening". Ceci augmente considérablement la complexité de l'algorithme.

Un "match" complet est un ensemble de segments de longueur égale dans lequel

- chaque segment appartient à une séquence différente,
- le nombre de segments formant le "match" complet est égal au nombre de séquences,
- les segments forment un lien complet, c'est-à-dire qu'ils sont similaires deux à deux.

Historiquement, il a été montré qu'il existait une corrélation entre ce problème et un problème de théorie des graphes bien connu : *La recherche des cliques maximales.*

Florence Badoux a écrit son programme, appelé *clique*, en se basant sur des résultats importants de la littérature de la théorie des graphes et en y ajoutant quelques caractéristiques propres au problème biologique étudié.

Dans mon mémoire, j'explique dans un premier temps le problème biologique (chapitre 1, 2, 3), le problème de cliques (chapitre 4) et ensuite la réalisation du programme *clique* en me basant sur le mémoire de F. Badoux [2] (chapitre 5). Le dernier chapitre (6) comportera l'analyse et l'interprétation de divers tests.

Chapitre 1

Biologie moléculaire

Ce chapitre est destiné à tous les non-biologistes qui liront ce mémoire, qui aborde des problèmes de biologie moléculaire. Le centre d'intérêt des biologistes moléculaires est étroitement lié à la compréhension des relations structure-fonction des macromolécules cellulaires et notamment de l'ADN (**A**cide **D**ésoxyribo**N**ucléique) et des protéines. Nous expliquerons dans un premier temps ce qu'est la biologie moléculaire et ensuite ce que sont l'ADN et les protéines.

1.1 Biologie moléculaire

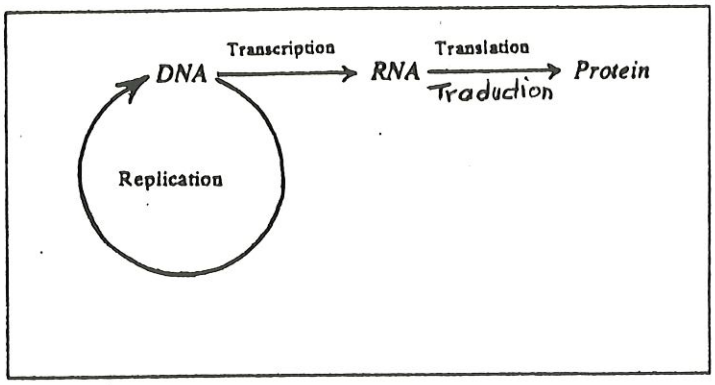
La biologie moléculaire est une branche de la biologie qui essaie de comprendre le fonctionnement cellulaire à l'échelle moléculaire en utilisant la technique de génétique, de biochimie, de physique, etc. La découverte de différentes techniques comme par exemple la cristallographie par rayons X, le microscope électronique ou la résonance magnétique nucléaire (NMR) ont permis de faire d'immenses découvertes. Malheureusement, ces méthodes sont souvent très coûteuses et ne permettent pas de tout résoudre. Il reste donc beaucoup de questions ouvertes dans cette branche de la biologie.

En 1953, Watson et Crick, deux biologistes, américain et anglais, ont élucidé la structure bicaténaire hélicoïdale de l'ADN. Ils décrivirent une image spirale d'une hélice double ordinaire — deux chaînes polynucléotidiques entrelacées — avec des grandes arêtes de sucre et de phosphate à l'extérieur et des paires de bases azotées au centre. Cette découverte provoqua une grande excitation dans le monde scientifique et engendra de nombreuses découvertes concernant les mécanismes d'hérédité. La biologie moléculaire était née.

Déjà en 1944, Avery et ses collègues avaient mis en évidence le rôle central de l'ADN comme agent de l'hérédité, mais le mécanisme de l'inhérence restait un mystère. Il restait surtout deux questions ouvertes :

1. Comment l'information génétique est-elle répliquée ? En d'autres termes, comment

Figure 1.1: Le dogme central de la biologie moléculaire : l'ADN se reproduit, l'ADN est transcrit en l'ARN et l'ARN est traduit en protéines.



la séquence de nucléotides des molécules d'ADN parentales passe-t-elle aux cellules filles ?

2. Comment le contenu chimique de l'ADN dirige-t-il les réactions biochimiques qui font de l'organisme ce qu'il est vraiment? En d'autres mots, comment l'information génétique contenue dans l'ADN dirige-t-elle la synthèse des protéines?

Le modèle de l'hélice double de l'ADN nous mène vers le dogme central de la biologie moléculaire (Figure 1.1). L'ADN dirige sa propre réplication et le flux d'informations biologiques se propage de l'ADN, réservoir de l'information génétique, à l'ARN et de là, à la protéine.

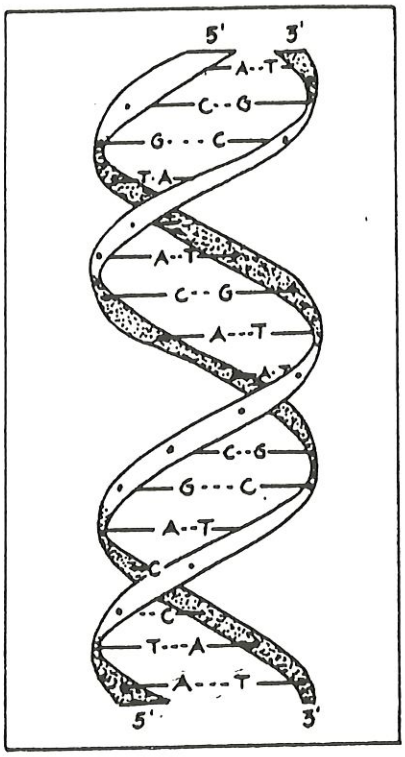
1.2 ADN (Acide DésoxyriboNucléique)

L'ADN contient toute l'information nécessaire à l'élaboration des êtres vivants. Chez les Eucaryotes, il se trouve dans le noyau de chaque cellule sous forme de chromosomes. Par exemple, une cellule humaine contient 46 chromosomes, répartis en 22 paires homologues ($2 \times 22 = 44$ chromosomes) et les chromosomes sexuels X et Y. Pour les autres Eucaryotes le principe reste le même, mais le nombre de chromosomes varie.

Pour un chromosome typique, la longueur d'ADN est de $3 \times 10^{-2} m$ et le diamètre est de $2 \times 10^{-9} m$. A titre comparatif, le diamètre d'un atome est de $10^{-10} m$, ou encore, proportionnellement, une ficelle de 2 mm d'épaisseur devrait mesurer 30 km de long !

L'ADN de la cellule eucaryote est entouré d'une membrane semi-perméable qui ne laisse passer que certaines molécules. Le tout, ADN et membrane, forme une boule d'un diamètre moyen de $10^{-7} m$ appelée noyau.

Figure 1.2: Un regard schématique sur la double hélice de l'ADN.



Le noyau est très petit par rapport à la longueur d'ADN qui est d'environ 1 m dans une cellule humaine. On voit donc que l'ADN n'est pas sous forme d'un filament étiré, mais qu'il est replié sur lui-même.

L'ADN est **une hélice à double brin** (Figure 1.2). Chaque brin est enroulé en hélice autour d'un axe central commun, et est constitué **d'unités de sucre (désoxyribose) et de phosphate** qui alternent. Les échelons de l'hélice double sont constitués de **paires de bases azotées** liées par des ponts hydrogènes. Il existe quatre bases azotées différentes dans l'ADN : les bases pyrimidiques cytosine (C) et thymine (T) et les bases puriques guanine (G) et adénine (A). Une molécule de sucre, de phosphate et de base forment *un nucléotide* et souvent l'ADN est appelé *polynucléotide* car il est constitué d'une chaîne de nucléotides.

L'ARN (**A**cide **R**ibo**N**ucléique) est aussi un polynucléotide, formé d'un seul brin où le désoxyribose est remplacé par le ribose et la thymine (T) par l'uracil (U) . Comme nous le verrons plus tard, l'ARN est formé sur le modèle de l'ADN.

La figure 1.3 montre la composition chimique des différentes molécules.

Comme nous l'avons déjà vu (Figure 1.2), l'ADN est formé de deux brins dont les bases sont liées par des ponts hydrogènes, mais les quatre bases ne peuvent se lier indifféremment entre-elles. En réalité il n'existe — sauf dans des cas très rares — que les deux possibilités de liaison suivantes :

Figure 1.3: La structure chimique des composantes d'un acide nucléique.

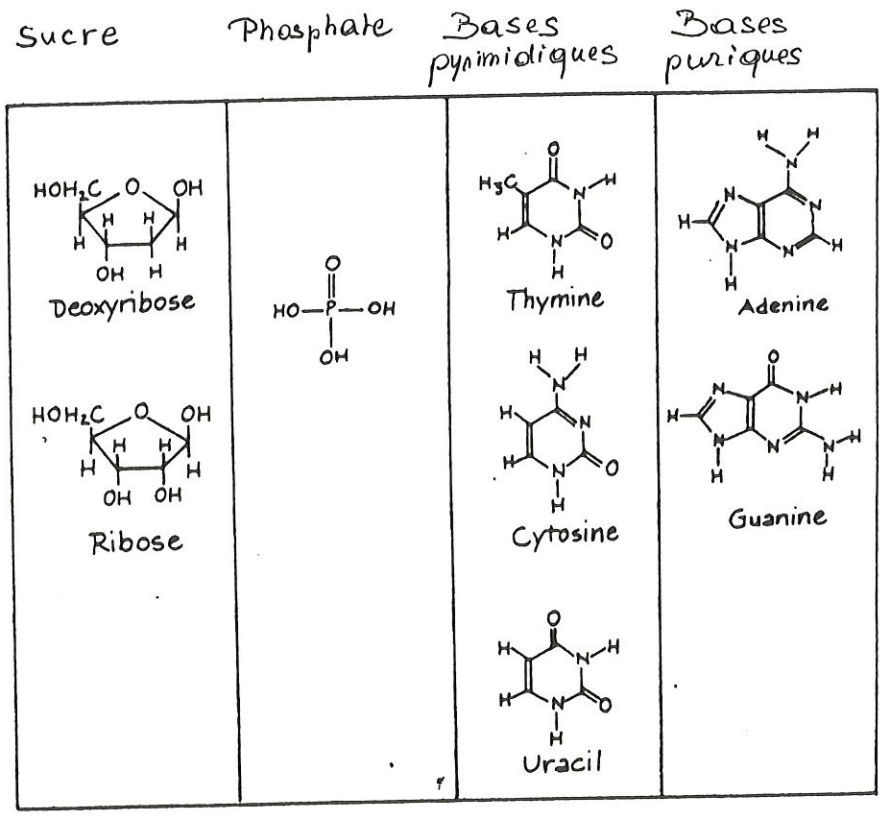
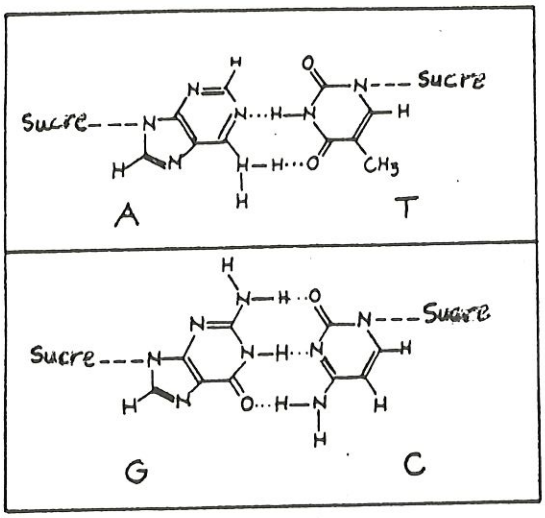


Figure 1.4: Les paires de bases se trouvant dans l'hélice double de l'ADN.



1. Thymine et adénine peuvent s'apparier par deux ponts hydrogènes.

2. Cytosine et guanine peuvent s'apparier par trois ponts hydrogènes.

Ces arrangements produisent les paires de bases C–G et T–A qui ont presque la même dimension (Figure 1.4). Les deux brins d'une hélice double d'ADN sont complémentaires et antiparallèles.

La découverte de la structure tridimensionnelle des hélices de l'ADN explique comment l'ADN (polynucléotide) peut se répliquer et encoder des informations génétiques.

Expliquons maintenant le processus de réplication de l'ADN et la synthèse des protéines.

1.2.1 Le processus de réplication de l'ADN

Puisqu'un brin d'ADN bicaténaire est complémentaire de l'autre, la séquence nucléotidique d'un brin détermine automatiquement la séquence de l'autre brin.

Chaque brin sert de moule à la synthèse du brin complémentaire. La réplication génère deux descendants : deux chaînes d'ADN dotées chacune d'un brin nouvellement synthétisé. (Figure 1.5).

1.2.2 Synthèse de protéines

Un rôle très important de l'ADN est de diriger la synthèse des protéines[1]. La séquence nucléotidique de l'ADN détermine la séquence d'acides aminés (AA) de la chaîne polypep-

[1]vous trouverez des informations supplémentaires sur les protéines en 1.3

Figure 1.5: La réplication de l'ADN.

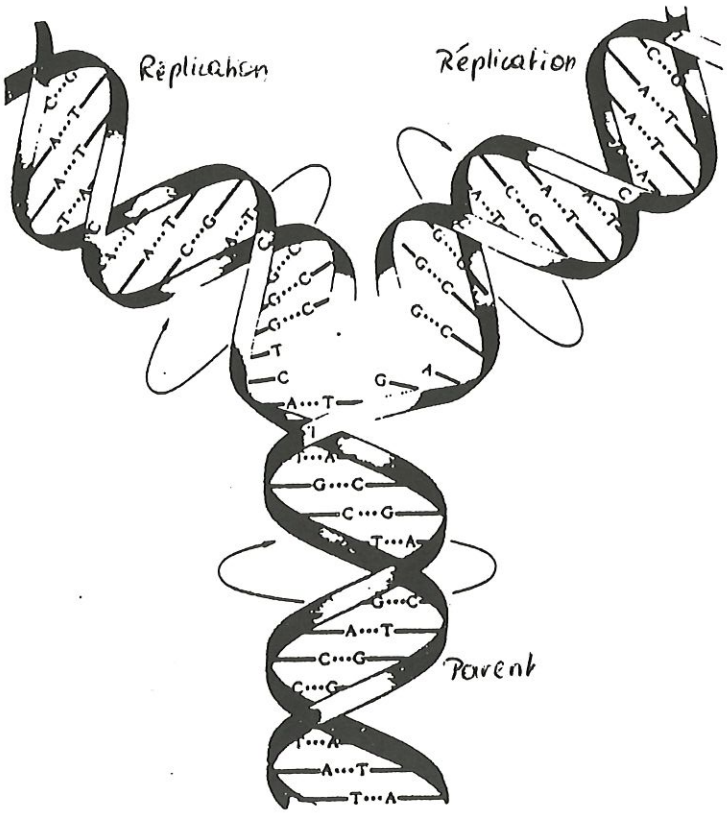


Table 1.1: Code génétique pour l'ADN.

1^e base	2^e base				3^e base
	T	**C**	**A**	**G**	
T	Phe	Ser	Tyr	Cys	T
	Phe	Ser	Tyr	Cys	C
	Leu	Ser	STOP	STOP	A
	Leu	Ser	STOP	Trp	G
C	Leu	Pro	His	Arg	T
	Leu	Pro	His	Arg	C
	Leu	Pro	Gln	Arg	A
	Leu	Pro	Gln	Arg	G
A	Ile	Thr	Asn	Ser	T
	Ile	Thr	Asn	Ser	C
	Ile	Thr	Lys	Arg	A
	Met	Thr	Lys	Arg	G
G	Val	Ala	Asp	Gly	T
	Val	Ala	Asp	Gly	C
	Val	Ala	Glu	Gly	A
	Val	Ala	Glu	Gly	G

tidique (protéine) selon un code génétique qui est presque universel dans le monde vivant (Table 1.1). Vous trouvez les noms et structures des différents AA sur la figure 1.6.

Une séquence de trois nucléotides est un codon. Chaque codon correspond à un seul acide aminé. Comme il y a 4 nucléotides différents il y a $4^3 = 64$ codons différents. 61 des 64 codons représentent un des 20 AA, 3 codons spéciaux (TAA, TAG et TGA) sont des signaux de terminaison ("STOP") de la synthèse des protéines.

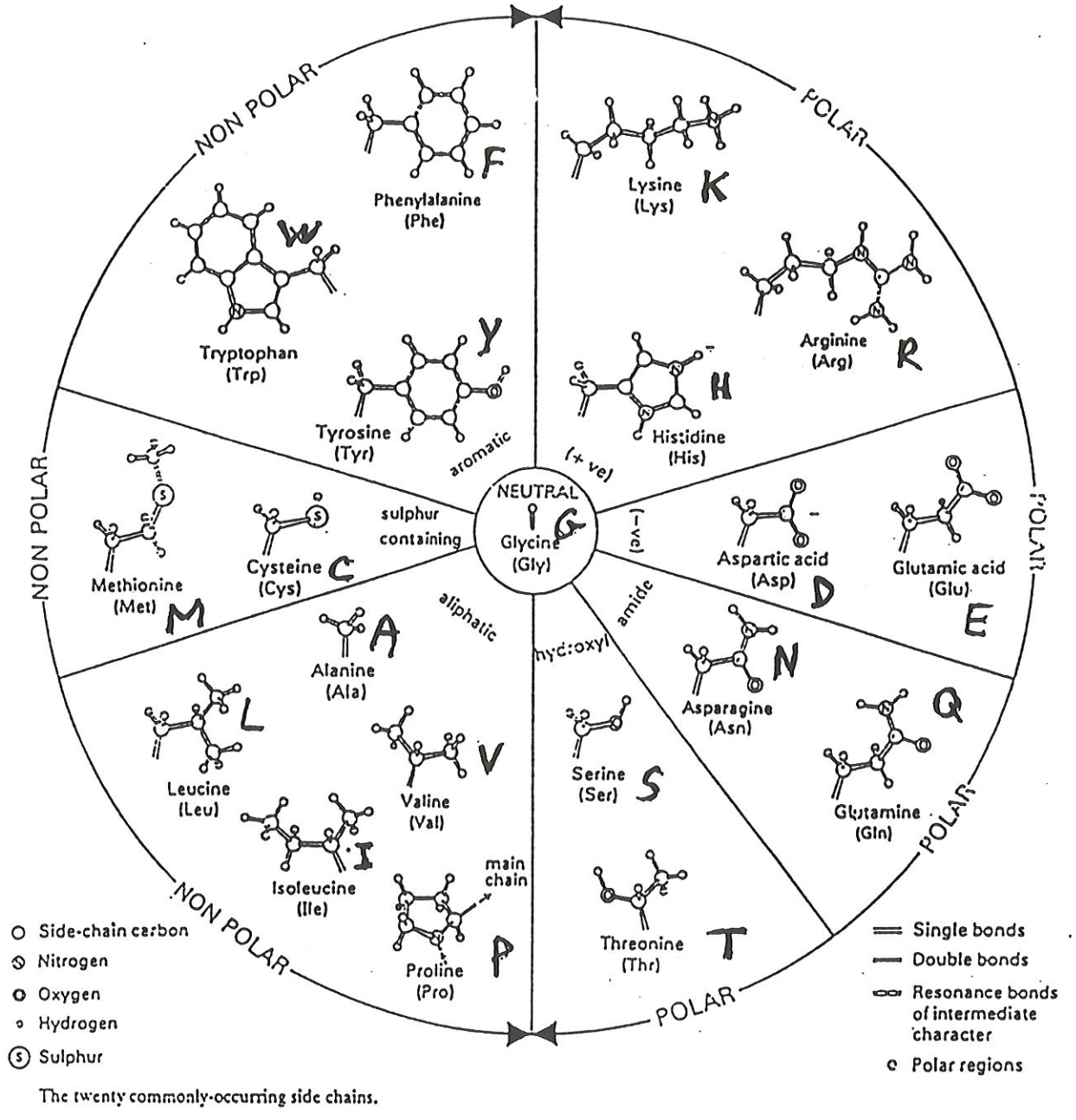
Remarquons que plusieurs codons peuvent correspondre au même AA. Par exemple, AAG et AAA spécifient tous les deux la lysine ('lys') : on dit que le code est dégénéré.

Pour transférer l'information de son ADN à sa protéine, la cellule commence par synthétiser des molécules d'ARN (ARN messager), un processus appelé transcription. La deuxième étape est la traduction de l'ARN messager et la polymérisation d'AA en protéine.

La traduction du code génétique en une séquence d'AA est effectuée par une classe de molécules d'ARN appelée ARN de transfert. Elles changent les AA sous forme active en vue du transfert à la chaîne polypeptidique naissante et interagissent avec l'ARN messager par l'appariement de bases complémentaires.

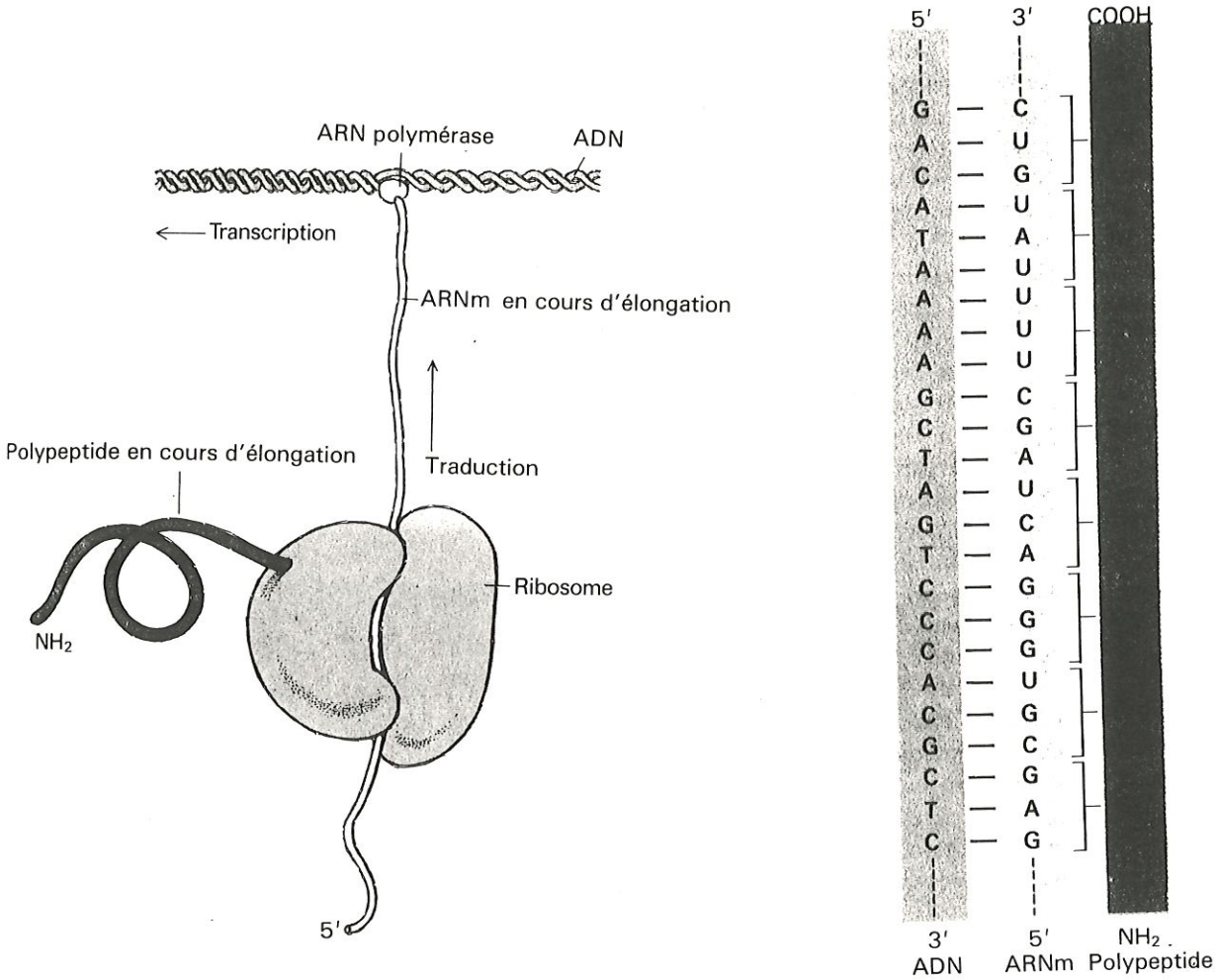
La molécule d'ARN messager est lue au niveau d'un organisme cellulaire appelé ribosome ("machine fabriquant les protéines").

Figure 1.6: Les noms et structures des AA



The twenty commonly-occurring side chains.

Figure 1.7: La synthèse de protéines au niveau des ribosomes.



Au fur et à mesure de la lecture, les AA correspondant aux codons lus sur l'ARN messager sont assemblés par des liaisons peptidiques et forment ainsi la protéine attendue. Au cours de sa synthèse, la protéine va prendre une structure tridimensionnelle complexe. (Figure 1.7)

Cette conformation tridimensionnelle détermine les fonctions biologiques de la protéine dans la cellule. Actuellement, il existe beaucoup de recherches pour trouver les liens entre structure et fonction (voir aussi 1.3 et chapitre 2).

La synthèse des protéines n'est pas la seule fonction de l'ADN. En fait, seulement 1 % de l'ADN du génome humain est transcrit en protéines. On ne sait pas à l'heure actuelle à quoi sert la grande majorité de l'ADN.

Comme les biologistes supposent qu'il existe une forte corrélation entre structure tridi-

mensionnelle de l'ADN et fonctions de l'ADN, ils pensent qu'une connaissance exacte de la structure de l'ADN dans l'espace pourrait permettre de comprendre ces fonctions.

Malheureusement, il est très difficile d'obtenir des données de résolution atomique et très souvent des méthodes théoriques représentent une alternative pour ces informations structurelles. En particulier, la méthode de l'énergie potentielle est reconnue comme une méthode effective pour prédire les structures ([34], [9], [28]). Le principe de cette méthode est de chercher la structure qui minimise la fonction de l'énergie potentielle, car on suppose que la configuration tend à minimiser son énergie. Selon ce critère la structure trouvée est donc celle avec la probabilité d'apparition la plus grande.

1.3 Les protéines

Les protéines sont à la base de la vie, elles ont un rôle crucial dans pratiquement tous les processus vitaux. Elles ont une très grande importance et une activité étendue dans les processus biologiques suivants :

- La catalyse enzymatique.
  Pratiquement toutes les réactions chimiques sont catalysées par des protéines spécifiques, les enzymes. Ces enzymes augmentent la vitesse des réactions dans les systèmes biologiques.

- Le transport et le stockage.
  Des protéines spécifiques transportent les petites molécules et les ions, par exemple, l'hémoglobine transporte l'oxygène dans les globules rouges.

- Les mouvements coordonnés.
  Les protéines sont les composantes majeures des muscles.

Les protéines sont construites sur base de vingt acides aminés (AA) différents (Figure 1.6). Les AA se distinguent entre eux par des propriétés chimiques et physiques, comme par exemple l'hydrophobicité[2], la charge, etc. Deux acides aminés peuvent être liés par une liaison covalente (liaison peptidique) et forment alors un dipeptide. Quand trois résidus sont enchaînés, nous parlons de tripeptide. Lorsqu'il s'agit d'un enchaînement plus long, nous parlons de polypeptide, comme pour les protéines, dont la taille moyenne compte entre 200 et 800 résidus. Il est clair que les propriétés des protéines sont radicalement plus complexes que celles de leurs composants isolés, vu le nombre important de ses composants.

Actuellement, on ne comprend pas encore toutes les propriétés et tous les modes de fonctionnement des protéines. Les biologistes ont montré que des protéines de structures

[2]Un AA hydrophobe est un AA qui évite l'eau, par opposition à l'hydrophile qui a de l'affinité pour l'eau.

très proches ont des fonctions semblables. On sait aussi que des séquences très semblables génèrent des structures très semblables. Par contre, on ne sait pas du tout ce qui détermine le repliement de la protéine, donc on est encore incapable de prédire la structure à partir de la seule séquence. Sur les photos des figures 1.8, 1.9, 1.10 et 1.11 vous trouvez les séquences et structures de deux protéines (des protéases). On voit que des parties similaires de séquences correspondent à des parties ayant également une structure similaire. Ces protéines ont des séquences et des structures très similaires car elles sont toutes les deux des protéases, donc elles ont les mêmes fonctions.

Les biologistes cherchent ainsi à connaître plus de détails sur la structure tridimensionnelle des protéines, de manière à élucider la relation structure–fonction. Mais cette recherche est confrontée à des problèmes importants.

Vu le nombre élevé d'acides aminés qui forment une protéine, on peut facilement imaginer que la structure tridimensionnelle des protéines est très compliquée. La cristallographie à rayons X ou les techniques de résonance magnétique (NMR) permettent de déterminer les structures tridimensionnelles. Malheureusement ces méthodes sont très laborieuses, et à l'heure actuelle, seules les structures de 600 protéines ont été définies avec précision parmi lesquelles 150 environ sont réellement différentes. Par contre, il est relativement facile de déterminer la structure primaire des protéines, c'est-à-dire la séquence en AA qui les compose. On connaît déjà 100 000 séquences et ce nombre augmente chaque jour.

Le défi est donc de déduire la structure tertiaire[3] d'une protéine à partir de la seule connaissance de sa séquence. Ce faisant, on se base sur le principe que toute l'information se trouve au niveau de la séquence. En effet :

1. Une protéine qui a été dénaturée se repliera en retrouvant sa structure indépendamment des ribosomes. Ceci contredit l'hypothèse selon laquelle la structure tridimensionnelle des protéines serait liée à des informations données par le ribosome.

2. Une séquence particulière d'au moins sept AA va toujours être amenée à prendre la même configuration, même si elle se trouve dans des protéines différentes (cfr. les photos des figures 1.8, 1.9, 1.10 et 1.11).

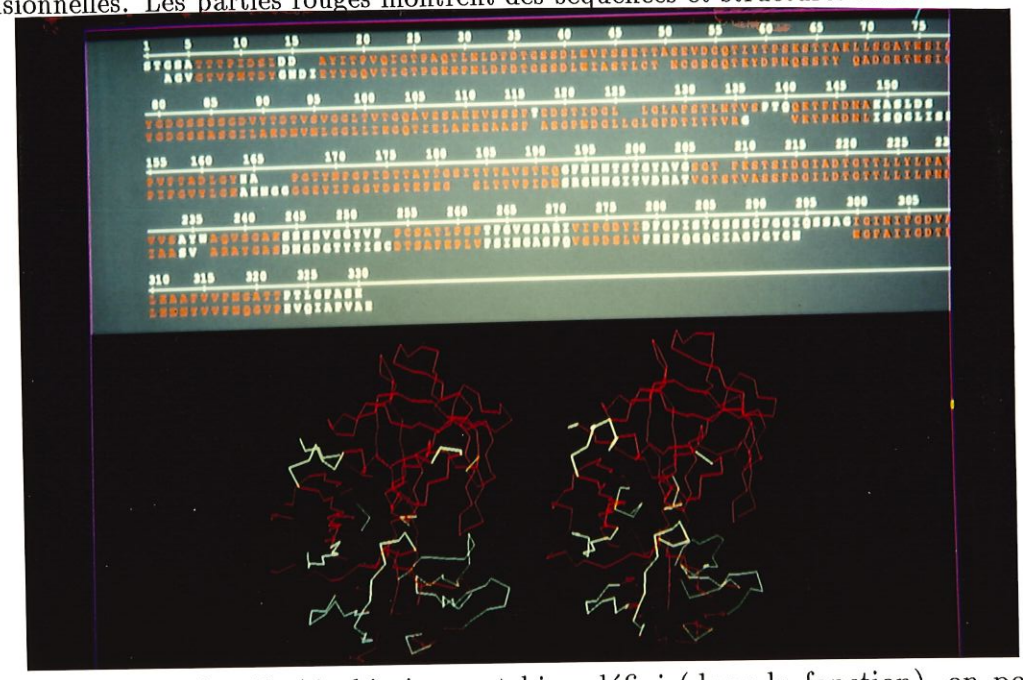
On veut donc déterminer la structure spatiale d'une protéine en comparant sa séquence avec d'autres séquences dont on connaît la structure et en supposant que les régions de séquences similaires ont aussi des structures similaires, donc des fonctions similaires, car on a supposé une liaison entre fonction et structure.

Il peut également être intéressant de pouvoir comparer des séquences dont aucune structure tridimensionnelle n'est connue. Si une séquence présente une similarité importante

[3]tridimensionnelle

Figure 1.8: Deux protéines.

Ce photo montre la séquence en AA de deux protéines (des protéases) ainsi que leurs structures tridimensionnelles. Les parties rouges montrent des séquences et structures similaires.



avec une autre dont le rôle biochimique est bien défini (donc la fonction), on peut supposer que la fonction de ces protéines est similaire. De plus, la comparaison de protéines éloignées peut mettre en évidence quelques résidus conservés, essentiels à la fonction des protéines.

Il existe beaucoup de nouvelles méthodes pour prédire la conformation. Celles-ci sont essentiellement basées sur les deux critères suivants :

1. **Critère cinétique et thermodynamique.**
   L'approche cinétique et thermodynamique est essentiellement liée à des contraintes énergétiques. Pour attribuer une conformation à une séquence, il suffirait en principe de choisir la conformation dont l'énergie globale est la plus faible parmi une série de conformations potentielles. Ce critère est aussi employé pour des chaînes polynucléotidiques.

2. **Critère de prédiction de la conformation par comparaison de séquences.**
   On se base sur le principe que l'information nécessaire au repliement des protéines se trouve dans la séquence en AA. Selon ce principe, des séquences très semblables peuvent avoir une structure tridimensionnelle très proche. Il faut aligner des séquences pour les comparer et détecter d'éventuelles similarités. E. Depiereux et E. Feytmans ont mis au point une méthode originale d'alignement multiple que l'on expliquera au chapitre 2.

Figure 1.9: Deux protéines.

Ce photo montre la supposition des structures des deux protéines. Ici on voit encore une fois que les parties rouges sont similaires.

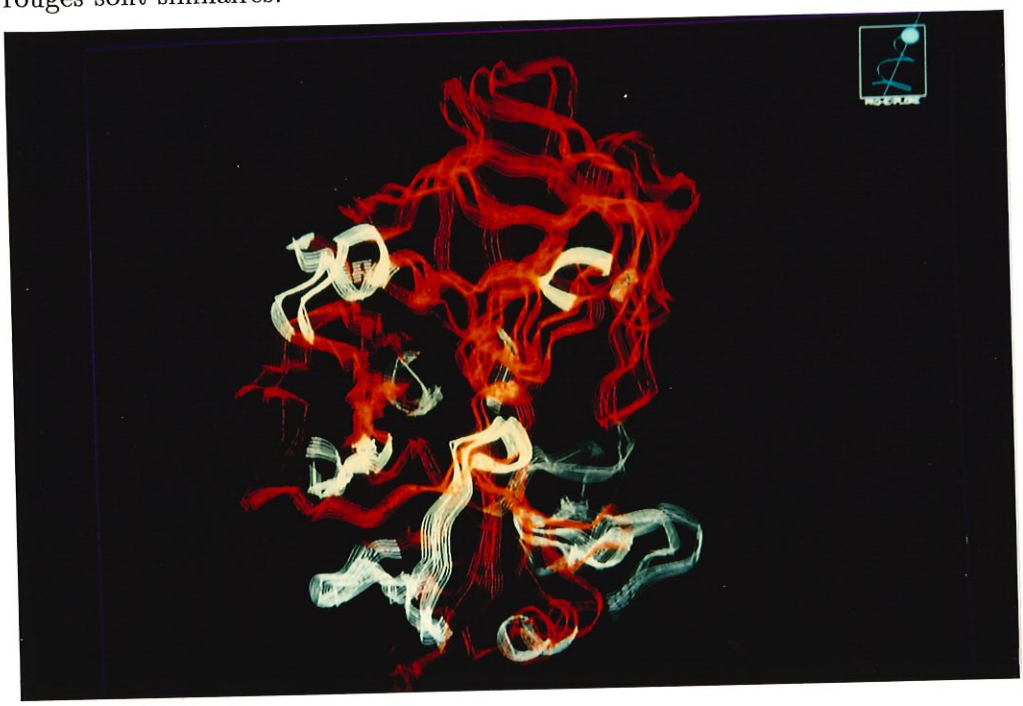


Figure 1.10: Deux protéines.

Une partie de l'alignement qui montre bien la similarité entre deux segments des protéines.

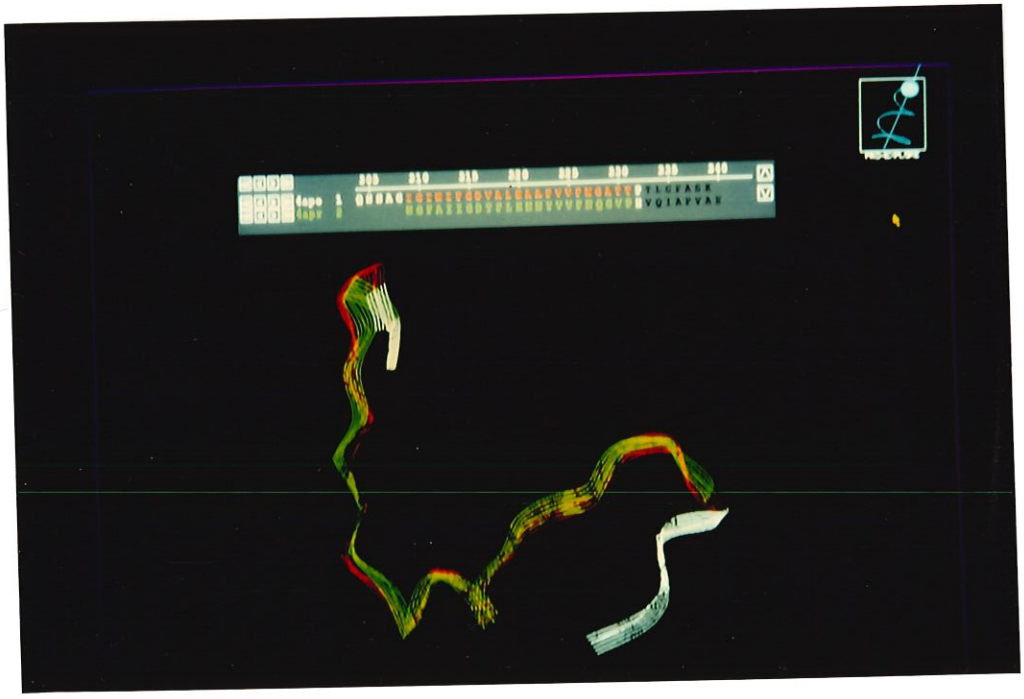
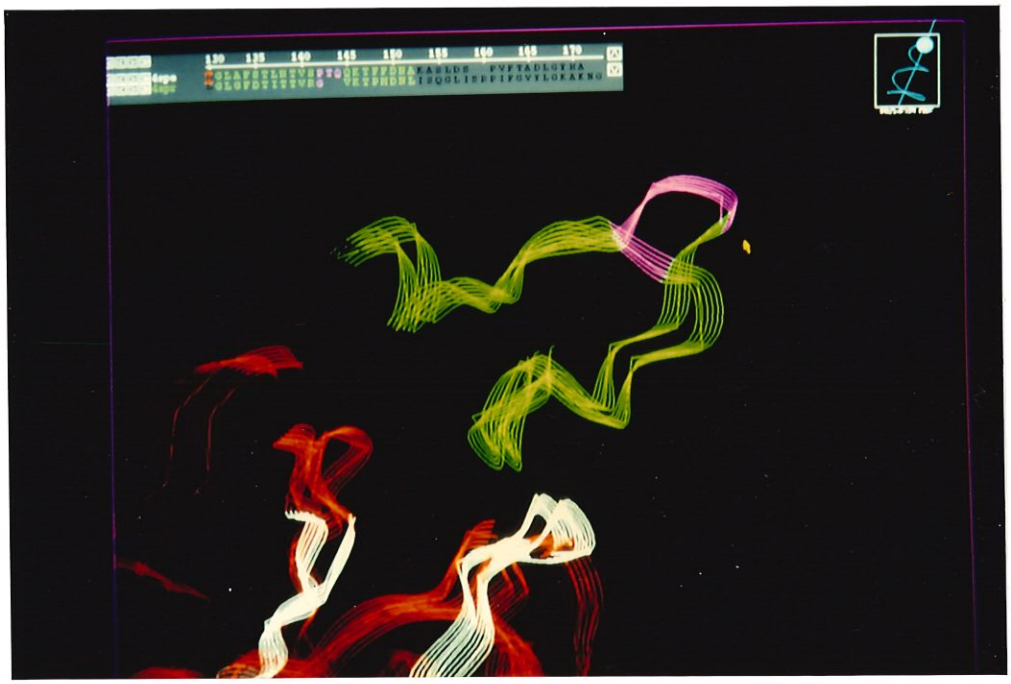


Figure 1.11: Deux protéines.

Ce photo montre à nouveau qu'un segment de la séquence en AA des protéines. Les parties vertes sont similaires. En mauve, on voit des parties non similaires des deux segments. Pour bien représenter les boites (parties vertes), on a du introduire des espaces ("gaps") dans la séquence en AA de la deuxième protéine.



Chapitre 2

Alignement multiple de séquences de protéines

2.1 Introduction

Un problème de la biologie moléculaire[1] est la connaissance des structures et des fonctions des protéines ainsi que le rapport structure-fonction. Malheureusement, on ne connaît la structure tridimensionnelle et les fonctions exactes que de très peu de protéines. Par contre, la séquence en AA[2] de beaucoup de protéines a été identifiée. On estime qu'il existe un lien étroit entre structure et fonction des protéines et que toute l'information sur cette structure et fonction se trouve inclue dans la séquence des AA. L'idée est donc de comparer différentes séquences de protéines pour trouver des régions similaires pouvant avoir des structures et des fonctions semblables.

Eric Depiereux et Ernest Feytmans ont mis au point une méthode originale d'alignement, *la méthode Match-Box*, permettant d'aligner plusieurs séquences de protéines simultanément. L'avantage de cette méthode par rapport aux méthodes existantes est de pouvoir aligner plus de deux séquences en même temps.

La méthode permet un rassemblement des séquences en groupes similaires, qui sera expliqué à la page 30 et la détection de régions similaires entre les différentes séquences de protéines alignées, de manière à prédire les régions structurellement conservées (SCR) et les fonctions semblables des différentes protéines.

L'alignement[3] est réalisé en trois parties.

1. Balayage des séquences ("SCANNING"), de façon à réaliser toutes les comparaisons

[1] défini au chapitre 1

[2] AA : Acide aminé

[3] plus précisément la détection de régions similaires

souhaitées.

2. Comparaison des segments ("MATCHING").

3. Tri des fenêtres sélectionnées, de manière à ne conserver que les segments significativement similaires ("SCREENING").

2.2 Algorithme

Soient r séquences de protéines de longueur L_i $(i = 1, ..., r)$ alignées simultanément dont on veut chercher des régions similaires. Avant de décrire les différentes étapes de l'algorithme, nous allons définir quelques termes utilisés.

2.2.1 Définitions

Une fenêtre de longueur w est un segment de w résidus[4] d'une séquence d'AA.

Un "match" (groupe) correspond à deux fenêtres similaires appartenant à deux séquences différentes.

Un "match" complet (groupe complet) est un ensemble de segments de longueur égale dans lequel :

1. Chaque segment appartient à une séquence différente.
2. Le nombre de segments formant le "match" complet est égal au nombre de séquences.
3. Tous les segments sélectionnés sont liés par un lien complet, c'est-à-dire ils sont similaires deux à deux.

Des régions structurellement conservées (SCR) sont différents segments dont la structure tridimensionelle est la même (cfr. les photos des figures 1.8, 1.9, 1.10 et 1.11).

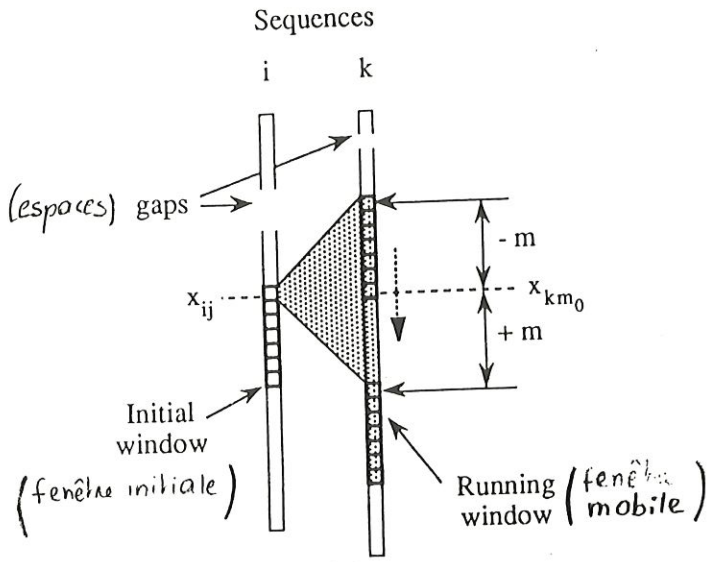
Des régions structurellement conservées prédites (pSCR) sont des SCR probables en fonction de méthodes théoriques, appliquées à des séquences d'AA.

Un "gap" (espace) dans une séquence est un espace qu'on introduit dans cette séquence pour aligner des protéines (cfr. la photographie de la figure 1.11).

Voici maintenant l'explication des différentes étapes de l'algorithme.

[4] AA

Figure 2.1: Le balayage des séquences



2.2.2 Balayage des séquences ("SCANNING")

On fixe une fois, pour tout le processus, la longueur des fenêtres w et le paramètre m. La taille des fenêtres (w) doit être suffisamment grande pour éventuellement représenter une structure conservée (SCR), mais pas trop longue de manière à ne pas affecter la précision de l'alignement (il est plus facile de trouver de petits segments similaires que des grands). En général, on prend w=7 (en relation avec le point 2 de la page 17).

On fixe une fenêtre initiale qui se déplace progressivement — d'un résidu à la fois — des w premiers résidus de la première séquence aux w derniers résidus de la $r^{ième}$ séquence. Pour chaque fenêtre initiale, une fenêtre mobile est déplacée de résidu à résidu le long de toutes les autres séquences, avec un déplacement positif ou négatif de m AA de part et d'autre de la position de la fenêtre initiale (Figure 2.1).

Si m = 0, alors la fenêtre mobile est définie seulement tous les 7 résidus situés juste en face dans les autres séquences. Par contre, si $m = \max_i (\frac{L_i}{2} \mid i = 1...r)$, alors la fenêtre mobile parcourt l'entierté des autres séquences. Les valeurs intermédiaires de m définissent une zone limitée pour le déplacement de la fenêtre mobile.

A chaque étape, le segment défini par la fenêtre mobile est comparé au segment défini par la fenêtre initiale selon la procédure de "MATCHING" décrite ci-après.

2.2.3 Comparaison de deux fenêtres ("MATCHING")

Il existe deux types de mesures de similarité des segments de protéines : celle basée sur la structure[5] et celle basée sur la séquence d'AA. Comme nous travaillons au niveau des

[5]structure spatiale ou tridimensionnelle

séquences d'AA nous nous basons sur la mesure de similarité correspondante.

Comme un "match" est formé de deux fenêtres similaires appartenant à deux séquences différentes, nous avons besoin d'une mesure de similarité entre les deux fenêtres. La détermination quantitative du degré de similarité entre les segments n'est pas triviale, sauf dans le cas où ceux-ci sont tous identiques.

Nous utiliserons deux critères :

1. Le nombre d'identités des AA entre deux fenêtres.

2. La similarité physicochimique de deux fenêtres.

Pour les deux critères, il faut que le degré de ressemblance soit plus important que celui qui pourrait être dû au seul hasard. Expliquons maintenant les deux critères.

I. Le nombre d'identités

Pour définir de fenêtres similaires, le nombre d'identités doit être plus élevé que celui obtenu pour des fenêtres générées aléatoirement. Supposons que l'on construise au hasard deux segments à partir des 20 AA. Notons B le nombre d'identités entre les résidus correspondants de deux fenêtres. Comme les fenêtres sont de longueur w il est évident que $0 \leq B \leq$ w. De plus, si l'on suppose que les 20 AA ont la même probabilité d'être choisis pour construire la chaîne polypeptidique (c'est-à-dire une probalilité $p = \frac{1}{20}$), nous pouvons approximer la distribution de probabilité de B par une binomiale de w essais et de probabilité $p = \frac{1}{20}$. Deux fenêtres sont similaires, si B (le nombre d'identités) est plus grand que le nombre maximum attendu aléatoirement, pour une probabilité égale à $1 - \alpha$, où α est le degré d'incertitude fixé dès le début de l'étude (α vaut environ 0.05).

II. La similarité physicochimique

Chaque paire d'AA est caractérisée par une distance physicochimique qui se trouve dans *une matrice de score*[6] de 210 valeurs (construite à partir des propriétés physiques et chimiques des AA). Nous employons ici la matrice de score de E. Depiereux et E. Feytmans (1991), qu'on appelle également matrice physicochimique, car sa construction est basée sur 10 propriétés physicochimiques (voir 3.4) des chaînes latérales des AA. La distance entre deux segments de longueur w, pour toutes les propriétés physicochimiques considérées, est égale à

$$D^2 = \sum_{k=1}^{210} D_k^2 x_k$$

[6]Voir chapitre 3 pour avoir une définition de la matrice de score.

où D_k^2 représente une des 210 valeurs de la matrice physicochimique et x_k représente le nombre d'occurences de la paire d'AA correspondant à D_k^2 dans les deux fenêtres. Notons que x_k vaut 0 si la paire d'AA est absente.

La distribution de probabilité de D^2 est un χ^2 avec pw degrés de liberté, si les p propriétés physicochimiques sont indépendantes. Pour la matrice de score de E. Depiereux et E. Feytmans nous savons que $p = 10$, donc la distribution de probabilité de D^2 est un $\chi^2(10\ w)$.

Deux fenêtres sont donc considérées comme similaires si la distance entre les profils physicochimiques est inférieure à une certaine limite égale à un $\chi^2_{pw,\alpha}$, où α est le degré d'incertitude fixé dès le début de l'étude.

En résumé, deux fenêtres sont déclarées similaires et forment donc un "match", si une des conditions suivantes est satisfaite :

- Le nombre d'identités entre les résidus correspondants, B', est supérieur au nombre maximum attendu pour des séquences aléatoires avec une probabilité $1 - \alpha$:

$$B' \geq B(1-\alpha) \text{ avec } B \sim B_i(\frac{1}{20}, w)$$

- La différence de profils physicochimiques D^2, définie à partir de p variables indépendantes, est inférieure à la valeur minimale attendue pour des séquences aléatoires avec une probabilité α :

$$D^2 \leq \chi^2_{pw,\alpha}$$

Remarque : Suivant ces critères, une proportion α de fenêtres générées au hasard seront déclarées similaires. Idéalement α vaut donc 0, pour ne pas déclarer des fenêtres non similaires comme similaires, mais alors on ne trouve que très peu de régions similaires et par conséquent l'erreur β est très grande. L'erreur β correspond au fait de ne pas déclarer deux fenêtres comme similaires qui le sont réelement.

En raison de l'erreur β, liée à α, α ne doit pas être ni trop grand, ni trop petit. Il est généralement de l'ordre de 5%, mais varie en fonction de la valeur du *cutoff* choisie.

2.2.4 Tri des fenêtres similaires ("SCREENING")

Cette étape consiste à rechercher des "matches" complets (groupes complets) en se basant sur tous les "matches" trouvées par l'étape précédente. Il reste deux problèmes :

1. Durant le "screening" une fenêtre initiale peut être similaire à plusieurs fenêtres mobiles dans une même séquence (Bruit de fond lié à α et β).

2. Un groupe de fenêtres mobiles n'est pas nécessairement complet, car la similarité n'est pas transitive.

Le premier problème est évité par la méthode de la distance-minimale. Pour une fenêtre initiale, on sélectionne dans toutes les autres séquences la fenêtre la plus similaire (s'il en existe). La sélection de la paire la plus similaire se fait de la façon suivante, avec par ordre d'importance :

- Le plus grand nombre d'identités.
- Le plus petit D^2.
- En cas de scores égaux, l'appariement est choisi en fonction de la plus petite différence entre les positions respectives des fenêtres dans les deux séquences.

Le deuxième problème est résolu par la sélection de fenêtres formant un groupe complet. On peut avoir deux liens entre les fenêtres :

Lien simple : A, B et C forment un groupe simple
$\Longleftrightarrow$ A "match" avec B et C,
ou B "match" avec C et A,
ou C "match" avec A et B.

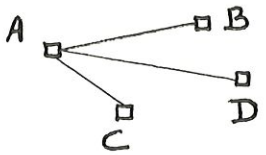
Lien complet : A, B et C forment un groupe complet
$\Longleftrightarrow$ A "match" avec B et C,
et B "match" avec C.

Pour définir un "match" complet (groupe complet), on doit obtenir un lien complet entre toutes les fenêtres appartenant à ce groupe. Pour chercher des fenêtres similaires ayant un lien complet entre elles, nous procédons de la façon suivante :
Soit un groupe de s segments possédant le même nombre de résidus, w, avec $s = \sum_{i=1}^{r} r_i$, r_i représentente le nombre de matches obtenus pour une fenêtre initiale au sein d'une même séquence. Nous construisons *une matrice de vérité* R (de dimension $s \times s$), dont les éléments R_{ij} valent 1 si les segments correspondants sont similaires et 0 sinon. Si la méthode de la distance-minimale est appliquée (2.2.4) $r_i = 1\ \forall i$ et $s = r$. Si la méthode de la distance-minimale n'est pas appliquée $r_i >> 1$ et $s >> r$. Dans ce cas, R_{ij} est assigné à 0 si i et j correspondent à des segments de même séquence. De cette façon, il est impossible de créer un match complet comprenant plus qu'un segment d'une séquence donnée.

Remarquons que les éléments de la diagonale de R valent tous 1. Ensuite, il faut chercher *des cliques de R*, c'est-à-dire des sous-matrices de R entièrement remplies de 1, qui correspondent à un groupe de segments ayant un lien complet entre-eux, donc à un

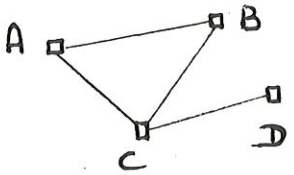
Figure 2.2: La représentation schématique des liens entre différentes fenêtres.
Soient A, B, C et D quatre fenêtres représentées par un □. Les fenêtres formant un groupe sont liées.
(a) A, B, C et D forment un groupe simple (les fenêtres sont liées par un lien simple) car A "match" avec B, C et D.

(a)



(b) A, B et C forment un groupe complet (les fenêtres sont liées par un lien complet) car A "match" avec B et C et B "match" avec C.

(b)



groupe complet (Figure 2.2). Chaque fenêtre initiale est donc susceptible de générer un seul match complet si la méthode de la distance-minimum est appliquée, et un grand nombre si elle n'est pas appliquée.

Enfin, pour réaliser l'alignement final des séquences de protéines, il faut mettre en correspondance les régions définies pour chaque groupe complet. Un problème subsiste lorsque plusieurs groupes complets incluent le même résidu mais ne proposent pas le même alignement (les "gaps" sont différents). Ils s'excluent donc mutuellement. Pour résoudre ce problème, le critère suivant est utilisé :
Des groupes complets sont considérés comme significatifs lorsqu'ils forment le plus grand ensemble de segments similaires compatible avec le même ensemble de "gaps".

Il paraît en effet logique que si l'on parvient à déterminer des zones similaires sur une grande partie des séquences, et que l'on trouve un petit groupe complet qui vienne 'casser' cet alignement, on n'en tienne pas compte.

En résumé, pour sélectionner des groupes complets, nous nous basons sur la méthode de la distance-minimum et du groupe complet. On a les trois étapes suivantes (Figure 2.3):

1. Sélectionner pour chaque fenêtre initiale la fenêtre la plus similaire dans chacune des autres séquences ou plusieurs fenêtres mobiles si la méthode de la distance-minimum n'est pas appliquée.

2. Tester si cette sélection forme un groupe complet. Si la méthode de la distance minimale n'est pas appliquée, on peut trouver plusieurs groupes complets pour une fenêtre initiale et on les cherche par la méthode *des cliques* expliquée au chapitre 4 (Figure 2.3 (a) et (b)).

3. Rechercher des groupes complets significatifs pour réaliser l'alignement final (Figure 2.3 (d)).

2.3 Schéma de l'algorithme

La méthode d'alignement simultané de plusieurs séquences est schématisée à la figure 2.4. Les rectangles correspondent aux différentes étapes, donc à des procédures ou programmes. Les ovales correspondent aux fichiers nécessaires.

Trois fichiers sont nécessaires avant l'alignement proprement dit. Le premier fichier **GCGfiles** contient les séquences en format UWGCG (University of Wisconsin, Genetic Computer Group). Le deuxième fichier **SCORING** contient des matrices de scores sous forme d'un vecteur de 210 valeurs. Remarquons qu'on peut utiliser n'importe quelle matrice de score X pour l'algorithme — même des matrices construites sur base d'autres critères que les propriétés physicochimiques, comme par exemple la matrice de Dayhoff (cfr. 3.3.1) — mais que les résultats ne sont pas nécessairement les mêmes. E. Depiereux et E. Feytmans ont surtout utilisé la matrice physicochimique, dont la construction est décrite en 3.4, pour tester l'efficacité de leur méthode d'alignement. L'utilisateur écrit le nom de la matrice de score choisie dans le fichier d'entrée **SEQUENCE.DAT** qui est le troisième fichier.

La procédure **SEQUENCE** construit le fichier **PROT.DAT** qui contient les séquences d'AA (y compris les espaces) des protéines ainsi que le fichier **PARAM.DAT** qui contient les paramètres w et m nécessaires à l'alignement et les limites en-dessous ou au-dessus desquelles deux segments sont considérés comme étant similaires et la matrice de score choisie. Ce dernier fichier peut être modifié par l'utilisateur.

L'alignement proprement dit se fait en trois étapes décrites en 2.2, qui sont :

1. Balayage des séquences.
2. Comparaison des fenêtres.
3. Tri des fenêtres similaires.

La première et deuxième étape sont réalisées grâce au programme **MATCHING**. Chaque comparaison reçoit une valeur selon la matrice de scores choisie et les groupes trouvés sont stockés dans le fichier **MATCH.DAT** qui est la base de données des groupes ("database of matches"). La troisième étape est accomplie selon le programme **SCREENING**. La sélection des fenêtres similaires se fait par la méthode de la distance-minimale et du groupe complet expliquée précédemment.

Lorsqu'une série de fenêtres a été sélectionnée de cette manière , il reste encore des appariements inconsistants. Ceux-ci sont détectés et éliminés selon différents critères :

Figure 2.3: La procédure du screening

On a les séquences 1, 2, 3 et 4. Des fenêtres sont représentées par des carrés blancs et les "matches" par des lignes. Un groupe de lignes parallèles montre un groupe de "matches" significatifs.

(a) Résultats de la procédure du "scanning" et "matching". (i) Pour une fenêtre initiale, plusieurs "matches" peuvent être choisies dans chaque séquence. (ii) Des groupes complets potentiels sont indiqués, seulement pour le premier match de la deuxième séquence.

(b) Screening par la méthode des cliques. (i) Tous les groupes complets sont trouvés par l'analyse d'une matrice de vérité associée à la fenêtre initiale. (ii) Plusieurs groupes complets peuvent être trouvés pour une fenêtre initiale.

(c) Screening par la méthode de la distance minimum. (i) Dans toutes les séquences le meilleur "match" est sélectionné. (ii) On teste si cette sélection forme un groupe complet. (iii) Le groupe complet est formé s'il y en a un.

(d) L'alignement final. (i) Des groupes significatifs sont représentés par des lignes continues, des bruits de fonds par des lignes discontinues. (ii) Des groupes significatifs sont sélectionnés. (iii) Des espaces ("gaps") sont introduits pour construire l'alignement final. Des pSCR sont représentés par des boites.

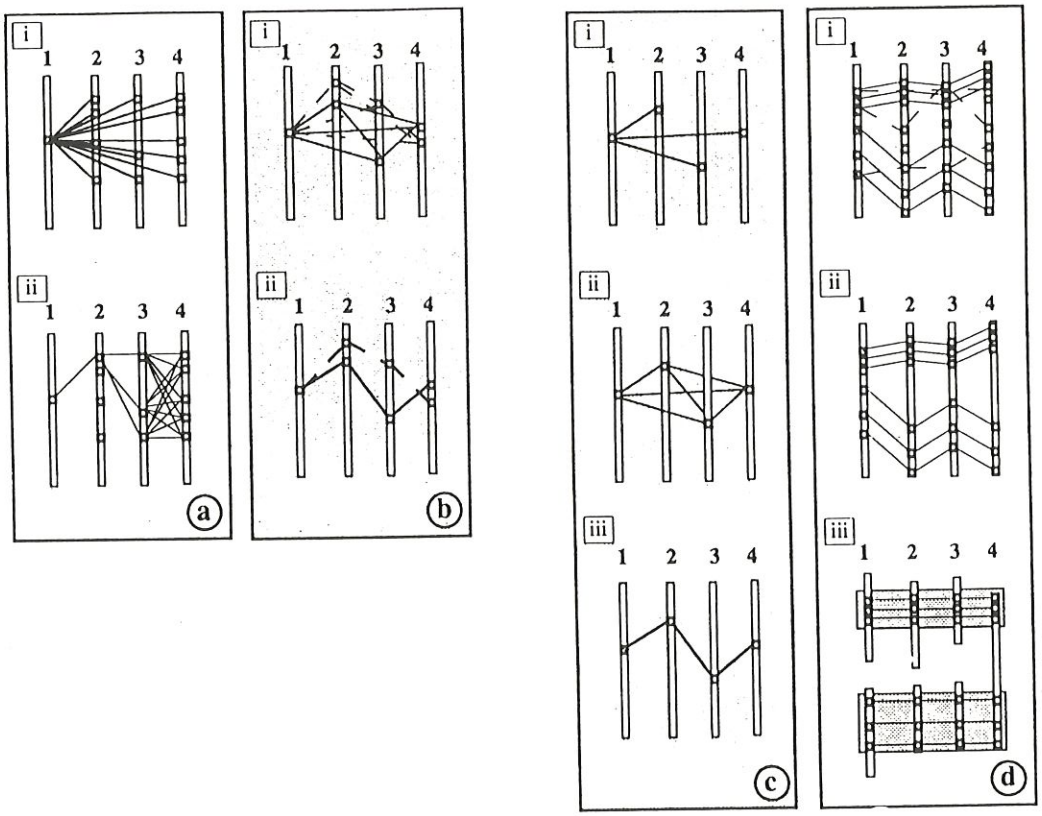
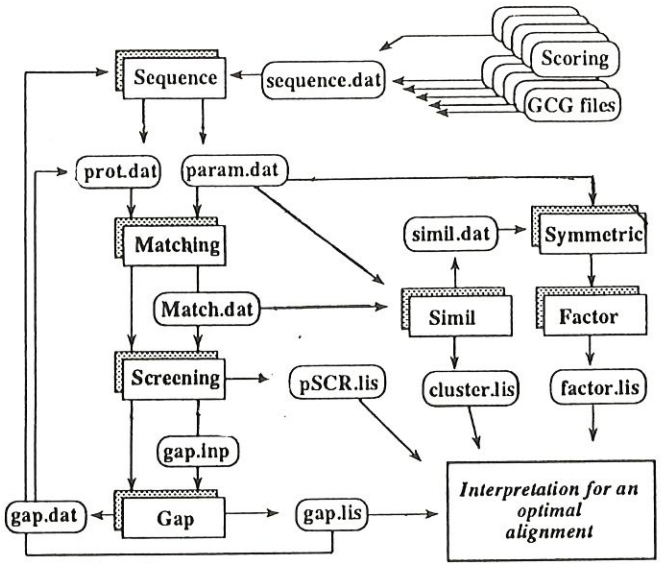


Figure 2.4: Algorithme de l'alignement multiple de E. Depiereux et E. Feytmans.



- La probabilité d'apparier des régions sans relation diminue avec la taille de ces régions. Plus le pSCR est grand, plus il est fiable.
- Le nombre d'espaces ("gaps") nécessaires pour aligner les régions similaires est également pris en compte. Plus ce nombre est faible, plus il est favorable.

Le résultat du SCREENING contient les pSCR qui seront stockés dans le fichier **pSCR.LIS**.

Après cela, la présentation finale sera faite par le programme **GAP**. Ce programme introduit les espaces dans les séquences selon les pSCR déterminés à l'étape précédente. Ces espaces sont placés arbitrairement à la position précédant le premier résidu d'une pSCR (cfr. les photos des figures 1.8, 1.9, 1.10 et 1.11). Lorsque les espaces ont été introduits, les pSCR sont délimités et encadrés et forment des boîtes. Sur les photos les boîtes ne sont pas encadrées, mais les AA correspondants sont notés en des couleurs différentes. En 6.3.2 les boites ne sont pas encadrés non plus, mais les AA appartenant à des boites sont écrits en minuscules alors que tous les autres sont indiqués en majuscules. Les "gaps" sont représentés par des tirets. Le fichier **GAP.INP** contient les modèles (pattern) utilisés pour aligner toutes les fenêtres similaires sélectionnées par la procédure SCREENING. Les séquences alignées sont stockées dans le fichier **GAP.LIS**.

Lorsque les séquences sont alignées pour la première fois, il est nécessaire de choisir un balayage assez large (m grand) permettant de retrouver des zones similaires même si elles sont éloignées dans les séquences. Lorsque les pSCR sont délimités, ils peuvent être affinés de la façon suivante : le fichier GAP.LIS contenant les séquences avec les espaces introduits sert de référence à la première étape de l'alignement, donc comme fichier d'entrée de la

procédure SEQUENCE. Comme les espaces principaux sont déjà introduits, l'amplitude du balayage m peut être fortement réduite, ce qui a pour effet de diminuer considérablement le bruit de fond.

Un alignement multiple ne sert pas seulement à trouver des pSCR, mais également à regrouper les séquences selon leur similarité. Le programme *Match-Box* propose de faire cela par une analyse de rassemblement ("cluster analysis") ou une analyse en coordonnées principales. Les procédures correspondantes peuvent être omises si les groupes sont définis selon des connaissances *a priori* des séquences ou si on ne s'intéresse pas à ce regroupement.

Situons maintenant les différentes étapes dans l'algorithme. Les procédures **SYMMETRIC** et **FACTOR** font l'analyse en coordonnées principales, qui va relativement vite. La procédure de **CLUSTERING** prend plus de temps et ne doit pas être employée trop souvent. Cette procédure est plus sensible au bruit de fond et ne peut pas être utilisée si m>15.

Expliquons maintenant plus en détails les différentes étapes.

2.3.1 SIMIL

La procédure **SIMIL** produit une matrice de similarité S de dimension r×r qui servira de donnée pour l'analyse de rassemblement et l'analyse en coordonnées principales. La matrice S sera déterminée à partir de la base de données des groupes.

L'élément S_{ij} est égal au nombre total de groupes dont la fenêtre initiale se trouve en la séquence i et la fenêtre mobile en j. Les éléments de la diagonale valent 0 car un groupe est toujours formé de deux fenêtres appartenant à deux séquences différentes. Comme la procédure de "matching" n'est pas symétrique, S n'est pas symétrique.

2.3.2 Analyse de rassemblement ("CLUSTERING")

En fait, cette analyse est faite dans la procédure SIMIL. Elle sera omise si on ne s'intéresse qu'à l'analyse en coordonnées principales, on passe alors directement à SYMMETRIC.

Les deux séquences avec la plus grande similarité sont regroupées, c'est-à-dire p et q (p≠q) tel que $S_{pq} = \max_{i \neq j} S_{ij}$. Une nouvelle matrice S'_{ij} de dimension (r-1)×(r-1) sera créée en rassemblant les indices i et j.

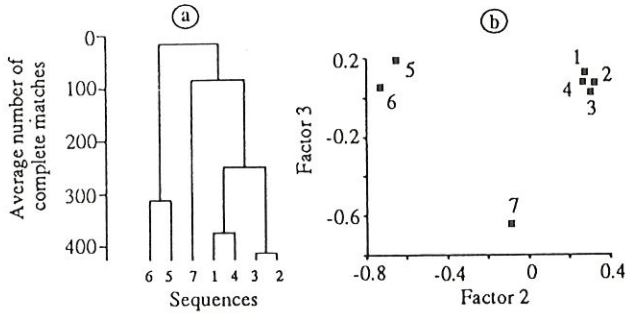
Après on refait la même chose avec S' et ainsi de suite jusqu'à ce qu'on a réduit la matrice S à une matrice de dimension 1×1.

On peut présenter le résultat dans un diagramme spécifique comme par exemple celui de la figure 2.5 (a).

Figure 2.5: Détermination des groupes de séquences.

(a) Diagramme obtenu par la procédure du "clustering" pour les sept séquences de l'axe X. L'axe Y représente le nombre moyen de groupes complets dans un groupe.

(b) Le plan des facteurs 2 et 3 extraits par l'analyse en coordonnées principales montre des groupes similaires. Les coordonnées sont les vecteurs propres normés par les valeurs propres.



2.3.3 SYMMETRIC

Cette procédure lit la matrice S créée par SIMIL et la transforme en une matrice R de dimension r×r symétrique. La transformation sera faite de la manière suivante :

$$R_{ij} = R_{ji} = \frac{\min\{S_{ij}, S_{ji}\}}{\sqrt{S_{ii} S_{jj}}}$$

où S_{ii} (S_{jj}) est le nombre de fenêtres initiales en la séquence i (respectivement j), donc $S_{ii} = L_i$-w+1 (respectivement $S_{jj} = L_j$−w+1).

R mesure la similarité entre les séquences i et j indépendamment de la longueur des séquences. R_{ij} estime la proportion de résidus pouvant être inclus dans des boîtes pour la paire de séquences correspondante. Cette estimation est influencée par les paramètres de la procédure MATCHING. La valeur correcte peut seulement être obtenue pour m=0.

2.3.4 FACTOR

Ce programme fait une analyse en coordonnées principales de la matrice R (Gower, *in Sneath and Sokal*, 1973 [29]). La décomposition en valeurs propres et en vecteurs propres de R est faite par l'algorithme de Sparks et Todd (in Hill and Griffiths, 1985 [19]).

Des séquences similaires sont représentées par des valeurs proches du vecteur propre. Chaque vecteur propre est associé à une valeur propre proportionnelle à la similarité des séquences (Figure 2.5 (b)).

2.4 L'efficacité de l'algorithme

De nombreux tests de performance de la méthode de la distance-minimale ont été réalisés par des alignements de séquences de protéines de structure connue. Dans 77% des cas, les régions des protéines où les séquences étaient les plus similaires (trouvées par la méthode de la distance-minimale) donc les pSCR coïncidaient aux régions structurellement conservées (SCR). Mais dans 23% des cas, il s'avèra que les groupes complets, non sélectionnés par la méthode de la distance-minimale, auraient donné un meilleur alignement structural. La méthode d'alignement avec la stratégie de la distance-minimale, échoue donc parfois dans la prédiction des régions structurellement conservées. Il n'existe donc *a priori* aucun argument qui permette de choisir directement les groupes complets qui correspondraient à un alignement optimal.

Il est donc nécessaire de rechercher une nouvelle stratégie dans la procédure de "SCREENING". Celle-ci devrait permettre d'inclure tous les groupes obtenus, pour une limite de similarité donnée, afin de trouver tous les groupes complets. Malheureusement, le problème devient alors extrêmement complexe, étant donné que le nombre de groupes complets potentiels augmente de façon très importante avec le nombre de séquences et de groupes trouvés par séquence. Jusqu'à présent, aucune solution algorithmique n'a été trouvée pour résoudre ce problème en un temps CPU satisfaisant.

Il faut donc trouver une méthode, autre que celle de la distance-minimale, qui permette de trouver rapidement le plus grand nombre possible de groupes complets. Une étude plus approfondie réalisée dans le cadre du développement de la méthode d'alignement a mis en évidence l'existence d'une corrélation entre cette question et un problème de la théorie des graphes : *la clique dans un graphe.* Nous expliquerons le concept de clique proprement dit, ainsi que la correspondance entre ce problème de théorie des graphes et notre problème biologique au chapitre 4.

Au chapitre 5 nous verrons différents algorithmes pour résoudre le problème de clique, en autre l'algorithme *clique* réalisé par Florence Badoux et implémenté dans l'ensemble du programme *Match-Box* par Eric Depiereux. Au chapitre 6 se trouvent les résultats de nombreux tests sur cet algorithme.

Chapitre 3

Matrices de scores

3.1 Introduction

Comme nous l'avons déjà décrit au chapitre 2, qui concernait l'alignement de séquences d'AA[1], un des problèmes le plus important dans l'analyse des séquences biologiques est la recherche de séquences ou de fragments de séquences similaires et la mesure de leur similarité. La comparaison de séquences ne peut pas se limiter aux seules identités de résidus, car certains AA se ressemblent plus que d'autres, et leur remplacement est mieux toléré. La notion de similarité entre AA est exprimée par une matrice de scores (de similarité ou de distance) où chaque terme exprime respectivement la ressemblance ou la différence, entre les AA correspondants. La valeur numérique exacte de la mesure de similarité dépend de la matrice de scores choisie.

La matrice de Dayhoff est une matrice de similarité souvent utilisée lorsqu'on travaille avec des séquences d'AA. Dans ce chapitre nous montrerons comment on a construit cette matrice et nous prouverons son optimalité (Leontovich [25]). Dans l'alignement multiple décrit au chapitre 2, une autre matrice de scores est souvent utilisée, celle de Depiereux et Feytmans; nous verrons également sa construction dans ce chapitre (en 3.4).

Il est important de signaler qu'une matrice de scores n'est pas un terme propre aux AA. On peut également utiliser une matrice de scores lors de l'alignement de séquences polynucléotidiques. Dans ce cas, le score mesure la similarité (respectivement la distance) entre les différents nucléotides.

Tout d'abord voyons la construction générale d'une matrice de scores X pour les AA.

[1] acide aminé

3.2 Construction d'une matrice de scores X

La comparaison de séquences de protéines est basée sur la comparaison des paires d'AA qui sont mises en correspondance par un processus d'alignement. Il faut donc exprimer la similarité ou la différence qui existe entre deux AA bien particuliers par un score qui sera attribué à chaque paire d'AA. Comme il existe 20 AA différents, la matrice de scores X contient 400 valeurs (20×20). Les 20 valeurs de la diagonale correspondent aux identités et les 380 valeurs restantes aux paires non identiques. Parmis ces 380 valeurs, seulement 190 valeurs sont originales car la matrice de scores est symétrique. En effet, la paire CH est caractérisée par la même valeur que la paire HC. La matrice de scores est donc en réalité une demi-matrice de 210 valeurs ($190 + 20$) dont les termes expriment la différence ou la similarité entre les deux AA correspondants.

Remarquons que la matrice d'identité (I_{20}) est la matrice de scores la plus ancienne. Actuellement elle n'est plus beaucoup utilisée, car elle ne tient pas compte des similarités entre les résidus.

La matrice de scores peut être établie de différentes façons.

1. A partir de caractéristiques physicochimiques des AA, par exemple l'hydrophobicité, la charge, etc.
   La matrice de Depiereux et Feytmans a été construite à partir de ces critères (3.4).

2. A partir de la statistique de la substitution des différents AA au cours de l'évolution.
   La matrice de Dayhoff a été construite à partir de ce critère (3.3.1).

3. A partir du minimum de mutations à réaliser pour passer d'un AA à un autre.

Il existe deux sortes de matrices de scores : *les matrices de distance et les matrices de similarité.* Les matrices de distances expriment la différence entre deux AA, le score est donc d'autant plus faible que les AA sont proches. Les matrices de similarités expriment la ressemblance entre deux AA, le score est donc d'autant plus élevé que les AA sont proches. Remarquons qu'on peut construire facilement une matrice de distance à partir d'une matrice de similarité et l'inverse par une opération mathématique.

3.3 La matrice de Dayhoff

3.3.1 Construction de la matrice de Dayhoff

La matrice de Dayhoff (Table 3.1) est couramment utilisée dans les alignements de séquences. Elle est construite à partir du deuxième critère annoncé en 3.2. Comme c'est une matrice

Table 3.1: Matrice de similarité de Dayhoff

	Gly	Pro	Asp	Glu	Ala	Asn	Gln	Ser	Thr	Lys	Arg	His	Val	Ile	Met	Cys	Leu	Phe	Tyr	Trp
Gly	29																			
Pro	12	14																		
Asp	11	10	23																	
Glu	11	11	19	20																
Ala	13	14	12	12	14															
Asn	11	10	14	12	12	17														
Gln	10	11	14	14	11	12	21													
Ser	12	11	12	11	13	13	11	13												
Thr	10	10	10	10	12	12	11	13	18											
Lys	7	8	10	10	9	12	11	10	9	24										
Arg	4	4	7	7	5	10	13	7	6	22	75									
His	6	6	9	8	8	13	12	9	9	11	20	59								
Val	7	8	7	8	10	8	9	9	11	8	5	6	22							
Ile	6	6	6	7	8	7	8	8	10	7	5	7	21	25						
Met	5	6	6	6	8	7	8	8	9	8	10	6	16	16	26					
Cys	6	4	5	5	7	6	5	11	9	4	2	3	11	9	12	166				
Leu	4	4	5	6	6	5	6	6	7	6	4	6	15	18	21	5	40			
Phe	2	2	2	3	4	3	3	4	5	3	4	7	7	11	11	2	12	70		
Tyr	1	1	1	1	2	2	2	3	2	3	3	7	3	6	6	1	6	66	137	
Trp	1	1	1	1	1	2	2	2	2	2	3	15	2	4	4	1	3	41	46	414

de similarité, les termes élevés correspondent à deux AA qui ont tendance à la substitution mutuelle. Dayhoff se base sur l'observation de substitutions d'AA acceptées au cours de l'évolution dans des familles de protéines homologues.

Construction

Soient α et β deux protéines[2] choisies aléatoirement parmi une famille de protéines. Soit a, b une paire d'AA et soit N le nombre de positions occupées par des AA dans les deux chaînes d'AA (et non par des espaces ("gaps"). Notons N_a^α (respectivement N_b^β) le nombre de positions occupées par a (respectivement b) dans α (respectivement β) et $N_{ab}^{\alpha\beta}$ le nombre de positions occupées par a dans α et par b dans β.

Nous définissons $\rho_{ab}^{\alpha\beta}$ comme suit

$$\rho_{ab}^{\alpha\beta} = \frac{N_{ab}^{\alpha\beta} \cdot N}{N_a^\alpha \cdot N_b^\beta} = \frac{\frac{N_{ab}^{\alpha\beta}}{N}}{\frac{N_a^\alpha}{N} \cdot \frac{N_b^\beta}{N}} \tag{3.1}$$

[2]une protéine est une chaîne d'AA

Remarquons que si $\rho_{ab}^{\alpha\beta} > 1$ alors les deux AA a et b ont tendance à occuper la même position dans les deux chaînes d'AA α et β respectivement.

Pour a et b fixé, on calcule la valeur de $\rho_{ab}^{\alpha\beta}$ pour toutes les paires de protéines α et β fixée de la famille initiale de protéines, ensuite, on calcule la moyenne arithmétique de tous les $\rho_{ab}^{\alpha\beta}$ que l'on note ρ_{ab} et qui sera l'élément de la matrice de Dayhoff correspondant aux AA a et b.

Remarquons que la matrice de Dayhoff est bien une matrice symétrique. En effet $\rho_{ab} = \rho_{ba}$, même si $\rho_{ab}^{\alpha\beta} \neq \rho_{ba}^{\alpha\beta}$ car pour le calcul de ρ_{ab} et ρ_{ba} on prend la moyenne pour toutes les paires (α, β) de protéines.

Pour mieux saisir les nouvelles notions, illustrons celles-ci par l'exemple suivant.

Exemple :

Pour cet exemple j'ai pris des séquences au hasard, il ne correspondent nullement à des protéines et pour cette raison la valeur trouvée pour ρ_{ab} ne correspond pas à l'élément ij de la matrice de scores de Dayhoff.

Soit la famille des 2 protéines α et β, où

$$\alpha = Gln\ Ser\ Thr\ Ser\ Ser$$

$$\beta = Ser\ Gln\ Gln\ Cys\ Cys$$

Choisissons $a = Ser$ et $b = Gln$. On a alors

$$\begin{aligned} N_a^\alpha &= N_{Ser}^\alpha &&= 3 \\ N_b^\beta &= N_{Gln}^\beta &&= 2 \\ N_{ab}^{\alpha\,\beta} &= N_{SerGln}^{\alpha\beta} &&= 1 \\ N &= 5 \end{aligned}$$

Donc en appliquant la définition de $\rho_{ab}^{\alpha\beta}$ on trouve

$$\rho_{ab}^{\alpha\beta} = \frac{N_{ab}^{\alpha\beta} \cdot N}{N_b^\alpha \cdot N_b^\beta} = \frac{5}{6}$$

De même, on a

$$N_a^\beta = 1$$

$$N_b^\alpha = 1$$

$$N_{ab}^{\alpha\beta} = 1$$

et on trouve pour $\rho_{ab}^{\beta\alpha}$ le résultat suivant :

$$\rho_{ab}^{\beta\alpha} = \frac{1 \cdot 5}{1 \cdot 1} = 5$$

Finalement on peut calculer la similarité entre les deux AA *Ser* et *Gln* pour la famille des protéines de départ. On trouve :

$$\rho_{ab} = \frac{\rho_{ab}^{\alpha\beta} + \rho_{ab}^{\beta\alpha}}{2} = \frac{\frac{5}{6} + 5}{2} = \frac{35}{12} \simeq 2,9$$

3.3.2 Définition de la similitude

Comme nous l'avons annoncé dans l'introduction, la matrice de similitude sera utilisée pour calculer la similitude entre différentes séquences polypeptidiques (c'est-à-dire séquences d'AA). Il faut encore définir la mesure de similitude. A l'heure actuelle, la définition donnée par Staden est une des plus connues, c'est pour cette raison que nous allons nous en servir dans ce chapitre.

Cette définition peut être appliquée à différentes séquences biologiques, comme par exemple des séquences polynucléotidiques et pas seulement aux séquences polypeptidiques. Nous allons donner la définition de similitude de Staden au sens général et parler de lettres appartenant à une chaîne biologique et non pas de l'AA appartenant à la chaîne polypeptidique.

Soient deux séquences biologiques $a_1, a_2, ..., a_{N_1}$ et $b_1, b_2, ...b_{N_2}$ de longueur N_1 et N_2 respectivement.
On définit la probabilité de la lettre a (respectivement b) dans la première séquence (respectivement deuxième séquence) par $p_a = \frac{N_a^1}{N_1}$ (respectivement $q_b = \frac{N_b^2}{N_2}$), où N_a^1 (respectivement N_b^2) est le nombre d'occurences de la lettre a (respectivement b) dans la première séquence (respectivement deuxième séquence).

Supposons qu'on a une matrice de similitude $P = ||\rho_{ab}||$, si en plus les séquences $a_1, a_2, ..., a_{N_1}$ et $b_1, b_2, ...b_{N_2}$ sont des séquences indépendantes de Bernoulli de probabilité p_a et q_b respectivement, alors la similitude $\rho_{a_{i+l}b_{j+l}}$ entre les lettres a_{i+l} de la première séquence et b_{j+l} de la deuxième séquence est une variable aléatoire indépendante de moyenne m et de variance D, où m et D sont définis de la façon suivante :

$$m = \frac{1}{N_1 N_2} \sum_{i=1}^{N_1} \sum_{j=1}^{N_2} \rho_{a_i b_j} = \sum_{a,b} \rho_{ab} p_a q_b$$

$$D = \frac{1}{N_1 N_2} \sum_{i=1}^{N_1} \sum_{j=1}^{N_2} (\rho_{a_i b_j} - m)^2 = \sum_{a,b} (\rho_{ab} - m)^2 p_a q_b = \sum_{a,b} \rho_{ab}^2 p_a q_b - m^2$$

Nous pouvons maintenant donner la définition de la mesure de similitude.

DEFINITION : mesure de similitude [STADEN]

Soient $a_i, a_{i+1}, ..., a_{i+L-1}$ et $b_j, b_{j+1}, ..., b_{j+L-1}$ une paire de fragments de deux séquences biologiques de longueur L, alors

$$\mathbf{A} = \sum_{l=0}^{L-1} \frac{\rho_{a_{i+l} b_{j+l-m}}}{\sqrt{LD}}$$

est la mesure de similitude définie par Staden.

Suite à cette définition, nous constatons que le score de similitude A est une variable aléatoire d'espérance 0 et de variance 1 et que la distribution de A tend vers une loi normale $N(0,1)$ pour L$\rightarrow +\infty$, ce qui nous permet d'écrire

$$P(A \geq \alpha) \sim \frac{1}{\sqrt{2\Pi}} \int_{\alpha}^{+\infty} e^{\frac{-x^2}{2}}\, dx \text{ pour } L \rightarrow +\infty$$

D'après la définition nous voyons que la valeur numérique de la similitude, A, dépend du choix de ρ_{ab} et donc du choix de la matrice de scores $P = ||\rho_{ab}||$. C'est ce que nous avons annoncé dans l'introduction.

On note souvent $A\{\rho_{ab}\}$ pour indiquer cette dépendance. La valeur de A est d'autant plus grande que la matrice P est sensible à la recherche de similarité entre les fragments. On peut se poser la question : *Quelle matrice P correspond à la valeur maximale de $A\{\rho_{ab}\}$ pour une paire de fragments donnés.* La réponse à cette question se trouve dans le paragraphe suivant.

3.3.3 L'optimalité de la matrice de Dayhoff

THEOREME :

Soient deux séquences en AA et a,b une paire d'AA.
Soit

$$\hat{\rho_{ab}} = \frac{n_{ab}}{L p_a q_b} \tag{3.2}$$

Alors pour toute matrice de similitude $||\rho_{ab}||$ on a

$$\begin{aligned} A\{\rho_{ab}\} \leq A\{\hat{\rho_{ab}}\} &= \sqrt{\frac{1}{L} \sum_{a,b} \frac{(n_{ab} - L p_a q_b)^2}{p_a q_b}} \\ &= \sqrt{L(\sum_{a,b} \frac{n_{ab}^2}{L^2 p_a q_b} - 1)} \end{aligned} \tag{3.3}$$

où n_{ab} représente le nombre d'occurences occupées par a dans la première séquence et par b dans la deuxième séquence.

p_a (q_b) la probabilité d'apparition de la lettre a (b) dans la première (deuxième) séquence.
L la longueur des deux séquences.

Remarque : Si on multiplie tout ρ_{ab} de la matrice de similitude par une constante ou si on additionne tout ρ_{ab} par une constante, alors on ne change pas la valeur de $A\{\rho_{ab}\}$.

PREUVE du théorème :

Le problème est de trouver un score $\{\rho_{ab}\}$ qui maximise la valeur de

$$A\{\rho_{ab}\} = \frac{\sum_{l=1}^{L}(\rho_{a_l b_l} - m)}{\sqrt{LD}}$$

et tel que les conditions suivantes sont vérifiées,

$$m = \sum_{a,b} \rho_{ab} p_a q_b = 0 \tag{3.4}$$

$$D = \sum_{a,b} (\rho_{ab})^2 p_a q_b - m^2 = 1 \tag{3.5}$$

Il s'agit donc de résoudre un problème d'optimisation sous contraintes. On peut simplifier le problème.

En effet, Comme, p, m, L et D sont des constantes, on peut écrire

$$\max A = \max \frac{\sum_{l=1}^{L} \rho_{a_l b_l}}{\sqrt{LD}} - \frac{\sum_{l=1}^{L} m}{\sqrt{LD}} = \max \frac{S\{\rho_{ab}\}}{\sqrt{LD}} - \frac{\sum_{l=1}^{L} m}{\sqrt{LD}}$$

et maximiser $A\{\rho_{ab}\}$ est donc équivalent à maximiser $S\{\rho_{ab}\} = \sum_{l=1}^{L} \rho_{a_l b_l}$.

On a donc un problème d'optimisation sous contrainte, qui s'écrit de la façon suivante :

$$(P) \begin{cases} \max \quad S = S\{\rho_{ab}\} = \sum_{l=1}^{L} \rho_{a_l b_l} = \sum_{a,b} n_{ab} \rho_{ab} \\ \\ S.C. \quad m = \sum_{a,b} \rho_{ab} p_a q_b = 0 \\ \qquad\quad D - 1 = \sum_{a,b} (\rho_{ab})^2 p_a q_b - m^2 - 1 = 0 \end{cases} \tag{3.6}$$

Pour résoudre le problème 3.6, on utilise la méthode des multiplicateurs de Lagrange. La fonction lagrangienne s'écrit $L = S - \lambda m - \mu(D-1)$. Le probléme dual (D) du problème primal (P) est donc le problème suivant :

$$(D) \begin{cases} \max_{\rho_{ab}} L &= \max_{\rho_{ab}} S - \lambda m - \mu(D-1) \\ &= \max_{\rho_{ab}} (\sum_{a,b} n_{ab} \rho_{ab} - \lambda \sum_{ab} \rho_{ab} p_a q_b - \mu(\sum_{ab} (\rho_{ab})^2 p_a q_b - m^2 - 1)) \end{cases} \tag{3.7}$$

Il s'agit ici d'un *problème séparable* [26]. Maximiser L est donc équivalent aux z problèmes suivants, avec $z = \sum_{a,b} 1$.

$$\max L_i = \max_{\rho_{ab}}(n_{ab}\rho_{ab} - \lambda\rho_{ab}p_a q_b - \mu(\rho_{ab})^2 p_a q_b + \frac{\mu}{z}(m^2+1))$$

Pour obtenir un point optimal $(\bar{\rho_{ab}})$, on dérive les z fonctions par rapport à ρ_{ab} et on annulle la dérivée.

On obtient alors :

$$\begin{aligned} \frac{\partial L_i}{\partial \rho_{ab}} &= n_{ab} - \lambda p_a q_b - 2\mu\rho_{ab}p_a q_b = 0 \\ \iff \bar{\rho_{ab}} &= \rho_{ab} \\ &= \frac{1}{2\mu}(\frac{n_{ab}}{p_a q_b} - \lambda) \end{aligned}$$

Comme $\hat{\rho_{ab}} = \frac{n_{ab}}{L p_a q_b}$, on peut écrire

$$\bar{\rho_{ab}} = \frac{1}{2\mu}(L\hat{\rho_{ab}} - \lambda) = \frac{L}{2\mu}\hat{\rho_{ab}} - \frac{\lambda}{2\mu}$$

On trouve donc

$$\hat{\rho_{ab}} = \frac{2\mu}{L}(\bar{\rho_{ab}} + \frac{\lambda}{2\mu})$$

où μ, L et λ sont des constantes.

On voit donc qu'on peut définir $\hat{\rho_{ab}}$ par multiplication et addition de l'optimum $\bar{\rho_{ab}}$, donc $\hat{\rho_{ab}}$ est également un score maximal pour A.

Donc $A\{\hat{\rho_{ab}}\} \geq A\{\rho_{ab}\}\ \forall\ \rho_{ab}$.

Calculons maintenant la moyenne et la variance pour $\hat{\rho_{ab}}$.

$$
\begin{aligned}
\hat{m} &= m\{\hat{\rho_{ab}}\} \\
&= \textstyle\sum_{a,b} \frac{n_{ab}}{Lp_aq_b} p_a q_b \\
&= \textstyle\sum_{a,b} \frac{n_{ab}}{L} = 1
\end{aligned}
$$

$$
\begin{aligned}
\hat{D} &= D\{\hat{\rho_{ab}}\} \\
&= \textstyle\sum_{a,b} (\frac{n_{ab}}{Lp_aq_b} - \hat{m})^2 p_a q_b \\
&= \textstyle\sum_{a,b} \frac{1}{L^2} \frac{(n_{ab} - Lp_aq_b)^2}{p_aq_b} \\
&= \textstyle\sum_{a,b} \frac{n_{ab}^2}{L^2 p_a p_b} + \sum_{a,b} p_a q_b - \sum_{a,b} \frac{2n_{ab}}{L} \\
&= \textstyle\sum_{a,b} \frac{n_{ab}^2}{L^2 p_a q_b} + 1 - 2 \\
&= \textstyle\sum_{a,b} \frac{n_{ab}^2}{L^2 p_a q_b} - 1
\end{aligned}
$$

Avec ces valeurs pour $\hat{m}$ et $\hat{D}$ on trouve pour $A\{\hat{\rho_{ab}}\}$ le résultat suivant :

$$
\begin{aligned}
\hat{A} &= A\{\hat{\rho_{ab}}\} \\
&= \frac{\sum_{l=1}^{L}(\hat{\rho_{a_l b_l}} - \hat{m})}{\sqrt{L\hat{D}}} \\
&= \frac{\sum_{a,b} n_{ab}(\hat{\rho_{ab}} - \hat{m})}{\sqrt{L\hat{D}}} \\
&= \sum_{a,b} \frac{n_{ab}(\frac{n_{ab}}{Lp_a q_b} - 1)}{\sqrt{L\hat{D}}} \\
&= (\sum_{a,b} \frac{n_{ab}^2}{L^2 p_a q_b} - \sum_{a,b} \frac{n_{ab}}{L}) \frac{L}{\sqrt{L\hat{D}}} \\
&= (\sum_{a,b} \frac{n_{ab}^2}{L^2 p_a q_b} 1) \frac{L}{\sqrt{L\hat{D}}} \\
&= \frac{DL}{\sqrt{L\hat{D}}} \\
&= \sqrt{L\hat{D}}
\end{aligned}
$$

En remplaçant $\hat{D}$ par sa valeur on trouve finalement le résultat voulu

$$
\begin{aligned}
\hat{A} &= \sqrt{L(\sum_{a,b} \frac{n_{ab}^2}{L^2 p_a q_b} - 1)} \\
&= \sqrt{\frac{1}{L} \sum_{a,b} \frac{(n_{ab} - Lp_a q_b)^2}{p_a q_b}}
\end{aligned}
\tag{3.8}
$$

□

Supposons maintenant que la probabilité d'apparition d'une lettre vaut :

$$
\begin{aligned}
p_a &= \frac{n_a^1}{L} \\
q_b &= \frac{n_b^2}{L}
\end{aligned}
\tag{3.9}
$$

(donc les fragments ont des caractéristiques des moyennes.) De l'équation 3.9 et de l'énoncé du théorème précédent il s'ensuit

$$
\hat{\rho_{ab}} = \frac{n_{ab} L}{n_a^1 n_b^2} \tag{3.10}
$$

$$
\begin{aligned}
A\{\hat{\rho_{ab}}\} &= \sqrt{\frac{1}{L}\sum_{a,b}\frac{(n_{ab}L - n_a^1 n_b^2)^2}{n_a^1 n_b^2}} \\
&= \sqrt{L(\sum_{a,b}\frac{(n_{ab})^2}{n_a^1 n_b^2} - 1}
\end{aligned}
\tag{3.11}
$$

La similarité entre les formules 3.8 et 3.10 justifie l'interprétation suivante du théorème.

Interprétation du théorème

Le choix de prendre les scores de la matrice de Dayhoff pour la substitution des lettres (AA) est près de l'optimum, si les séquences (protéines) que l'on compare ont une similarité intermédiaire qui est près du degré de similarité des familles de protéines employée pour la construction de la matrice de Dayhoff. Dans le cas contraire, il est préférable d'utiliser une autre matrice.

Exemple :
Considérons des séquences presque identiques. Pour ces séquences les valeurs n_a^1,n_a^2,n_{aa} sont presque les mêmes, mais n_{ab} est petit ($b \neq a$) comparé à n_a^1 ou n_a^2.

La formule 3.10 montre qu'une matrice de scores optimale est la matrice dont les éléments de la diagonale valent p_a et les autres 0.

Remarque : Pour de tels séquences, le choix de la matrice de scores est sans importance parce qu'on trouvera certainement des fragments similaires avec la matrice d'unité, la matrice de Dayhoff ou tout autre matrice de scores.

Modifications de la définition de similarité

Les formules 3.10 et 3.2 suggèrent des modifications possibles de l'approche de Staden de la définition de la similarité mesurée entre des segments.

L'idée de ces modifications est que la matrice de scores n'est pas définie une fois pour toute, mais dépend des séquences considérées. Deux approches sont possibles.

1. Durant le processus de la détermination des similarités entre fragments, on recalcule les poids. Le nouveau calcul dépent des séquences entières et pas des segments considérés.

   D'abord des segments similaires sont trouvées avec une matrice de poids ordinaire (par exemple la matrice de Dayhoff), et on aligne les séquences. Ensuite, le degré de similarité de ces séquences est déterminé. Si nécessaire, les poids sont redéfinis en

employant la formule suivante.

$$\rho'_{ab} = \frac{1}{2}(\frac{N_{ab}N}{N_a^1 N_b^2} + \frac{N_{ba}N}{N_b^1 N_a^2})$$

où N est la longueur des séquences à aligner, N_a^1 (N_b^2) le nombre de positions occupées par la lettre a (b) dans la première (deuxième) séquence et N_{ab} le nombre de positions occupées par la lettre a dans la première séquence et par b dans la deuxième.

Puis une recherche plus précise des fragments similaires a lieu avec ces nouveaux scores ρ'_{ab}.

Le désavantage de cette méthode est qu'elle est trop compliquée, parce qu'on doit recalculer chaque fois la matrice de scores avant d'aligner les séquences pour la deuxième fois.

2. Les poids dépendent directement des fragments comparés et non des séquences toutes entières. On suggère d'employer comme matrice de scores les valeurs $\hat{\rho_{ab}}$ qui maximisent la similarité mesurée (Formule 3.2).

   La similarité est définie par la formule 3.3 tel que $\hat{A} = A\{\hat{\rho_{ab}}\}$.

   La probabilité des lettres peut être calculée soit par les séquences toutes entières ou par les fragments considérés. Dans ce cas $\hat{A}$ et $\hat{\rho_{ab}}$ sont définis par les formules 3.10 et 3.11.

   Il est encore mieux de calculer $\hat{A}^2$ au lieu de $\hat{A}$.

   Pour cette approche, on ne recalcule pas tous les poids, pour cela elle est moins encombrant. En plus, la simplicité des formules 3.3 et 3.11 est attractive.

Finalement, les deux méthodes ont des inconvénients (entre autre le fait qu'elles n'ont pas été analysées en détail). D'un autre côté, le choix de la matrice de scores n'est pas tellement important pour les biologistes, car souvent des matrices construites sur des critères différentes donnent les mêmes résultats.

3.4 La matrice de distances de Depiereux et Feytmans

E. Depiereux et E. Feytmans ont établi une matrice de scores pour les 20 AA. Ces scores sont utilisés dans leur méthode de prédiction de régions structurellement conservées (Chapitre 2).

Il s'agit d'une matrice de distance tenant compte des propriétés physicochimiques. Pour chacune des 190 paires univoques de résidus, ils ont calculé le score en se basant sur les

facteurs physicochimiques fournis par Kidera (1985) [23], les 20 couples identiques se voient attribuer la valeur minimale de distance, c'est-à-dire 0.

Kidera a fait une analyse factorielle de 188 propriétés chimiques et physiques des AA, comme le poids, la surface accessible, le volume des résidus, la taille, l'entropie absolue, l'hydrophobicité, etc. Il a pu en déduire 10 facteurs indépendants rendant compte de l'essentiel de l'information contenue dans les 188 propriétés de départ. En fait, les 10 facteurs représentent 86% de la variabilité des 188 propriétés de départ. Ces facteurs sont corrélés respectivement à l'encombrement stérique, à l'hydrophobicité, à des préférences de structure secondaire (α ou β) et à d'autres combinaisons de différentes propriétés physicochimiques des AA. Par construction, ils sont orthogonaux et standardisés.

Pour chacune des 190 paires univoques de résidus il s'agit de définir une distance entre les deux AA i et j à partir d'un des dix facteurs physicochimiques. Cette distance (ou dissimilarité) est donnée par :

$$d_{ijk} = |z_{ik} - z_{jk}|$$

où z_{ik} (z_{jk}) est la valeur du facteur k caractérisant le résidu i (j).

Plus la dissimilarité entre les deux AA est importante, plus cette différence est élevée (en valeur absolue). Si les séquences sont générées aléatoirement à partir des 20 AA indépendants, la distribution de probabilité de $\frac{d_{ijk}}{\sqrt{2}}$ est approximativement une distribution normale de moyenne 0 et de variance 1. Le carré $\frac{d_{ijk}^2}{2} = \frac{(z_{ik}-z_{jk})^2}{2}$ suit approximativement une distribution de probabilité χ^2 d'un degré de liberté, donc

$$\frac{d_{ijk}}{\sqrt{2}} \sim N(0,1)$$

$$\frac{d_{ijk}^2}{2} \sim \chi_1^2$$

La distance physicochimique entre deux AA i et j pour les 10 facteurs physicochimiques de Kidera est la somme des distances $\frac{d_{ijk}^2}{2}$ pour les dix facteurs :

$$D_{ij}^2 = \sum_{k=1}^{10} \frac{d_{ijk}^2}{2}$$

D_{ij}^2 suit une loi χ^2 avec 10 degrés de liberté. E. Depiereux et E. Feytmans ont calculé cette distance physicochimique pour chaque paire d'AA univoque et ainsi créé leur matrice de scores (Table 3.2).

Table 3.2: Matrice physicochimique de Depiereux et Feytmans. Les scores sont multipliés par 100 et arrondis à l'unité.

	Ala	**Cys**	**Asp**	**Glu**	**Phe**	**Gly**	**His**	**Ile**	**Lys**	**Leu**	**Met**	**Asn**	**Pro**	**Gln**	**Arg**	**Ser**	**Thr**	**Val**	**Trp**	**Tyr**
Ala	0																			
Cys	114	0																		
Asp	64	121	0																	
Glu	41	121	44	0																
Phe	81	111	90	97	0															
Gly	119	143	126	133	138	0														
His	124	124	58	107	99	180	0													
Ile	71	121	111	106	67	156	122	0												
Lys	111	179	113	90	98	130	110	147	0											
Leu	49	116	104	88	54	148	134	104	75	0										
Met	104	105	128	91	108	128	73	116	95	121	0									
Asn	92	123	61	83	60	102	102	57	104	136	123	0								
Pro	120	117	102	152	105	169	137	150	118	98	140	102	0							
Gln	93	132	125	82	99	159	134	73	75	129	77	19	101	0						
Arg	5136	100	99	81	129	168	135	114	93	112	159	105	161	112	0					
Ser	69	145	57	93	140	127	100	82	112	107	116	88	77	97	118	0				
Thr	69	71	82	80	108	103	98	65	70	74	81	76	77	58	51	34	0			
Val	76	127	106	118	100	123	81	48	93	73	88	109	141	113	108	52	37	0		
Trp	139	164	125	99	125	172	169	124	163	130	104	144	143	134	139	107	111	140	0	
Tyr	124	128	75	111	49	102	83	76	81	84	112	64	105	118	77	85	69	64	80	0

Chapitre 4

Le problème de clique

4.1 Introduction

A la fin du chapitre 2 (en 2.4), nous avons vu que la méthode d'alignement *Match-Box* avait certaines limites. En effet, la méthode originale, avec la stratégie de la distance-minimale, échoue parfois dans la prédiction des régions structurellement conservées.

Une idée d'amélioration de l'algorithme est d'inclure toutes les "matches" obtenus pour une limite de similarité et non un seul "match" par séquence pour chaque fenêtre initiale, afin de trouver tous les groupes complets.

Le problème devient alors extrêmement complexe, étant donné que le nombre de groupes complets potentiels augmente de façon très importante avec le nombre de séquences et de groupes trouvés par séquence. En effet, avec la méthode de la distance-minimale, on pouvait trouver au maximum un groupe complet par fenêtre initiale, maintenant on peut en trouver $m_2 * ... * m_r$ pour une fenêtre initiale se trouvant dans la première séquence et avec m_i le nombre de "matches" trouvés dans la $i^{ième}$ séquence pour cette fenêtre initiale.

Il faut donc trouver une méthode permettant de trouver le plus rapidément le plus grand nombre possible de groupes complets. Il s'agit de chercher dans la matrice de vérité ("truth-matrice") les sous-matrices entièrement remplies de 1 composée d'un élément de chaque séquence.

C'est un problème de la théorie des graphes : *La recherche des cliques maximales dans un graphe.*

Dans ce chapitre, nous verrons le concept du problème de clique et au chapitre suivant, nous verrons différents algorithmes pour résoudre ce problème.

4.2 Problème des cliques en théorie des graphes

4.2.1 Le contexte de la théorie des graphes

La théorie des graphes a été développée dans un premier temps comme une théorie abstraite de la mathématique. Ses premières applications étaient la résolution des puzzles combinatoires, comme ceux d'Euler (1736) [15] et d'Hamilton (1856) [18]. Leurs recherches, ainsi que les recherches de Kirchhoff (1847) [24] et de Cayley (1857) [11] qui ont développé indépendamment la théorie des arbres, ont permis la fondation de la théorie des graphes.

Plus récemment, la théorie des graphes est devenue un instrument important dans beaucoup de domaines de la mathématique appliquée pour résoudre des problèmes d'optimisation.

Une liste partielle des applications, pour lesquelles le développement de la théorie des graphes montre un grand avantage, comprend par exemple des problèmes de route [5], la tomography informatisée [29], des problèmes d'emplacement [19] et des problèmes de société [31].

La liste incomplète des références montre la contribution considérable de la théorie des graphes pour résoudre des problèmes d'optimisations (D.Burton [8]).

Avant de décrire le problème de clique proprement dit, précisons quelques notions élémentaires de la théorie des graphes.

4.2.2 Notions élémentaires

Definitions

Nous allons donner ici quelques définitions des termes de la théorie des graphes que nous utiliserons. Il n'existe malheureusement pas de terminologie standard. Celle que nous présentons ici est celle du cours de théorie des graphes de Monsieur Callier qui se base sur les livres de référence [4], [12], [32] et [22].

- **Un graphe** $G = (X, U)$ consiste en un ensemble fini non vide $X = \{x_i\}$ **de sommets** et un ensemble fini $U = (u_j)$ **d'arcs.**
- **Un arc** est une branche orientée reliant deux sommets.
  Par exemple $u_j = (x_i, x_k)$ est l'arc dont **l'extrémité initiale** est x_i et **l'extrémité terminale** est x_k.
- **L'ordre de G** est le nombre de sommets du graphe G, donc # X.
- **Un multigraphe** $\bar{G} = (X, \bar{U})$ consiste en un ensemble fini non vide $X = \{x_i\}$ **de sommets** et un ensemble fini $\bar{U} = (\bar{u_j})$ **d'arêtes.**

- **Une arête** est une branche non orientée reliant deux sommets appelés **ses extrémités.** Comme l'arête est non orientée, on ne fait plus la différence entre une extrémité initiale et terminale et on peut écrire $\bar{u_j} = (x_i, x_k) = (x_k, x_i)$, ce qui n'était pas le cas pour les arcs.

- **L'ordre de** $\bar{G}$ est le nombre de sommets du graphe $\bar{G}$, donc # X.

En théorie des graphes, le concept de graphe est utilisé pour des concepts orientés comme par exemple la circulation. Des multigraphes sont utilisés pour des concepts non orientés. Le problème de clique que nous traitons ici est un concept non orienté, car on suppose que si A est similaire à B, alors B est similaire à A. Il est donc seulement important de savoir que A et B sont similaires, et on n'a pas d'orientation. Pour cette raison nous allons seulement développer la notion de multigraphe et non celle de graphe. Dorénavant, comme il n'y a pas de confusion possible, nous allons parler de graphe à la place de multigraphe et on le note aussi $G = (X, U)$ à la place de $\bar{G} = (X, \bar{U})$ pour faciliter l'écriture. Il est cependant important de ne pas oublier qu'il s'agit d'un concept non orienté et que U représente l'ensemble des arêtes et non des arcs.

- **Un sous-graphe** $G_{X'} = G' = (X', U')$ d'un graphe $G = (X, U)$ est un graphe pour lequel $X' \subseteq X$ est l'ensemble des sommets et $U' = \{u = (x_i, x_j) \in U \mid x_i, x_j \in X'\}$ est l'ensemble des arêtes.

- L'arête u est **incidente** au sommet x, si l'une des deux extrémités de u est x.

- Deux sommets joints par une arête sont appelés **sommets adjacents** ou **sommets voisins.**

- **L'application Γ(W)**, pour un ensemble de sommets W, donne l'ensemble des sommets adjacents aux sommets de W.
  $\delta(x)$ est l'ensemble des arêtes dont une extrémité est le sommet $x \in X$.
  Le nombre # $\delta(x)$ est **le degré du sommet** $x \in X$.

- **Un graphe est complet** si tous les sommets pris deux à deux sont joints par une arête, autrement dit, si tous les sommets sont adjacents deux à deux.

- **Le graphe complémentaire** G^c d'un graphe G est un graphe qui a le même ensemble de sommets et dans lequel deux sommets sont adjacents si et seulement s'ils ne sont pas adjacents dans G.

- Un ensemble de **sommets indépendants** est un ensemble de sommets dont deux sommets ne sont pas adjacents.

- **Une clique** dans un graphe G=(X,U) est un sous-ensemble de sommets de X dont tous les sommets sont adjacents deux à deux.

Comme la terminologie en théorie des graphes n'est pas normalisée, il existe d'autres définitions pour une clique qui sont couramment utilisées et qui sont plus ou moins équivalentes.

X' est une **clique** du graphe G si $G_{X'}$ est un sous-graphe complet [maximal]. Pour certains auteurs, une clique n'engendre donc pas seulement un graphe complet, mais en plus ce graphe doit être maximale, en ce sens qu'il n'est inclu dans aucun autre. D'autres auteurs font la différence entre clique et clique maximale.

On peut également dire, qu'une clique [maximale] dans un graphe G est un ensemble de sommets indépendants [maximaux] dans le graphe complémentaire G^c.

- On peut définir **la densité d'un graphe** comme le nombre de sommets de la clique maximale.

- On dit qu'un graphe est **dense** si ce nombre est élevé et il est **"sparse"** dans le cas contraire.

- **Une chaîne** $\mu = (u_1, ..., u_q)$ est une séquence finie d'arêtes tel que les arêtes u_i et u_{i+1} ont une extrémité en commun.
  La chaîne peut être définie par la suite des sommets rencontrés le long de le chaîne, donc $\mu = (x_1, ..., x_{q+1})$, avec $u_i = (x_i, x_{i+1})$.
  On note $\mu[s,t]$ la chaîne partant du sommet s et arrivant au sommet t.

- **Un cycle** $\sigma = (u_1, ..., u_q) = (x_1, ...x_q, x_1)$ est une chaîne partante et aboutissante au même sommet. On l'appelle aussi parfois **chaîne fermée.**

- Deux sommets s et t d'un graphe G sont dits **connectés ou simplement connexes** s'il existe dans G une chaîne $\mu = [s,t]$ reliant s et t.

- G est appelé **connecté ou simplement connexe** si tous les sommets de G sont connectés deux à deux.

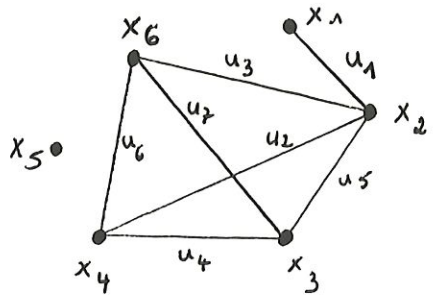
On parle de **relation de connexité simple R**, qui se définit de la façon suivante :

$$s \quad R \quad t \iff s = t \quad ou \quad \exists \quad \mu[s,t]$$

R est une relation d'équivalence (réflexif, symétrique, transitif).

- **Les composantes simplement connexes de G (C.S.C.)** sont des classes d'équivalences pour R, ce sont des sous-graphes connectés maximaux du graphe G.
  On trouve les C.S.C. de la façon suivante :

Figure 4.1: Exemple



On part d'un sommet x_1 de X et on cherche $C_1 = C(x_1)$ l'ensemble des sommets connexes avec x_1. Dans le sous-graphe restant, c'est-à-dire $G' = (X', U')$ avec $X' = X \backslash C(x_1)$ on recommence avec un point x_2 et on trouve $C_2 = C(x_2)$, et ainsi de suite, jusqu'à ce que le sous-graphe restant est vide. On a trouvé toutes les C.S.C. de G : C_1, C_2,...,C_i.

Illustration des définitions

Pour une meilleure familiarité avec les nouveaux concepts et notations, nous vous proposons un exemple concret.

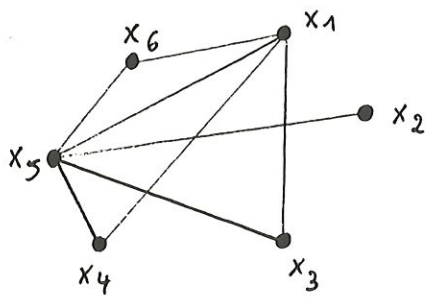
Soit le graphe $G = (X, U)$, repris à la figure 4.1, avec $X = \{x_1, x_2, x_3, x_4, x_5, x_6\}$, l'ensemble des sommets et $U = \{u_1, u_2, u_3, u_4, u_5, u_6, u_7\}$, l'ensemble des arêtes.

- L'ordre de G est 6, comme il y a 6 sommets.

- On peut dire :
  L'arête u_1 est incidente à x_1 et x_2.
  L'arête u_2 est incidente à x_2 et x_4.
  L'arête u_3 est incidente à x_2 et x_6.
  L'arête u_4 est incidente à x_3 et x_4.
  L'arête u_5 est incidente à x_2 et x_3.
  L'arête u_6 est incidente à x_4 et x_6.
  L'arête u_7 est incidente à x_3 et x_6.

- De même, on peut dire :
  x_1 et x_2 sont deux sommets adjacents.
  x_2 et x_4 sont deux sommets adjacents.
  x_2 et x_6 sont deux sommets adjacents.
  x_3 et x_4 sont deux sommets adjacents.
  x_2 et x_3 sont deux sommets adjacents.

Figure 4.2: Exemple



x_4 et x_6 sont deux sommets adjacents.
x_3 et x_6 sont deux sommets adjacents.

- Soit $W = \{x_1, x_3\}$, alors $\Gamma(W) = \{x_2, x_4, x_6\}$.
  $\Gamma(\{x_4\}) = \Gamma_{x_4} = \{x_2, x_3, x_6\}$
  $\delta(x_4) = (u_2, u_6, u_4)$ et le degré de x_4 vaut $\#\delta(x_4) = 3$.

- G n'est pas complet, car par exemple x_1 et x_3 ne sont pas liés par une arête.

- $G' = (X', U')$ où $X' = \{x_2, x_3, x_4, x_6\}$ et par définition $U' = (u_2, u_3, u_4, u_5, u_6, u_7)$ est un sous-graphe complet de G. X' est une clique maximale de G. D'autres cliques (non maximales) sont par exemple les cliques $\{x_2, x_3\}$, $\{x_1, x_2\}$ ou $\{x_2, x_3, x_4\}$.

- Le graphe complémentaire de G, G^c est le graphe de la figure 4.2.

- Un ensemble de sommets indépendants dans G, comme par exemple $\{x_1, x_3\}$ est une clique de G^c.

- Un ensemble de sommets indépendants dans G^c, comme par exemple $\{x_2, x_3, x_4, x_6\}$ est une clique de G.

- $\mu = (u_1, u_2, u_6) = (x_1, x_2, x_4, x_6)$ est une chaîne de G. Remarquons qu'on peut trouver plusieurs chaînes différentes liant les mêmes sommets, par exemple $\mu = [x_1, x_6]$ peut être représentée par

$$\begin{array}{lllll} & \mu & = & (u_1, u_2, u_6) & = & (x_1, x_2, x_4, x_6) \\ ou & \mu & = & (u_1, u_3) & = & (x_1, x_2, x_6) \\ ou & \mu & = & (u_1, u_5, u_7) & = & (x_1, x_2, x_3, x_6) \\ ou & \mu & = & (u_1, u_5, u_4, u_2, u_3) & = & (x_1, x_2, x_3, x_4, x_2, x_6) \\ etc. & & & & & \end{array}$$

La chaîne $\mu = (u_1, u_3)$ est la chaîne de longueur minimale.

- G n'est pas simplement connexe, car il existe des sommets qui ne peuvent pas être liés par une chaîne, comme par exemple x_3 et x_5.
  Les composantes simplement connexes de G sont les suivantes :

$$\begin{aligned} C_1 &= C(x_1) = \{x_1, x_2, x_3, x_4, x_6\} \\ C_2 &= C(x_5) = \{x_5\} \end{aligned}$$

4.2.3 Enoncé du problème de clique proprement dit

Quand nous avons défini une clique, nous avons fait la différence entre une clique et une clique maximale.

Soit S $\subseteq$ X un ensemble de sommets. S forme une clique si et seulement si le sous-graphe associé à S, $G_S = (S, U')$ est complet, avec $U' = \{u = (x_i, x_j) \in U \mid x_i, x_j \in S\}$

Souvent il est important de trouver la clique maximale d'un graphe. Lorsque nous parlons du problème de clique maximale, nous faisons référence à un problème d'obtimisation combinatoire bien connu, qui est le suivant :
Etant donné un graphe non orienté défini par un ensemble de sommets X et un ensemble d'arêtes U, déterminer une clique ayant le nombre maximum de sommets.
Autrement dit, étant donné G=(X,U), trouver une clique maximale dans G, c'est-à-dire le plus grand sous-graphe complet contenu dans G. (Melhorn K., 1984 [27])

A ce problème d'optimisation peut être associé un problème de décision qu s'énonce comme suit :
Etant donné un graphe non orienté G=(X,U) à n sommets, et un entier k. Existe-t-il une clique de taille k dans G, c'est-à-dire peut-on trouver $X' \subseteq X$, avec $\#X' = k$, tel que $G_{X'}$ est complet, c'est-à-dire $\forall x_i, x_j \in X'\ \exists\ \mu = (x_i, x_j) \in U$? (Melhorn K., 1984 [27])

4.2.4 Algorithme trivial pour résoudre le problème de clique

Il existe un algorithme trivial pour solutionner ce problème de cliques. Ce problème stipule qu'il suffit de générer tous les sous-ensembles de sommets $X' \subseteq X$ de cardinalité k et de regarder si le sous-graphe induit $G_{X'}$ forme une clique.

Si on veut chercher la clique maximale, on augmente k jusqu'à ce qu'on ne peut plus trouver de clique. Le désavantage de cette méthode est qu'il y a $\binom{n}{k}$ sous-ensembles X' à k éléments (n=#X). Par conséquent, nombreux sont les candidats pour une clique de taille k. Vérifier si un sous-ensemble de sommets $X' \subseteq X$ forme une clique, c'est-à-dire vérifier si un candidat est vraiment une solution, se fait alors assez facilement. Toutefois, la seule façon connue pour trouver une solution au problème de décision est de faire une recherche exhaustive (cfr. page 57) parmi tous les candidats.

Pour un entier $k = \frac{n}{2}$, l'algorithme trivial doit contrôler $\binom{n}{\frac{n}{2}} > \frac{2^n}{n+1}$ sous-ensembles. Le temps d'exécution de cet algorithme est donc très élevé et croît exponentiellement avec la taille des données. (Melhorn K., 1984)

Au chapitre 5, nous verrons plusieurs algorithmes plus efficaces pour chercher des cliques dans un graphe.

4.3 Résolution du problème de cliques en informatique

4.3.1 Représentation d'un graphe

Un graphe est un objet mathématique qui modélise précisément certaines situations. Pour résoudre des problèmes de la théorie des graphes à l'aide d'un programme d'ordinateur, il faut tout d'abord décider comment représenter un graphe en informatique.

Il existe deux représentations communément utilisées. Le choix d'utiliser plutôt l'une ou l'autre d'entre-elles dépend de la densité en arêtes du graphe et de la nature des opérations à réaliser.

1. **Matrice d'adjacence.**
   La représentation la plus directe pour les graphes est celle appelée matrice d'adjacence. On crée un tableau de dimension n×n (où n=#X). Les éléments mat(i,j) de cette matrice valent 1 si les sommets i et j sont adjacents et 0 sinon. Comme par définition la relation d'adjacence est symétrique (car nous traitons un concept non orienté), mat(i,j)=mat(j,i) et la matrice est donc symétrique. On peut épargner de la place en stockant seulement la moitié de cette matrice. Cependant, en pratique, on ne fait pas souvent cela, car les algorithmes sont souvent plus simples en travaillant avec la matrice entière. Les éléments de la diagonale mat(i,i) valent soit tous 0 ou bien tous 1 selon la nature du problème à résoudre.
   Pour des graphes avec des poids attribués aux arêtes, il suffit de remplacer la place des valeurs booléennes de la matrice d'adjacence par les poids correspondants.

2. **La liste d'adjacence.**
   La création d'une liste d'adjacence est une autre représentation d'un graphe qui est plus adaptée pour les graphes peu denses. L'idée est d'associer à chaque sommet du graphe une liste de tous les autres sommets qui lui sont adjacents. L'ordre des sommets dans la liste d'adjacence affecte l'ordre dans lequel les sommets sont traités par l'algorithme.

4.3.2 Complexité des algorithmes

Pour estimer l'efficacité d'un algorithme, on étudie le rapport entre la taille naturelle du problème solutionné, c'est-à-dire la quantité des données traitées (notée n ou N), et la quantité des ressources, c'est-à-dire le temps et l'espace utilisés pour la résolution du problème.

- **La fonction de complexité spatiale g : N $\longrightarrow$ N** d'un algorithme exprime l'espace maximum nécessaire g(n) pour solutionner n'importe quelle occurence d'un problème dont la longueur d'encodage vaut au plus n.
- **La fonction de complexité temporelle ou la fonction du temps d'exécution f : N $\longrightarrow$ N** d'un algorithme exprime le temps maximum f(n) nécessaire pour solutionner n'importe quelle occurence d'un problème dont la longueur d'encodage vaut au plus n.

On s'intéresse généralement à la durée moyenne et à la durée maximale d'un programme. Le temps d'exécution d'un algorithme est une **constante (c)** si la plupart des instructions du programme sont exécutées une seule fois ou seulement un très petit nombre de fois. Plus souvent, le nombre des instructions à exécuter est proportionnel au nombre des données n et dans ce cas le temps d'exécution est une fonction de n, comme par exemple un temps d'exécution **linéaire (n)**, **logarithmique** ($\log n$), **cubique** (n^3), **exponentiel** (2^n), etc.

4.3.3 Algorithmes polynomiaux et non déterministes polynomiaux

Introduction

En théorie de graphes et en analyse opérationnelle, il existe des problèmes pour lesquels nous ne savons pas garantir de trouver une solution optimale, même en utilisant les ordinateurs les plus efficaces.

La plupart de ces problèmes sont des problèmes NP-complets. Pour la classe des problèmes NP-complets, de même que pour la classe NP, aucun algorithme asymptotiquement efficace n'a encore été trouvé. Dans ce domaine, on ne travaille qu'avec des algorithmes du type exponentiels. Nous allons montrer que le problème de clique est un problème NP-complet, ensuite, nous allons voir au paragraphe 4.3.4 des méthodes de résolution de ces problèmes et au chapitre 5 des algorithmes s'y rapportants.

Mais tout d'abord définissons les nouvelles notions.

Definitions

Pour énoncer les définitions suivantes, nous nous sommes basés sur le mémoire de Florence Badoux qui a utilisé pour cette partie les articles et livres de références [17], [35], et [16].

- **Un algorithme polynomial** est un algorithme dont la fonction de complexité temporelle f(n) satisfait l'inégalité f(n) $\leq$ p(n) $\forall n \in N$, pour un polynôme p. Cet algorithme possède donc un temps polynomial d'exécution.

- **Un algorithme déterministe** est un algorithme qui peut accomplir qu'une seule chose à la fois. Quelque soit l'état qu'il a atteint, il y a au plus un état qu'il puisse atteindre consécutivement.

- **Un algorithme non déterministe** est un algorithme qui peut accomplir plusieurs choses à la fois. A un certain état, il peut atteindre plusieurs états différents et poursuivre simultanément sur chacun des états suivants.

  Par conséquent, alors qu'un algorithme déterministe doit explorer une seule alternative à la fois parmi un ensemble d'alternatives, un algorithme non déterministe examine toutes les alternatives parallèlement.

- **La classe P** comprend l'ensemble des problèmes pouvant être résolus par un algorithme déterministe qui possède un temps d'exécution polynomial.

- **La classe NP** comprend l'ensemble des problèmes pouvant être résolus par un algorithme non déterministe qui possède un temps d'exécution polynomial.

- Un problème Pr est dit **NP-dur** si l'algorithme déterministe polynomial qui le solutionne peut être utilisé pour obtenir un algorithme polynomial déterministe pour chaque problème appartenant à la classe NP, c'est-à-dire si Pr est au moins aussi dur que n'importe quel problème dur de la classe NP.

- Un problème NP-dur dans NP est appelé **NP-complet**. Ce type de problème est au moins aussi dur que n'importe quel problème de la classe NP et même souvent plus dur.

  On peut encore définir un problème NP-complet de la façon suivante :
  Un problème Pr est **NP-complet** s'il appartient à la classe NP et si tous les autres problèmes de NP peuvent être transformés en un temps polynomial en Pr, c'est-à-dire si tous les problèmes de la classe NP lui sont polynomialement réductibles.

Le problème de clique est un problème NP-complet

Kurt Melhorn ([27],p.198) a démontré que **le problème de clique est un problème NP-complet.**

4.3.4 Résolution des problèmes NP-complets

Comme nous l'avons déjà indiqué précédemment, les problèmes NP-complets sont très difficiles à résoudre, mais comme on les rencontre fréquemment dans la pratique, il est nécessaire de les solutionner. On se base sur différentes méthodes de résolution. Voici quelques approches de résolution proposées par Kurt Melhorn ([27], p.208-209).

I. Approches de résolution

- **Cas spéciaux :** on réexamine le problème de façon visuelle, puis l'on vérifie si le problème NP-complet est réellement à solutionner dans toute sa généralité, ou si l'on peut se contenter de solutionner un cas spécial. L'avantage est que le cas spécial peut avoir une solution en un temps polynomial.

- **La programmation dynamique et la technique "Branch-and-Bound"** sont deux techniques qui peuvent être appliquées à la plupart des problèmes NP-complets. Ces deux techniques sont essentiellement des variantes intelligentes de recherche exhaustive.

- **Des analyses de probabilités** peuvent parfois montrer que les occurences "difficiles" d'un problème NP-complet sont assez rares. Il est alors possible de concevoir des algorithmes avec des temps d'exécution appréciables. Bien sûr, il subsiste toujours le problème de la justification de la distribution de probabilité postulée pour les occurrences du problème.

- **Des algorithme d'approximation** peuvent parfois aboutir à de très bonnes solutions en un temps court.

- **Des algorithmes heuristiques** peuvent parfois, pour une raison inconnue, donner des résultats satisfaisants.

II. Recherche exhaustive dans un graphe

Pour beaucoup de problèmes, comme par exemple le problème de cliques, il ne suffit pas de trouver une solution, mais on veut trouver toutes les solutions ou au moins une grande partie. Pour ce faire, il est nécessaire d'appliquer une recherche exhaustive.

Lors d'une recherche exhaustive, toutes les solutions potentielles sont examinées afin de vérifier si elles correspondent ou non à la solution exacte du problème (processus de décision ou "decision-making process"). Il n'existe pas, semble-t-il, d'algorithme efficace pour solutionner les problèmes que nécessite une recherche exhaustive. Toutefois, il est parfois possible de réduire considérablement le nombre de possibilités contrôlées, en essayant de découvrir les décisions incorrectes dans le processus de décision le plus rapidement possible.

Voyons maintenant deux techniques de recherche exhaustive fort importantes sur lesquelles sont basées les algorithmes pour résoudre le problème de clique (cfr. chapitre 5). Remarquons encore que les deux techniques ("Backtracking" et "Branch-and-bound") peuvent être combinées.

Technique dite du "Backtracking".

La méthode de recherche avec rebroussement ("Backtracking") est une technique systématique de recherche exhaustive, c'est-à-dire donnant toutes les solutions. Elle explore une branche de l'arbre et, lorsqu'elle arrive dans un cul de sac, repart du dernier point de bifurcation. Les noeuds correspondent à des solutions partielles. Donc on doit analyser tous les noeuds pour trouver une solution complète. Descendre dans l'arbre correspond à progresser vers une solution plus complète (on s'approche d'une solution) et remonter correspond à faire marche arrière ("backtrack").

Le temps pris pour explorer l'arbre de toutes les possibilités ("exhaustive search tree") est proportionnel au nombre de noeuds qu'il contient et sera important lors du traitement de grands arbres.

L'idée de cette technique est de réduire le nombre de noeuds à explorer en coupant certains arbres par la suppression de noeuds de l'arbre. En fait, comme l'algorithme est récursif, le fait de supprimer un noeud annule la recherche dans tout le sous-arbre de ce noeud-racine. Une coupure dans l'arbre peut conduire à des économies significatives de temps de traitement et pour cela il faut le faire autant que possible.

Une technique importante d'élagage consiste en la suppression de symétries : il s'agit d'empêcher qu'une même solution soit trouvée plusieurs fois.

Il est généralement vrai que le temps d'exécution d'un algorithme de "backtracking" reste exponentiel, cela est dû au nombre élevé de noeuds. Par exemple, si chaque noeud dans l'arbre de recherche a en moyenne α enfants et que la longueur du chemin pour arriver à la solution est N, alors le nombre attendu de noeuds dans l'arbre sera proportionnel à α^N.

Des règles heuristiques sont parfois utilisées pour stopper la recherche dans une branche de l'arbre. Les différentes règles de "backtracking" ont pour objectif de réduire la valeur de α (c'est-à-dire réduire le nombre de choix à faire pour chaque noeud), ce qui permet de solutionner des problèmes de plus grande taille.

Technique "Branch-and-bound".

C'est également une technique d'élagage ("pruning") d'un arbre de recherche exhaustive. Le but de cette méthode est de limiter le nombre de solutions entières devant être examinées. Cela est fait de la façon suivante : En tout noeud n de l'arbre auquel on aboutit, on essaie d'aiguiller ("branch") la suite de la recherche vers une ramification particulière de l'arbre en calculant une borne locale (donc un coût minorant) et en la comparant à la borne globale obtenue jusqu'à ce stade pour le problème.

4.3.5 Définition de notre problème sous forme informatique

Notre problème biologique, la recherche de groupes complets, peut être associé au problème de recherche de cliques. Comme un groupe complet contient un segment de chaque séquence, la taille de la clique doit être au moins égale à n (si n est le nombre de séquences). D'autre part, nous savons qu'un segment d'une séquence ne peut jamais être considéré en même temps qu'un autre segment de la même séquence, donc la taille de la clique ne peut pas être strictement supérieur à n. Donc notre problème est la recherche de cliques de taille n.

Pour l'algorithme *clique*, nous utiliserons la représentation du graphe par la matrice d'adjacence.

Chapitre 5

Algorithmes pour résoudre le problème de clique

5.1 Articles intéressants de la littérature

Il existe de nombreux articles dans lesquels le concept de clique est utilisé ou des articles se rapportant aux ensembles indépendants maximaux. Florence Badoux [2] fait référence à de nombreux articles s'y rapportant.

Beaucoup de ces articles traitent le problème NP-complet de la recherche d'une clique de cardinalité la plus grande ou de toutes les cliques maximales dans un graphe (ou la recherche d'un ou de tous les ensembles de sommets indépendants maximaux).

Ces travaux s'orientent selon deux directions :

1. La recherche d'algorithmes qui solutionnent le problème pour *des graphes arbitraires.* Le temps de ces algorithmes est raisonnable mais exponentiel.

2. La recherche d'algorithmes adaptés à *des classes spéciales de graphes.* Parfois on trouve des méthodes polynomiales pour ces algorithmes.

Comme notre problème de clique n'appartient pas à une catégorie bien définie de graphes, nous nous basons sur des méthodes générales pour écrire l'algorithme. Notre but n'est pas de chercher *toutes les cliques* mais seulement *des cliques maximales*, nous nous limiterons donc aux articles de recherche des cliques (ou d'ensembles indépendants) maximales dans un graphe. Remarquons encore que seul C.Bron et J.Kerbosch cherchent *toutes les cliques maximales* dans un graphe, les autres auteurs se limitent à la recherche *d'une clique maximale.*

Voici maintenant quelques articles intéressants de la littŕature.

5.1.1 R. Carraghan et P.M. Pardalos (1990)

R. Carraghan et P.M. Pardalos proposent *un algorithme de "backtracking"* pour résoudre le problème de la recherche d'une clique maximale, c'est-à-dire de cardinalité la plus grande. Pour ce faire, ils utilisent *une règle heuristique d'ordonnancement des sommets* ainsi qu'une *méthode "branch-and-bound"* qui permet de réduire fortement l'espace de recherche en évitant de descendre le long d'une branche qui ne conduirait pas à une solution. Les noeuds de l'arbre sont des sous-graphes complets, que l'on essaie d'étendre pour former une clique. Les sommets sont ordonnés suivant la règle heuristique suivante. Remarquons qu'un tel ordonnancement réduit le temps de calcul lorsque le graphe est dense. Pour une question de rapidité d'exécution, un graphe peu dense requiert un algorithme sans ordonnancement.

I. La règle heuristique

L'ordonnancement des sommets est fonction de leur degré dans le graphe. Initialement l'algorithme considère un ordonnancement des sommets de G=(X,U), disons $x_1, x_2, ..., x_n$ tel que x_1 est le sommet de plus petit degré dans G, et x_k est le sommet de plus petit degré dans $G\backslash\{x_l, x_2, ..., x_{k-1}\}$.

Avant de décrire la condition de limite ("bound-condition") pour cet algorithme expliquons la notion de profondeur qui est un point crucial pour la compréhension de l'algorithme.

II. La profondeur

On note x_{di} le $i^{ième}$ sommet de l'ordonnancement à la profondeur d. C'est ce point qui va être le $d^{ième}$ point de la clique.

- A la profondeur 1.
  On considère tous les sommets, ordonnancés ou non, soient $x_{11}, x_{12}, ...x_{1n}$. On choisit le sommet x_{1i} comme premier sommet de la clique.

- Pour la profondeur 2.
  On considère tous les sommets adjacents au sommet x_{1i} et on choisit un point x_{2i} pour étendre la clique.

- A la profondeur 3.
  On ne maintient comme nouvel ensemble que les sommets de l'ensemble formé à la profondeur 2 qui sont adjacents au sommet que l'on a choisi d'étendre à la profondeur 2 et qui sont d'ordre supérieur dans l'ordonnancement initial.

On peut généraliser cette règle comme suivant:

- A la profondeur d.
  On ne conserve comme nouvel ensemble pour cette profondeur que les sommets de l'ensemble formé à la profondeur d-1 qui sont adjacents au sommet que l'on a choisi d'étendre à la profondeur d-1 et qui sont d'ordre supérieur dans l'ordonnancement initial. L'ensemble ainsi formé correspond à l'intersection de tous les ensembles des sommets adjacents aux sommets choisis jusqu'à cette profondeur.

A chaque étape d on a donc une suite de sommets $x_{d1}, x_{d2}, ..., x_{dm}$ formée successivement par réduction de l'ensemble des sommets du graphe initial à chaque fois qu'un nouveau sommet est ajouté. Ainsi, seuls les sommets qui sont adjacents à tous les sommets sélectionnés jusqu'à présent sont conservés, ce sont des sommets candidats pour élargir la clique.

Avec cette notion de profondeur, on peut maintenant décrire la condition limite de l'algorithme.

III. La condition limite ("bound condition")

Il suffit de regarder s'il y a encore assez de sommets pour former une clique plus grande que la clique trouvée jusqu'à présent.

A chaque profondeur on connait :

- L'ordonnancement des sommets $x_{d1}, ..., x_{di}, ..., x_{dm}$ où m est le nombre de sommets à la profondeur d et x_{di} est le sommet qu'on essaie d'ajouter à la clique en formation.
- CBC ("current best clique") la taille de la meilleure clique trouvée jusqu'à présent. Au début de l'algorithme CBC vaut normalement 0. Si on sait qu'il existe au moins une clique de taille t alors CBC vaut t-1.
- d+(m-i) est la taille de la clique la plus grande possible que l'on peut former en étendant x_{di}.

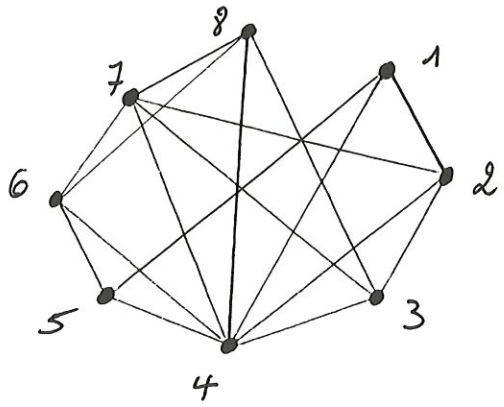
La condition de limite est donc la suivante :
Si d+(m-i) $\leq$ CBC alors il faut élaguer, car la taille de la clique la plus grande possible formée en étendant x_{di} est inférieure ou égale à CBC. Si l'inégalité n'est pas vérifiée, alors l'algorithme progresse en essayant d'étendre la clique le plus possible pour obtenir une clique de taille maximale supérieure à CBC.

Si on est à la profondeur 1 et que la condition limite est vérifiée, alors on s'arrête.

Si on sait que la clique maximale a une taille a, alors on peut utiliser comme critère pour élaguer (stopper), le critère suivant :

$$d + (m - i) < a$$

Figure 5.1: Le graphe G = (X U) avec X ={ 1, 2, 3, 4, 5, 6, 7, 8}.



Dès que l'on a trouvé une clique de taille a, le critère devient

$$d + (m - i) \leq a$$

Remarque : Le but de cet algorithme était la recherche d'**une** clique maximale. Si on veut trouver **toutes** les cliques maximales, on doit modifier la condition d'élagage. La condition de limite est alors

$$d + (m - i) < CBC$$

ou lorsqu'on connait la grandeur de la clique maximale

$$d + (m - i) < a$$

Lorsque $d + (m - i) = CBC$ (respectivement a) on continue pour trouver toutes les cliques de taille maximale.

IV. Exemple

Soit le graphe G = (X, U) de la figure 5.1, avec X = 1, 2, 3, 4, 5, 6, 7, 8. On veut chercher **une clique maximale** de ce graphe. Nous allons écrire le sommet choisi comme candidat pour étendre la clique en gras. Comme on ne connait pas la taille de la clique maximale, on a CBC=0.

Etape 1

L'ordonnancement initial des noeuds selon leur degré est
1 5 2 3 4 6 7 8. L'ordonnancement est tel que x_1 est le sommet de plus petit degré dans G et le sommet x_i est le sommet de plus petit degré dans $G \backslash \{x_1, x_2, ..., x_{i-1}\}$.

Etape 2

Profondeur 1 : **1** 5 2 3 4 6 7 8 i=1 m=8 CBC=0

Profondeur 2 : **5** 2 4 i=1 m=3 CBC=0
Les sommets adjacents à 1 dans
l'ordre de la profondeur 1.

Profondeur 3 : **4** i=1 m=1 CBC=0
Sommet adjacent à 5 se trouvant
après 5 dans la profondeur 2.

On ne sait plus étendre, la meilleure clique actuellement est 1 5 4 de taille 3, donc CBC=3. On doit aller un pas en arrière, c'est-à-dire continuer avec un autre point de la profondeur 2.

Profondeur 2 : 5 **2** 4 i=2 m=3 CBC=3

On a atteint la condition d'arrêt d+(m-i)=2+(2-1)=3≤3, on élague et on remonte d'une profondeur. On a maintenant fini avec le noeud 1.

Etape 3

Profondeur 1 : 1 **5** 2 3 4 6 7 8 i=2 m=8 CBC=3

Profondeur 2 : **4** 6 i=1 m=2 CBC=3

On a d+(m-i)=2+(2-1)=3≤3, on élague. Avec le noeud 5 on ne peut donc pas trouver une clique plus grande que celle que l'on avait avec le noeud 1.

Etape 4

Profondeur 1 : 1 5 **2** 3 4 6 7 8 i=3 m=8 CBC=3

Profondeur 2 : **3** 4 7 i=1 m=3 CBC=3

Profondeur 3 : **4** 7 i=1 m=2 CBC=3

Profondeur 4 : **7** i=1 m=1 CBC=3

On ne sait plus étendre la clique qui est de taille 4, donc plus grande que CBC, maintenant CBC=4 et la meilleure clique trouvée est 2 3 4 7.

Profondeur 3 : 4 **7** i=2 m=2 CBC=3

d+(m-i)=3+(2-2)=3≤4, on élague.

Profondeur 2 : 3 **4** 7 i=2 m=3 CBC=3

d+(m-i)=2+(3-2)=3≤4, on élague.

Etape 5

Profondeur 1 : 1 5 2 **3** 4 6 7 8 i=4 m=8 CBC=3

Profondeur 2 : **4** 7 8 i=1 m=3 CBC=3

d+(m-i)=2+(3-1)=4≤4, on élague.

Etape 6

Profondeur 1 : 1 5 2 3 **4** 6 7 8 m=5 m=8 CBC=4

1+(8-5)=4≤4,

on élague.

Comme on est à la profondeur 1 et que la condition limite est vérifiée on s'arrête.

La clique maximale est 2 3 4 7.

5.1.2 C. Bron et J. Kerbosch

De nombreux ouvrages, articles et applications s'appuient sur l'algorithme de C. Bron et J. Kerbosch, en y apportant parfois des modifications appropriées à des problèmes particuliers.

Les auteurs s'intéressent à la recherche de **toutes** les cliques[1] d'un graphe. Ils présentent *deux algorithmes de "backtracking"* utilisant *une technique de "branch-and-bound"* pour supprimer les branches qui ne conduisent pas à une clique. La première version (l'algorithme de base) génère les cliques dans l'ordre alphabétique (lexicographique). La deuxième version est dérivée de la première et génère les cliques dans un ordre imprévisible afin de minimiser le nombre de branches qui doivent être traversées. Cette dernière version tend à produire en premier lieu les cliques les plus grandes, puis de générer, de manière séquentielle, les cliques qui ont une grande intersection commune avec les cliques les plus grandes.

Nous allons expliquer le principe du premier algorithme et après décrire les modifications pour le deuxième.

I. Description de la première version

Ensembles.

Dans l'algorithme, les trois ensembles suivants jouent un rôle important :

1. L'ensemble COMPSUM.
   C'est l'ensemble correspondant aux noeuds dans l'arbre de "backtracking" qui s'accorde à des sous-graphes complets qu'on essaie d'agrandir autant que possible dans le but

[1] pour eux une clique engendre un sous-graphe complet maximal.

d'obtenir une clique. On ajoute un point à cet ensemble si on **avance** dans l'arbre et on en retire un si on y **recule**. Les points qui peuvent être choisis pour étendre COMPSUM, c'est-à-dire ceux qui sont connectés à tous les points se trouvant dans COMPSUM, sont collectés de manière récursive dans les deux autres ensembles.

2. L'ensemble CANDIDATES.
   Cet ensemble contient tous les sommets qui peuvent être ajoutés à COMPSUM et qui n'ont pas encore servi pour étendre COMPSUB.

3. L'ensemble NOT.
   C'est l'ensemble de tous les points qui ont, à une étape précédente, servi comme extension à la configuration actuelle de COMPSUB et qui sont maintenant explicitement exclus.

Le coeur de l'algorithme consiste en *un opérateur d'extension défini de manière récursive* qui sera appliqué aux trois ensembles décrits. L'algorithme doit générer toutes les extensions de la configuration de COMPSUM possibles avec les données de CANDIDATES et qui ne contiennent aucun point de NOT, car toutes les extensions de COMPSUB qui contiendraient un point de l'ensemble NOT ont déjà été générées.

Mécanisme de base de l'algorithme.
Ce mécanisme comporte les cinq étapes suivantes :

1. Sélection d'un candidat.

2. Addition du candidat sélectionné à COMPSUB.

3. Création des nouveaux ensembles CANDIDATES et NOT en gardant les anciens ensembles intacts (pour savoir les utiliser lorsqu'on fait retour en arrière). Pour rester cohérent avec la définition, on doit créer les nouveaux ensembles en supprimant des ensembles de départ tous les points non connectés au point sélectionné (en autre le candidat lui-même).

4. Appel de l'opérateur d'extension pour intervenir sur les ensembles qui viennent d'être formés.

5. Lors du retour on doit restraire le candidat sélectionné de COMPSUB et l'ajouter à NOT.

Condition d'arrêt.

Une condition nécessaire, mais non suffisante pour la création d'une clique est d'avoir l'ensemble CANDIDATES vide, sans quoi COMPSUB pourrait encore être étendu. Cette condition n'est pas suffisante, car si l'ensemble NOT n'a pas encore été vidé, alors on peut affirmer par sa définition que la configuration présente de COMPSUB est déjà contenue dans une autre configuration et n'est donc pas maximale.

La condition d'arrêt est donc la suivante : **COMPSUB est une clique dès que les deux ensembles NOT et CANDIDATES sont vides.**

Condition limite ("bound condition").

Si à une étape donnée NOT contient un point connecté à tous les autres points de l'ensemble candidates, alors on peut prévoir qu'il ne sera jamais retiré des configurations ultérieures de NOT lors des extensions futures et donc cette branche ne conduit jamais à une clique. La méthode "branch-and-bound" permet de détecter rapidement ces branches pour ne pas progresser dans des directions qui ne conduisent à rien.

La condition limite peut être formulée comme suit : **Il existe un point de NOT connecté à tous les points de CANDIDATES.**

L'implémentation.

Pour l'implémentation, l'ensemble COMPSUB se comporte comme une pile il est maintenu et mis à jour sous la forme d'un tableau global. Les ensembles CANDIDATES et NOT sont passés à l'opérateur d'extension en tant que paramètres. L'opérateur déclare alors un tableau local dans lequel les nouveaux ensembles sont construits, ils seront ensuite passés comme paramètres intérieurs. Les deux ensembles sont stockés dans un seul tableau sous la forme suivante :

1 ne ce ...
$\leftarrow$ NOT $\rightarrow$ $\leftarrow$ CANDIDATES $\rightarrow$

qui jouit des propriétes suivantes:

1. ne $\leq$ ce
2. ne = ce : l'ensemble CANDIDATES est vide, il n'y a donc plus d'extensions possibles.
3. ne = 0 : l'ensemble NOT est vide
4. ce = 0 : on a atteint la condition d'arrêt et trouvé une clique.

Pour la première version de l'algorithme le candidat sélectionné est à la position ne+1. L'ajout du candidat à NOT (lors de l'étape 5) est implanté avec ne=ne+1.

Cette stratégie ne donne jamais lieu à un avancement interne équivoque et toutes les cliques sont donc générées dans un ordre lexicographique selon l'ordonnancement initial des candidats lors de l'appel externe (tous les points).

II. Description de la deuxième méthode

Il s'agit du même algorithme que le premier, la seule chose qui change est que le candidat choisi ne se trouve pas nécessairement à la position ne+1, mais à une position s donnée. On intervertit simplement les éléments des positions s et n+1 après la détermination de s et le reste de l'algorithme reste inchangé.

Comment choisir l'élément s ?

On doit choisir s de façon à minimiser le nombre de répétitions des étapes 1 à 5 à l'intérieur d'un opérateur d'extension. Les répétitions s'arrêtent soit lorsqu'on a trouvé une clique (donc par la condition d'arrêt) ou lorsque la condition de limite ("bound condition") est atteinte. Cette condition limite peut être formulée comme suit :

il existe un point de NOT connecté à tous les points de CANDIDATES.

Pour minimiser les répétitions, on doit essayer que l'existence d'un tel point se montre le plus vite possible. Pour ce faire on associe à chaque point de NOT un compteur du nombre de points dans CANDIDATES auquel ce point n'est pas connecté. Déplacer un candidat sélectionné dans l'ensemble NOT revient à diminuer de 1 tous les compteurs des points dans NOT auquel n'est pas connecté ce candidat sélectionné et à introduire pour lui un nouveau compteur. Les compteurs sont donc diminué de 1 à la fois et lorsqu'un compteur atteint zéro la condition limite est atteinte.

Pour atteindre la condition limite le plus vite possible, on doit donc choisir le point s dans l'ensemble CANDIDATES qui produit la plus petite valeur d'un des compteurs après son addition à NOT. Ce point s est soit un point non connecté au point x de NOT ayant le plus petit compteur (l'addition de ce point s à NOT diminue de 1 la valeur du compteur de x), soit il est un point de CANDIDATES qui est lié à beaucoup d'autres points de CANDIDATES et tel que le nouveau compteur de ce point va avoir la plus petite valeur de tous les compteurs.

En résumé, on peut donc dire que l'on choisit le point s de CANDIDATES non connecté à un point x de CANDIDATES $\cup$ NOT, qui est le sommet ayant le plus de sommets adjacents dans CANDIDATES. Si x $\in$ CANDIDATES, alors s=x.

En annexe A vous trouverez le pseudo-code informatique des deux algorithmes repris de [6].

5.1.3 R.E. Tarjan et A.E. Trojanowski

R.E.Tarjan et A.E.Trojanowski présentent un algorithme qui trouve un ensemble de sommets indépendants maximal[2] dans un graphe G = (X,U) en un temps de $O(2^{\frac{n}{3}})$, où n=#X. Cet algorithmme est donc beaucoup plus vite qu'un algorithe naïf, qui requiert un temps d'exécution de $O(p(n)2^n)$, car il génère tous les sous-ensembles de X (il y en a 2^n) et teste sa propriété de clique (en un temps polynomial p(n)).

L'algorithme utilise un schéma récursif de rebroussement et dépend d'une analyse de cas compliqués. Le point de départ de l'algorithme est l'observation suivante :
Soit $x \in X$ et $A(x)$ l'ensemble des points adjacents à X, alors tout ensemble de sommets indépendants maximal, soit contient x (et alors il ne contient aucun point de $A(x)$), soit ne contient pas x.

Donc tout ensemble de sommets indépendants maximal, soit contient x combiné à un ensemble indépendant dans $G_{X\setminus\{x\}\setminus A}$, soit un ensemble indépendant maximal dans $G_{X\setminus\{x\}}$.

Si S $\subseteq$ X avec $A(S) = \cup_{x\in S} A(x)$, alors tout ensemble maximal I dans G consiste en un ensemble indépendant I $\subseteq$ S dans G_S et un ensemble indépendant maximal $I\setminus S$ dans $G_{X\setminus S\setminus A(I)}$

L'idée de l'algorithme est donc de sélectionner un sous-ensemble S $\subseteq$ X, trouver chaque ensemble indépendant $J \subseteq G_S$ et pour chaque J, trouver de manière récursive un ensemble indépendant maximal dans $G_{X\setminus S\setminus A(J)}$. Cette méthode est améliorée en introduisant le concept de dominance, qui réduit le nombre d'opérations à faire.

La Dominance

Si $S \subseteq X$ et I,J sont indépendants dans G_S, on dit que *I domine J* si $\forall J' \subset X\setminus S$ tel que $J \subseteq J'$ est indépendant, il y a un ensemble $I' \subseteq X\setminus S$ tel que $I \subseteq I'$ est indépendant et $|I \cup I'| \geq |J \cup J'|$.

Pour tout ensemble dominé J on ne doit pas solutionner un sous-problème puisqu'on a un ensemble indépendant au moins aussi grand en solutionnant un sous-problème pour I.

En résumé, l'algorithme sélectionne un ensemble $S \subseteq X$, il détermine un ensemble indépendant, dominant dans S et résout de manière récursive un sous-problème pour chaque ensemble dominant.

[2] ce qui est équivalent au problème de recherche d'une clique de taille maximale dans un graphe donné.

5.1.4 L. BABEL

Comme nous l'avons vu précédemment, il existe deux méthodes pour résoudre le problème de recherche des cliques maximales. Soit on considère des graphes arbitraires (les algorithmes nécessitent d'un temps exponentiel) ou bien des graphes appartenant à des classes spéciales (ici, on trouve parfois des méthodes polynomiales).

L. Babel combine les deux directions. Il a développé *un algorithme "branch-and-bound"* qui tourne rapidement sur des graphes arbitraires et qui, de plus, possède un temps d'exécution polynomial pour quelques classes spéciales de graphes pour lesquelles il n'y avait pas encore de méthodes polynomiales. Cet algorithme repose principalement sur les idées suivantes.

1. L'ordonnancement des sommets.
2. La partition du problème en sous-problèmes eux-mêmes partitionnés en sous-problèmes, ...
   Cette partition forme un arbre de recherche dont les noeuds correspondent aux sous-problèmes.
3. La détermination de composantes connexes.
4. Le calcul de limites inférieures et supérieures pour la taille de la clique maximale de chaque composante.

Schéma général de branchement et arbre de recherche exhaustive

Nous développons une solution générale pour le problème de la clique maximale. Elle peut être spécifiée pour obtenir une méthode plus efficace.

Définition
Soit le graphe G=(X,U) et $(x_1, x_2, ..., x_n)$ un ordonnancement des sommets. Nous définissons $X_i = A(x_i) \cap \{x_{i+1}, ..., x_n\}$, où $A(x_i)$ est l'ensemble des sommets adjacents à x_i.

Lemme 1
Si Cl est l'ensemble des sommets d'une clique dans G, alors $Cl \subseteq \{x_i\} \cup X_i$ pour au moins un $i \in \{1, 2, ..., n\}$.

Sur base de ce lemme, le problème de clique maximale dans un graphe de cardinalité n, peut être transformé en n problèmes dans les graphes $G(X_i)$ de cardinalité d'au plus $n - i$, $i = 1, ..., n$.

Le problème $P(U_t, X_t)$ correspond au problème $MC(G_{X_t})$: *trouver une clique maximale dans* G_{X_t}, *où* $X_t \subseteq X$, $U_t \cap X_t = \emptyset$, $X_t \subseteq \cap_{u \in U_t} A(u)$ *et* G_{U_t} *est un graphe complet, donc* U_t *est une clique en extension, et* X_t *est l'ensemble des sommets adjacents à tous les sommets de* U_t.

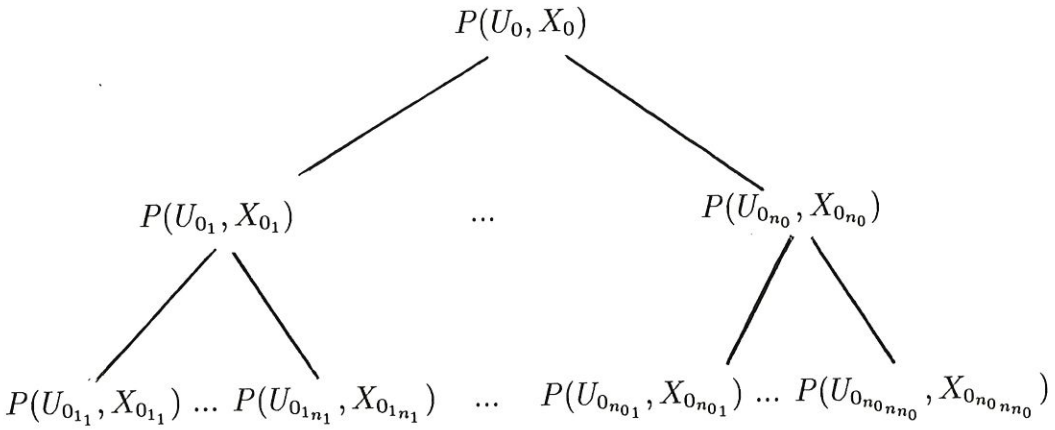
Avec cette définition, on a que $P(U_0, X_0)$, avec $U_0 = \emptyset$ et $X_0 = X$, correspond au problème original $MC(G)$.

Les sommets de U_t (respectivement X_t) sont *des sommets fixes* (respectivement *libres*). La structure de ces ensembles résulte du schéma général de branchement.

Schéma général de branchement

Soient $P(U_t, X_t)$ et $x_{t_1}, ..., x_{t_{n_t}}$ un ordonnancement des sommets de X_t. Pour $i = 1, ..., n_t$ faire : $\begin{cases} U_{t_i} := U_t \cup \{x_i^t\} \\ X_{t_i} := A(x_i^t) \cap \{x_{i+1}^t, ..., x_{n_t}^t\} \end{cases}$

Avec cette règle, un arbre de recherche exhaustive peut être construite, pour lequel $P(U_0, X_0)$ est la racine et les sous-problèmes correspondent aux noeuds de l'arbre.



Estimation de la taille de l'arbre de recherche

Lemme 2

Le nombre de noeuds dans un arbre de recherche exhaustive avec une distance maximale k par rapport à la racine est restreint par

$$b(n,k) = \sum_{i=0}^{k} \begin{pmatrix} n \\ i \end{pmatrix}$$

Preuve du lemme 2

Si le graphe examiné est complet, l'arbre a un nombre maximal de noeuds. Soit l'ensemble

des sommets U_t en $P(U_t, X_t)$ de distance i à la racine (i=$\#U_t$). Tous les ensembles U_t correspondants à des sous-problèmes de distance i à la racine sont distincts. Pour tous les ensembles de cardinalité i, il existe un $P(U_t, X_t)$ avec $U_t = U$. Or il y a $\begin{pmatrix} n \\ i \end{pmatrix}$ combinaisons possibles des n sommets pour former un ensemble de cardinalité i et donc le nombre total des noeuds de cette distance vaut $\begin{pmatrix} n \\ i \end{pmatrix}$.

□

Grâce à ce lemme, on peut dire que pour construire l'arbre de recherche exhaustive entier, l'effort serait de

$$O(\sum_{i=1}^{n} \begin{pmatrix} n \\ i \end{pmatrix}) = O(2^n)$$

Pour exclure l'investigation plus avancée de beaucoup de sous-problèmes, Babel [1] introduit dans sa méthode de recherche de cliques maximales, le calcul de limites sur la taille de la plus grande clique. Cette taille est appelée le nombre clique ("clique number"). Un nouvel algorithme plus performant peut être établi en se basant sur la sélection des noeuds.

L'algorithme *clique* est basé sur l'idée de la décomposition du problème initial en sous-problèmes.

5.2 Algorithme "efficace" pour résoudre le problème de clique

Nous allons présenter le travail du développement d'un algorithme "efficace" de recherche de clique. Au chapitre 6, nous présenterons divers tests concernant l'efficacité de cet algorithme appelé *clique*.

L'algorithme a été inventé afin de résoudre le problème rencontré dans une méthode d'alignement de séquences de protéines : *La recherche de toutes les appariements complètes ("complete match") de segments de protéines.*

Au département de biologie quantitative, il existait un programme de Judy Hempel pour ce faire.

L'idée de départ pour l'écriture du nouvel algorithme était:

1. Soit de se baser sur le programme de Judy Hempel pour obtenir un algorithme qui trouve le plus grand nombre de cliques maximales en un minimum de temps.

2. Soit de trouver un algorithme efficace dans la littérature et éventuellement l'adapter à notre problème.

5.2.1 Programme de Judy Hempel

Ce programme a été écrit dans le cadre d'une application du problème de clique liée à l'analyse conformationnelle des protéines. Il génère **des cliques de toutes les tailles**, mais malheureusement il ne les génère **pas toutes**, car il ne parcourt pas toute la matrice. En effet, pour éviter de générer plusieurs fois une même clique, l'algorithme ne considère plus certains sommets comme début d'une clique. Ceci permet d'accélérer les choses lorsque le nombre de cliques à trouver est important, mais un nombre considérable de cliques ne sont pas prises en compte.

Améliorations possibles

Une première idée d'amélioration est de faire tourner ce programme plusieurs fois sur la même matrice, dont les colonnes et les lignes sont permutées aléatoirement. En faisant cela, le chemin de visite de la matrice est différente et cela génère de nouvelles cliques ainsi que des cliques déjà trouvées. En refaisant tourner plusieurs fois l'algorithme, on arrive vers un plateau de saturation et on peut supposer qu'on a trouvé presque toutes les cliques.

Les auteurs de l'algorithme "efficace" ont laissé tomber cette idée, parce qu'il leur paraissait que même si l'algorithme de J. Hempel est plus rapide qu'un algorithme qui trouve directement toutes les cliques[3], le fait de laisser tourner plusieurs fois le même programme serait une perte de temps, car on perd beaucoup de temps à retrouver d'anciennes cliques pour essayer d'en trouver de nouvelles et on n'est jamais sûr d'avoir trouvé toutes les cliques.

Une deuxième idée est d'améliorer directement le code de l'algorithme pour l'optimiser et l'adapter à notre problème, c'est-à-dire la recherche de toutes les cliques d'une taille maximale donnée (le nombre de séquences moins 1). Cette deuxième idée a été abandonnée pour les raisons suivantes :

- Le programme de Judy Hempel n'était pas optimisé. Il est donc impératif que l'algorithme soit efficace et valablement mis en oeuvre vu le nombre considérable de graphes à traiter lors d'un alignement de séquences de protéines.
- Le programme de J. Hempel était très difficile à "débroussailler" (code inutile, code en commentaire) et il n'était pas possible de contacter l'auteur. En plus, trop de changements et de vérifications étaient à faire.

[3]L'algorithme de J. Hempel est vraiment d'une rapidité exceptionnelle.

Finalement, le nouveau programme a donc été écrit sur base des idées importantes de la littérature.

5.2.2 Particularités de notre problème

On a déjà dit (Page 60) que nos matrices n'appartiennent pas à une classe bien précise de matrices et pour cette raison on doit se baser sur des méthodes générales. D'un autre côte notre problème jouit d'une certaine particularité qui ne se trouve pas dans les articles de la littérature et qui permet de simplifier le problème et d'augmenter la rapidité d'exécution de l'algorithme. Par la définiton d'un groupe complet ("complete match") nous connaissons la taille des cliques maximales se trouvant dans le graphe. En effet, cette taille est égale au nombre de séquences à aligner moins 1, car la séquence de laquelle est issue la fenêtre initiale, n'est pas prise en considération pour former le graphe.

Nous sommes donc confrontés à un problème particulier du problème de cliques que nous pourrions qualifier de **recherche de cliques maximales en population groupée**.

5.2.3 Idées importantes de la littérature

Au chapitre 4 et 5, on a repris différentes idées pour la résolution du problème de cliques. Voici maintenant un aperçu des idées les plus importantes :

- La recherche de toutes les cliques maximales dans un graphe est un problème NP-complet. Aucun algorithme asymptotiquement efficace n'a encore jamais été trouvé pour résoudre ce genre de problème (4.3.3 et 4.3.4).
- Plusieurs approches peuvent être envisagées face à un problème NP-complet (4.3.4).
  - Ou bien, s'il n'est pas traité dans toute sa généralité, un cas spécial du problème est traité. C'est le cas lorsque le problème de clique est résolu pour des classes spéciales de graphes.
  - Ou bien, s'il est traité en entier, des variantes intelligentes de recherche exhaustive, comme *la programmation dynamique ou la technique du "branch-and-bound"* sont utilisées.
  - Parfois aussi, une analyse montre que les occurrences qui poseraient problème, comme par exemple celles pour lesquelles la taille des entrées serait élevée, sont rares. Dans le cas du problème des cliques, il faut analyser si la taille, ainsi que la densité des matrices que nous aurons à traiter sont souvent très grandes.
  - Parfois encore, il n'est pas nécessaire de donner *une solution exacte au problème.* Dans le cas du problème de clique, nous pouvons nous contenter par exemple de rechercher le plus grand nombre de cliques et pas toutes.

- En fin de compte, nous avons vu que *des heuristiques* pouvaient être utilisées. Par exemple l'ordonnancement des sommets selon leur degré [R. Carraghan et P.M. Pardalos (5.1.1)].

- Le concept de clique dans un graphe G=(X,U) est équivalent au concept d'ensemble de sommets indépendant maximal dans le graphe complémentaire.

- La technique de la recherche avec rebroussement ("backtracking") combiné à une méthode "branch-and-bound" est généralement rencontrée pour résoudre le problème de clique (ou le problème des ensembles de sommets maximaux) sous toutes leurs formes [recherche de toutes les cliques (ensembles de sommets indépendants), d'une clique maximale (ensemble de sommets indépendants),...] [R. Carraghan et P.M. Pardalos (5.1.1), C. Bron et J. Kerbosch (5.1.2)]

- Les conditions limites généralement rencontrées dans les méthodes "branch-and-bound" développées par différents auteurs sont:

  - Il ne reste plus de sommet adjacents non traité à tous les sommets de la solution partielle [C. Bron et J. Kerbosch (5.1.2)].
  - Il ne reste plus assez de sommets adjacent à tous les sommets de la solution partielle pour obtenir une clique de taille maximale ou encore, il ne reste plus suffisamment de sommets non adjacents pour obtenir un ensemble de sommets indépendants [R. Carraghan et P.M. Pardalos (5.1.1)].

- Le problème de recherche de toutes les cliques maximales d'un graphe peut être partitionné en sous-probèmes eux-mêmes partitionnés en d'autres sous-problèmes, etc. [L. Babel (5.1.4)].

- L'introduction du concept de dominance améliore la méthode récursive de recherche avec rebroussement [R.E. Tarjan et A.E. Trojanowski (5.1.3)].

5.2.4 Idées reprises de la littérature pour la construction de l'algorithme clique

L'algorithme "efficace" a été écrit principalement sur base des algorithmes de R. Carraghan et P.M. Pardalos (5.1.1), ainsi que de celui de L. Babel (5.1.4). Nous utiliserons qu'un seul ensemble de données. Donnons les caractéristiques principales de l'algorithme.

- L'algorithme est basé sur *une procédure récursive de "backtracking"*. Un arbre de recherche exhaustive est utilisé où chaque noeud correspond à une solution partielle (une clique en extension) qui doit être étendue pour trouver une solution complète (une clique maximale).

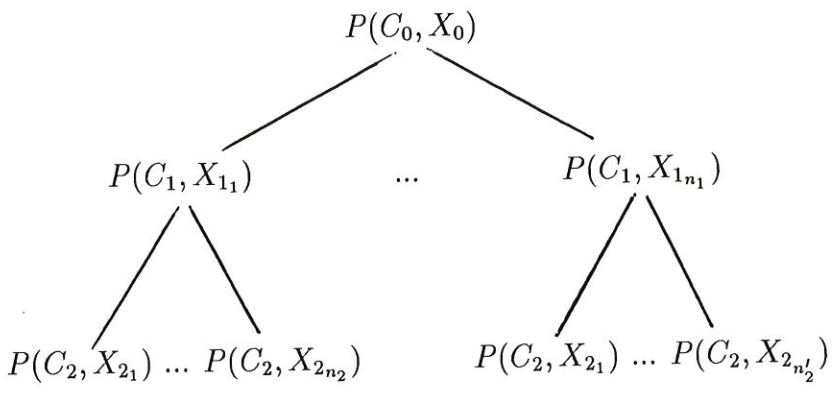
- *Une technique d'élagage* est utilisée, ce qui permet de traiter des problèmes de grande taille. Cette technique consiste à empêcher des symétries, c'est-à-dire à empêcher qu'une même solution soit générée plusieurs fois.

- L'algorithme repose sur l'utilisation d'*une méthode de "branch-and-bound"* qui diffère cependant de toutes celles rencontrées dans les articles de la littérature.

  Introduire la condition limite ("bound condition") est une technique d'élagage, elle permet d'arrêter les recherches dans une branche de l'arbre, dès que l'on sait qu'une solution ne peut pas être atteinte.
  Une particularité de notre problème était qu'une clique doit contenir exactement un "match" issu de chacune des séquences. Nous obtenons dès lors un critère d'arrêt supplémentaire, que voici :
  Dans une des séquences comparées à la fenêtre initiale, nous pouvons arrêter la recherche dès qu'il n'y a plus de "matches" adjacents à tous les "matches" contenus dans la solution partielle.

- Comme Babel, nous pouvons exprimer notre problème en terme de partition en sous-problèmes.

  - Le problème $P(C_i, X_i)$ correspond au problème $MC(G_{X_i})$:
    trouver toutes les cliques maximales du graphe G_{X_i}. C_i correspond à un ensemble de sommets formant un graphe complet, donc à une solution partielle (clique de taille i) destinée à être étendue pour former une clique maximale.
    $X_i = \cap_{c \in C_i} A(c) \subseteq X$ où A(c) est l'ensemble des sommets adjacents au sommet c. Avec cette définition, X_i est l'ensemble des sommets adjacents à tout sommet de la clique en extension.
  - Le problème initial, $P(C_0, X_0)$ correspond au problème original $MC(G_X)$: *trouver toutes les cliques maximales de G=(X,U).* Donc $X_0 = X$ et $C_0 = \emptyset$.
  - L'arbre de recherche a comme racine $P(C_0, X_0)$ et peut être représenté comme suit :



- La profondeur de l'arbre de recherche est égale au nombre de groupes de "matches" qui est aussi la taille d'une clique maximale.
- Les sous-problèmes sont créés de la manière suivante : Soit le problème $P(C_i, X_i)$ à décomposer. A la profondeur i+1, on crée pour chaque sommet x_{i_j} du groupe de "matches" i l'ensemble de sommets $X_{i_j} := A(x_{i_j}) \cap x_{l_i+1} ... x_{l_s}$ où l_i correspond à l'indice du dernier "match" du groupe i et l_s au dernier match du dernier groupe. Le sommet x_{i_j} sera alors le $i + 1^{ième}$ sommet de la clique en extension. $C_{i_j} = C_i \cup x_{i_j}$

Cette stratégie permet de résoudre le problème de manière récursive en solutionnant des problèmes de plus petite taille.

Comme nous utilisons une méthode groupée et que nous connaissons la taille d'une clique maximale, nous avons moins de sous-problèmes à traiter à chaque niveau de l'arbre.

En effet, contrairement à la stratégie de L.Babel, nous développons à l'étape i seulement les sommets du $i^{ième}$ groupe. De cette manière nous évitons le problème de symétrie qui justifiait l'utilisation de deux ensembles de sommets.

De plus, comme nous connaissons la taille d'une clique maximale, nous n'avons pas besoin d'heuristiques pour l'estimer et nous connaissons également la profondeur de l'arbre de recherche et le nombre de noeuds maximaux qu'il aura.

En effet, à chaque niveau, le nombre maximal de sous-problèmes sera égal au nombre de "matches" trouvés dans le groupe correspondant à ce niveau (n_i), donc le nombre total de noeud sera au maximum égal à $n_1 * ... * n_i * ... * n_s$, où s est le nombre de groupes de "matches".

5.2.5 Résolution abstraite de l'algorithme efficace

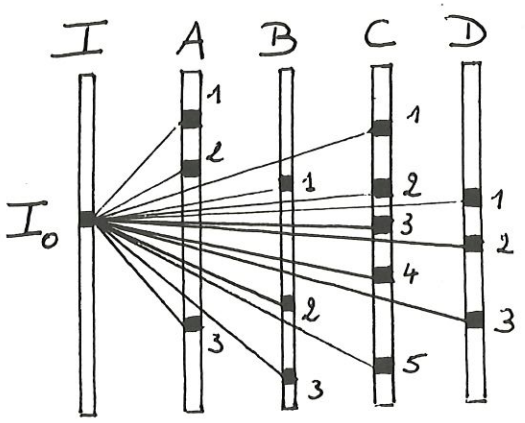
Pour mieux comprendre les grandes lignes du nouvel algorithme, nous allons l'exécuter de manière abstraite sur un petit exemple[4] :

EXEMPLE :

Soient I,A,B,C,D les séquences en AA de cinq protéines que nous voulons aligner. La procédure de "matching" de la méthode d'alignement a identifié pour la fenêtre initiale I_0 se trouvant dans la séquence I les "matches" A_1, A_2 et A_3 dans la séquence A, B_1, B_2 et B_3 dans B, C_1, C_2, C_3, C_4 et C_5 dans C et D_1, D_2 et D_3 dans D (Figure 5.2).

[4]repris du mémoire de Florance Badoux [2]

Figure 5.2: Exemple pour l'algorithme "efficace".



Nous cherchons les groupes complets, c'est-à-dire un ensemble formé d'un segment de chaque séquence de protéines.

La représentation matricielle de ce problème est celle de la table 5.1 :

Table 5.1: Matrice d'adjacente

	A_1	A_2	A_3	B_1	B_2	B_3	C_1	C_2	C_3	C_4	C_5	D_1	D_2	D_3
A_1	1	0	0	1	1	0	1	0	0	1	0	0	1	0
A_2	0	1	0	1	0	0	1	1	1	0	0	1	0	1
A_3	0	0	1	1	0	1	1	0	1	0	1	0	0	0
B_1	1	1	1	1	0	0	0	1	0	1	1	1	0	1
B_2	1	0	0	0	1	0	1	0	0	1	1	1	0	1
B_3	0	0	1	0	0	1	0	1	1	0	0	1	1	1
C_1	1	1	1	0	1	0	1	0	0	0	0	0	1	0
C_2	0	1	0	1	0	1	0	1	0	0	0	1	1	1
C_3	0	1	1	0	0	1	0	0	1	0	0	0	1	0
C_4	1	0	0	1	1	0	0	0	0	1	0	0	0	0
C_5	0	0	1	1	1	0	0	0	0	0	1	0	1	0
D_1	0	1	0	1	1	1	0	1	0	0	0	1	0	0
D_2	1	0	0	0	0	1	1	1	1	0	1	0	1	0
D_3	0	1	0	1	1	1	0	1	0	0	0	0	0	1

Pour la conception de l'algorithme, reprenons la notion de profondeur développée par R.Carraghan et P.M.Pardalos.

De la même façon que ces auteurs, nous travaillons avec des ensembles de sommets candidats, qui sont restreints successivement, pour former une clique. Lorsque nous sélectionnons un candidat, nous supprimons les sommets qui ne sont pas adjacents au sommet étendu. De façon pratique, nous mettons une valeur zéro dans un vecteur (le

vecteur CANDIDAT), pour indiquer que ces sommets ne peuvent plus être utilisés pour étendre la clique. Une valeur 1 indique, bien entendu, qu'ils peuvent servir pour étendre la clique en formation. Nous appellerons CLIEXT cette clique en extention.

Notre algorithme procède comme suit :

1. Initialisation :

A_1	A_2	A_3	B_1	B_2	B_3	C_1	C_2	C_3	C_4	C_5	D_1	D_2	D_3
1	1	1	1	1	1	1	1	1	1	1	1	1	1

A l'initialisation tous les sommets du graphe sont considérés comme des candidats possibles pour former une clique maximale. Le vecteur CANDIDAT est dès lors rempli de 1.

Pour tous les "matches" du premier groupe, nous essayons de former une clique maximale.

2. Profondeur 1:

A_1	A_2	A_3	B_1	B_2	B_3	C_1	C_2	C_3	C_4	C_5	D_1	D_2	D_3
1	1	1	1	1	1	1	1	1	1	1	1	1	1

(a) On sélectionne un candidat dans le premier groupe pour étendre une clique.

Remarque : Le sommet en gras est celui qu'on essaye d'étendre. Le tableau suivant montre le groupe dans lequel on doit choisir un candidat adjacent à tous les sommets formant la clique en extension.

	B_1	B_2	B_3	C_1	C_2	C_3	C_4	C_5	D_1	D_2	D_3
	1	1	1	1	1	1	1	1	1	1	1
A_1	1	1	0	1	0	0	1	0	0	1	0
A_1	1	1	0	1	0	0	1	0	0	1	0

(b) On vérifie le fait qu'il existe encore au moins un candidat dans tous les groupes après l'addition du candidat sélectionné. Dans l'affirmative, la clique en formation trouvée à la profondeur précédente est étendue avec ce candidat (A_1 est le début d'une clique).

Comme le tableau ci-dessus l'illustre, il suffit pour cette vérification, de faire une multiplication booléenne d'une partie du vecteur CANDIDAT avec une partie

de la ligne de la matrice correspondante au candidat sélectionné. Et ensuite, il suffit de vérifier s'il y a au moins un "match" adjacent à tous les sommets de la clique en extension (CLIEXT est A_1). Autrement dit, si toutes les entrées du vecteur résultat sont nulles pour un groupe, alors nous pouvons élaguer. En effet, nous ne formerons jamais de clique de taille maximale égale au nombre de groupes. Tandis que si après la multiplication, il existe encore au moins une entrée non nulle dans chaque groupe, nous passons à l'étape suivante.

(c) Retrait de tous les "matches" qui ne sont pas adjacents au "match" étendu à la profondeur précédente, dans le but de former l'ensemble des candidats pour la profondeur suivante, et essai de manière récursive d'étendre la clique.

Remarque : cela consiste à prendre, comme nouvel ensemble candidat, le vecteur résultant de la multiplication.

3. Profondeur 2 :

	B_1	B_2	B_3	C_1	C_2	C_3	C_4	C_5	D_1	D_2	D_3
A_1	**1**	1	0	1	0	0	1	0	0	1	0

(a) On sélectionne le premier "match" du groupe des B adjacents au match sélectionné à la profondeur précédente, c'est-à-dire à A_1 ($= B_1$).

	C_1	C_2	C_3	C_4	C_5	D_1	D_2	D_3
A_1	1	0	0	1	0	0	1	0
B_1	0	1	0	1	1	1	0	1
A_1 B_1	0	0	0	1	0	0	0	0

(b) On vérifie qu'il existe encore au moins un candidat dans tous les groupes après l'addition du candidat sélectionné.

Ici, le groupe des D ne contient aucune entrée non nulle, donc nous élagons.

Nous retournons à l'étape (a) et nous sélectionnons le "match" suivant dans le groupe de B pour lequel la valeur de candidat vaut 1 ($=B_2$).

	C_1	C_2	C_3	C_4	C_5	D_1	D_2	D_3
A_1	1	0	0	1	0	0	1	0
B_2	1	0	0	1	1	1	0	1
A_1 B_2	1	0	0	1	0	0	0	0

Toutes les entrées dans le groupe D son nulles. Donc, nous élaguons et nous retournons à l'étape (a). Nous sélectionnons alors le match suivant, s'il en existe,

dans le groupe B pour lequel la valeur candidat vaut 1. Comme il n'y en a plus, nous élagons et nous retournons à la profondeur 1.

4. Profondeur 1 :

A_1	$\mathbf{A_2}$	A_3	B_1	B_2	B_3	C_1	C_2	C_3	C_4	C_5	D_1	D_2	D_3
1	**1**	1	1	1	1	1	1	1	1	1	1	1	1

(a) On sélectionne le prochain candidat (A_2) dans le premier groupe.

	B_1	B_2	B_3	C_1	C_2	C_3	C_4	C_5	D_1	D_2	D_3
	1	1	1	1	1	1	1	1	1	1	1
$\mathbf{A_2}$	1	0	0	1	1	1	0	0	1	0	1
A_2	1	0	0	1	1	1	0	0	1	0	1

(b) Nous vérifions qu'il reste, pour ce candidat sélectionné, au moins un "match" adjacent dans les autres groupes.

(c) Pour former l'ensemble des candidats de la profondeur suivante, et pour essayer de manière récursive d'étendre la clique en extension obtenue à la profondeur précédente, on retire tous les "matches" qui ne sont pas adjacents à tous les sommets de celle-ci.

5. Profondeur 2 :

	$\mathbf{B_1}$	B_2	B_3	C_1	C_2	C_3	C_4	C_5	D_1	D_2	D_3
A_2	**1**	0	0	1	1	1	0	0	1	0	1

(a) On sélectionne le premier "match" du groupe des B adjacent à A_2 ($=B_1$).

(b) Nous vérifions qu'il reste, pour ce candidat sélectionné, au moins un "match" adjacent dans les autres groupes après l'addition du candidat sélectionné.

	C_1	C_2	C_3	C_4	C_5	D_1	D_2	D_3
A_2	1	1	1	0	0	1	0	1
$\mathbf{B_1}$	0	1	0	1	1	1	0	1
$A_2\ B_1$	0	1	0	0	0	1	0	1

Remarque : Tous les groupes possèdent des candidats. A_2B_1 forme une clique en extension et nous pouvons passer à l'étape (c).

(c) On retire tous les "matches" non adjacents au candidat sélectionné et on passe ensuite à la profondeur suivante.

6. Profondeur 3 :

	C_1	$\mathbf{C_2}$	C_3	C_4	C_5	D_1	D_2	D_3
A_2 B_1	0	**1**	0	0	0	1	0	1

(a) On sélectionne le premier "match" du groupe adjacent à tous les "matches" de la clique en extension, c'est-à-dire à A_2 B_1 $(=C_2)$

(b) On vérifie qu'il existe encore au moins un candidat dans tous les groupes après l'addition du candidat sélectionné.

	D_1	D_2	D_3
A_2 B_1	1	0	1
$\mathbf{C_2}$	1	1	1
A_2 B_1 C_2	1	0	1

Le groupe D possède des candidats. C_2 est ajouté à la clique en extension. Nous pouvons passer à l'étape (c).

(c) Nous retirons tous les "matches" non adjacents et nous passons à la profondeur suivante.

7. Profondeur 4 :

	$\mathbf{D_1}$	D_2	$\mathbf{D_3}$
A_2 B_1 C_2	**1**	0	**1**

Ici, la profondeur est égale au nombre de groupes de "matches". Les cliques maximales correspondent aux cliques formées des sommets sélectionnés aux profondeurs précédentes, et de chaque "match" de cette dernière profondeur adjacent à ceux-ci. Nous avons ainsi formé les cliques maximales A_2 B_1 C_2 D_1 et A_2 B_1 C_2 D_3 et nous retournons à la profondeur précédente.

8. Profondeur 3 :

	C_1	C_2	C_3	C_4	C_5	D_1	D_2	D_3
A_2 B_1	0	1	0	0	0	1	0	1

(a) Il n'y a plus de "match" candidat dans le groupe C, alors nous élagons et retournons à la profondeur 2.

9. Profondeur 2 :

	B_1	B_2	B_3	C_1	C_2	C_3	C_4	C_5	D_1	D_2	D_3
A_2	1	0	0	1	1	1	0	0	1	0	1

(a) Il n'y a plus de "match" candidat dans le groupe B, alors nous élagons et retournons à la profondeur 1.

10. Profondeur 1 :

A_1	A_2	$\mathbf{A_3}$	B_1	B_2	B_3	C_1	C_2	C_3	C_4	C_5	D_1	D_2	D_3
1	1	**1**	1	1	1	1	1	1	1	1	1	1	1

(a) On sélectionne le prochain candidat dans le premier groupe pour étendre une clique (=A_3).

	A_1	A_2	A_3	B_1	B_2	B_3	C_1	C_2	C_3	C_4	C_5	D_1	D_2	D_3
	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$\mathbf{A_3}$	0	0	1	1	0	1	1	0	1	0	1	0	0	0
A_3	0	0	1	1	0	1	1	0	1	0	1	0	0	0

(b) On vérifie qu'il existe pour ce candidat sélectionné au moins un "match" adjacent dans les autres groupes.

Ici, d'une part, le groupe D ne comprend aucune entrée non nulle et d'autre part, nous sommes au dernier sommet du premier groupe de la première profondeur. La recherche de toutes les cliques de taille maximale est donc terminée. Ces tailles correspondent au nombre de groupes.

Nous pouvons déduire aisément notre algorithme de la description donnée ci-dessus. Cet algorithme est décrit en pseudo-code en annexe B.

Chapitre 6

Evaluation et interprétation des tests de l'algorithme clique

6.1 Tests réalisés par Florence Badoux

Florence a testé l'efficacité de son programme sur des matrices générées aléatoirement avec une taille et une densité donnée. Pour ce faire, elle a comparé son programme à celui de Judy Hempel, *Brandnewcli* (5.2.1) et à celui de R. Carraghan et P.M. Pardalos (5.1.1), *Proglit*.

En fait, elle a considéré deux programmes *Proglit1* et *Proglit2*. Pour les deux programmes, l'égalité de la condition limite de l'algorithme a été enlevée pour chercher toutes les cliques maximales et pas seulement une. Le premier, *Proglit1*, ne réalise aucun ordonnancement des sommets et le deuxième, *Proglit2*, le réalise uniquement lorsque la densité de la matrice considérée est ≥ 0.4.

Elle a également implanté deux versions de l'algorithme *clique*, pour juger de l'efficacité du critère d'arrêt (à savoir la présence de sommets candidats à l'extension dans tous les groupes). La première, *clique1*, génère les cliques dans l'ordre lexicographique avec le critère d'arrêt supplémentaire et le deuxième, *clique2*, sans critère d'arrêt.

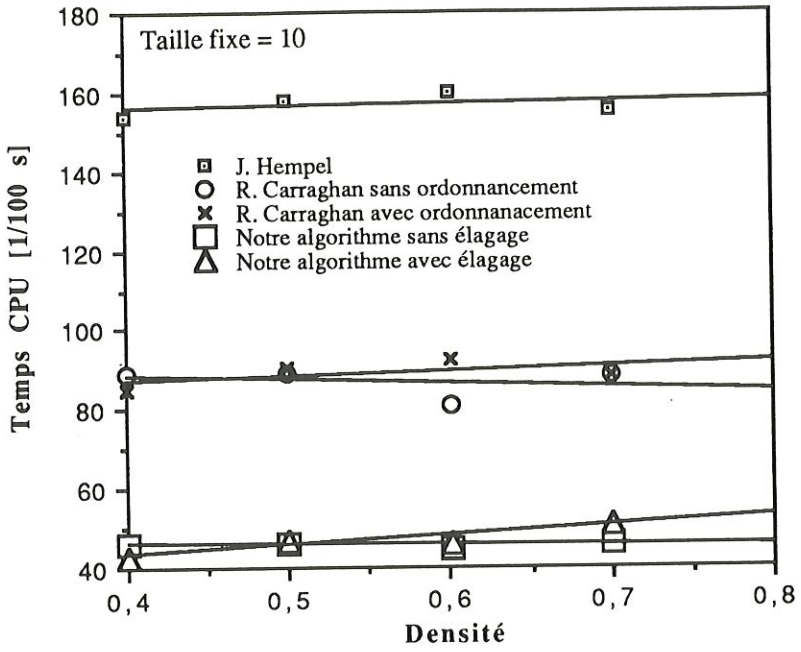
6.1.1 Programmes non comparables

En fait, les programmes ne peuvent pas vraiment être comparés pour les raisons suivantes :

1. Ils n'ont pas été conçus pour les mêmes problèmes.
   Brandnewcli cherche les cliques de toutes les tailles, *Proglit* a été écrit pour chercher une clique maximale et *clique* pour trouver toutes les cliques maximales.

2. Les programmes sont écrits dans des langages différents.
   Clique est écrit en C, alors que les autres programmes sont écrits en FORTRAN.

Figure 6.1: Temps d'exécution en fonction de la densité



3. Il est difficile de faire une analyse rigoureuse et détaillée, car on a beaucoup de données qui influencent les résultats, comme la taille de la matrice, sa densité, le nombre de séquences,...

4. Florence Badoux ne disposait pas de beaucoup de place mémoire et devait se limiter à des matrices de faible taille. Elle n'a donc pas pu tester le problème dans toute sa généralité.

6.1.2 Résultats obtenus

CPU en fonction de la densité

Pour les premiers tests, la taille des matrices est fixée à 10 (respectivement 30) et la densité varie entre 0.4 et 0.8 (respectivement 0.1 et 0.7). Les figures 6.1 et 6.2 montrent clairement que l'algorithme clique est plus efficace.

CPU en fonction de la taille

Pour ces tests (Figure 6.3), la densité des matrices est fixée à 0.7 et la taille varie entre 10 et 50.

On voit que *Brandnewcli* a un comportement exponentiel (ce programme n'est donc pas adapté à des matrices de grande taille), alors que *clique* reste linéaire.

Figure 6.2: Temps d'exécution en fonction de la densité de la matrice

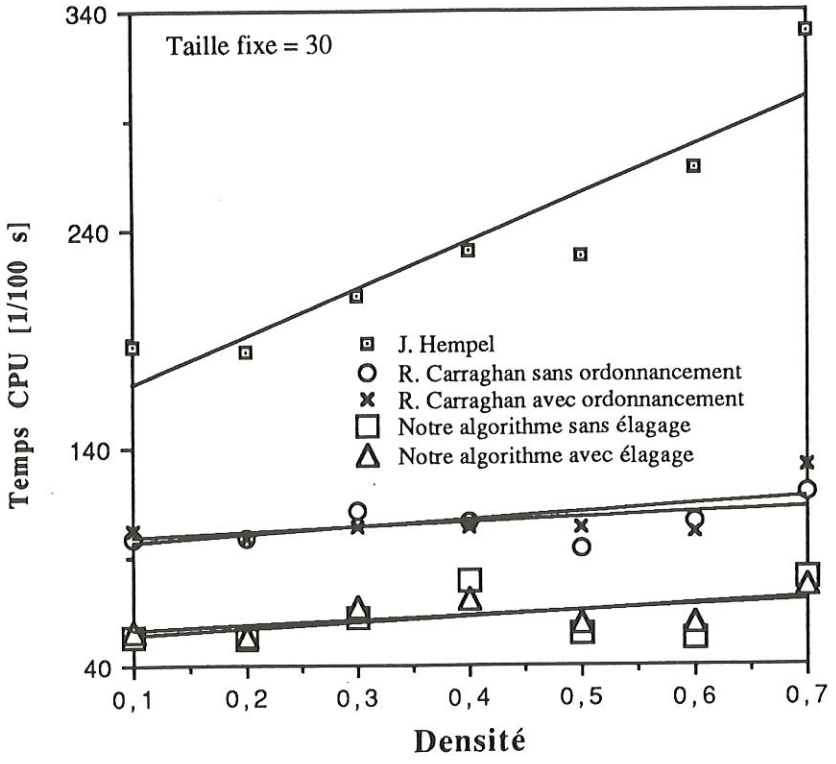
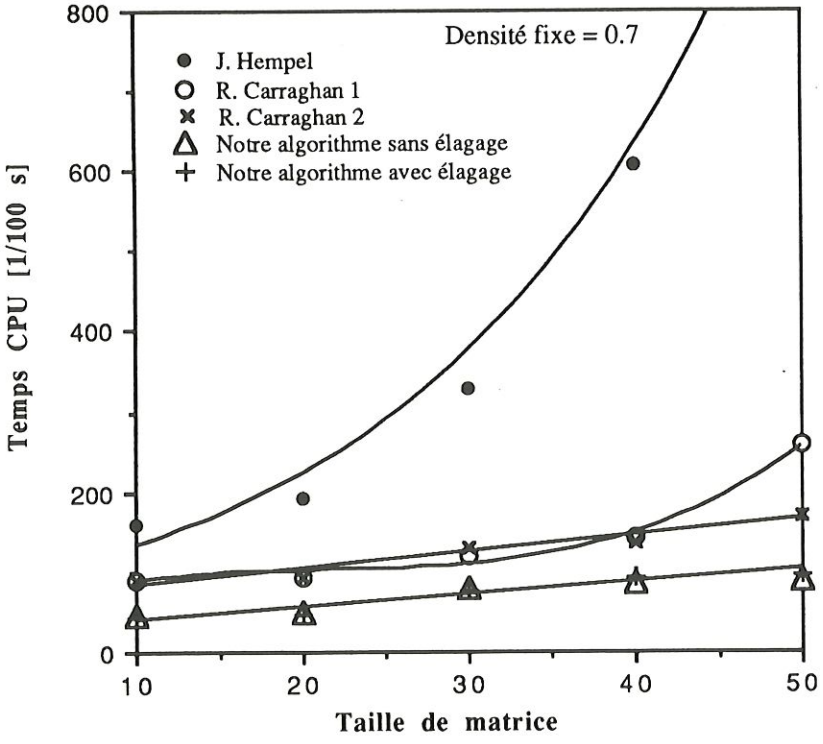


Figure 6.3: Temps d'exécution en fonction de la taille de la matrice



6.2 Evaluation et interprétation de mes tests

Eric Depiereux a implanté le programme *clique* comme sous-routine du programme *Match-Box*, plus précisément dans le programme *Matching*. Ce programme cherche tous les "matches" complets de taille maximale lors de l'alignement des protéines. La sous-routine *clique* est seulement utilisée si on considère plus d'un match par séquence pour une fenêtre initiale (c'est-à-dire si deep > 1). Le nouveau programme ainsi formé est appellé *MATCH-CLI*. Nous avons donc pu tester l'efficacité de l'algorithme à partir de situations réelles, alors que Florence devait se limiter à des simulations. Remarquons encore que nous avons seulement travaillé avec le programme *Match-cli* qui réalise le "matching" et non avec tout le programme *Match-Box*. Le programme était stocké sur VAX.

Nous avons travaillé sur des terminaux liés à deux stations VAX possédant un "operating system" du type VMS 5.4-2, la première station (QUICK) est du type VAX 6220, c'est donc un biprocesseur avec une mémoire centrale de 64 MB et la deuxième (FLUPKE) est du type VAX 3600 avec une mémoire centrale de 16 MB.

Par construction du programme, nous pouvions donner des valeurs différentes à certaines variables qui intervenaient dans le problème en modifiant le fichier d'entrée *param.dat*. Le programme *match-cli* crée un fichier sortie [*match.log* (cfr. annexe C)] reprenant les données initiales (se trouvant dans *param.dat*) et indiquant des résultats obtenus comme par exemple les différents CPU, le nombre de fenêtres initiales, de "matches",... Vous trouvez les résultats des tests en 6.3.1.

Dans mon tableau, nous représentons les valeurs des variables auxquelles nous nous sommes intéressées id, n, m, CPUtotal, deep, ncompmat, CPUsetup, CPUsort, CPUtrmat, CPUclique, nmatch, nfinit. Tous les CPU sont exprimé en centièmes de secondes. Voici maintenant les explications concernant les différentes variables.

- id = le numéro du test.
- n = le nombre de séquences.
  Nous avions à notre disposition 13 séquences de protéines de longueur différentes.

  Il s'agit de séquences d'enzymes de la glycolyse, la glyceraldehyde de 3-phosphate deshydrogenase, pour 13 espèces différentes. Elles ont été choisies parce que l'on sait qu'elles peuvent s'aligner, mais que leur alignement n'est pas trivial. En 6.3.2 vous trouverez l'alignement final des 13 séquences réalisé par le programme match-box[1].

  Voici maintenant le nom des séquences (protéines) et le nombre AA qu'ils contiennent.

[1] Cet alignement m'a été communiqué par E.Depiereux

	nom	nombre AA
1.	*G3P1_CAEEL.SW*	431
2.	*G3P1_ECOLI.SW*	331
3.	*G3P1_ESCVU.SW*	294
4.	*G3P1_HUMAN.SW*	334
5.	*G3P1_YEAST.SW*	331
6.	*G3PA_MAIZE.SW*	403
7.	*G3PC_TOBAC.SW*	326
8.	*G3PG_TRYBB.SW*	358
9.	*G3PG_TRYCR.SW*	359
10.	*G3P_BACST.SW*	335
11.	*G3P_HOMAN.SW*	333
12.	*G3P_THEAQ.SW*	331
13.	*G3P_THEMA.SW*	332

Les 75 premiers tests ont été faits avec les 3 premières séquences de protéines de notre liste.
Les tests 76 à 152 avec les séquences 4, 5, 6, 7, 8 et les tests 153 à 188 avec toutes les 13 séquences.

- m
  m est une variable qui joue un rôle important lors du "scanning" des séquences (2.2.2), plus elle est importante, plus la probabilité de trouver plusieurs "matches" au sein d'une même séquence augmente. Sa valeur se trouve dans la troisième colonne.

- CPUtotal
  Cette variable comporte le temps CPU que prend le programme dans son intégralité.

- deep
  Variable contenant la valeur choisie pour deep. Deep indique combien de "matches" sont pris en considération par fenêtre initiale pour chaque séquence. Pour le programme original (*MATCHING*), la valeur de deep était toujours égale à 1, car on ne considérait que le meilleur "match" par séquence pour chaque fenêtre initiale. Grâce au nouveau algorithme, *clique*, on peut également choisir deep>1. Les "matches" sont classés suivant les critères énoncés pour la sélection du meilleur match (2.2.4). Deep détermine le nombre de matches retenus parmi les meilleurs. Pour l'implémentation la valeur de deep est comprise entre 1 et 10.

- ncompmat
  Variable contenant le nombre de "matches" complets trouvés par le programme lors de l'exécution du programme MATCHING.

- CPUsetup
  Le temps nécessaire pour gérer les inputs et outputs.

- CPUsort
  Le temps pour trier les données. Ce temps est égal à 0 lorsque deep vaut 1, car on garde toujours que le meilleur "match" par séquence pour chaque fenêtre initiale. Dès que l'on trouve une fenêtre courante qui est plus similaire que la meilleure trouvée jusqu'à ce moment, on chasse l'ancienne fenêtre et on prend la meilleure comme nouveau "match".

  Lorsque deep>1, le processus est plus compliqué, on doit gérer un tableau de deep éléments qui peut contenir les *deep* meilleurs "matches", dans l'ordre décroissant de leur degré de similitude. Lorsque l'algorithme trouve un "match", il doit l'insérer dans la liste selon son degré de similitude et éventuellement écarter un élément de la liste (le $deep + 1^{ième}$). La variable CPUsort indique le temps nécessaire pour trier la liste des "matches" de ce tableau.

- CPUclique
  Le temps utilisé par la sous-routine *clique* pour chercher toutes les cliques maximales dans toutes les matrices de vérités.

- CPUtrmat
  Le temps pour établir une matrice de vérité ("truth-matrix") pour chaque fenêtre initiale. Cette étape requiert de tester les critères de similitude pour toutes les paires de fenêtres mobiles appartenant à des séquences différentes.

- nfinit
  Le nombre de fenêtres initiales qui vaut $\sum_{i=1}^{r}(L_i - w + 1)$, il est donc constant pour tous les tests avec les mêmes séquences.

- nmatch
  Le nombre de "matches" trouvés. Ce nombre dépend des séquences choisies et du paramètre m et de la sécurité α (cfr. page 23) qui est maintenu constant pour tous les tests.

Une première idée de ce travail était de voir si l'algorithme détecte vraiment toutes les cliques maximales. Pour ce montrer nous avons fait tourner plusieurs fois le programme *Match-cli* avec les mêmes données en prenant des permutations des séquences. A chaque

fois, nous obtenions exactement le même nombre de "matches" complets, donc de cliques maximales. Le CPU variait toujours de peu, mais pas significativement.

Nous avons fait ce travail pour les tests suivants :

$$\begin{array}{lll} sequences = 1,2,3 & deep = 1 & m = 15 \\ sequences = 1,2,3 & deep = 5 & m = 15 \\ sequences = 10,11,12 & deep = 1 & m = 15 \\ sequences = 4,5,6,7,8 & deep = 10 & m = 200 \end{array}$$

On voit donc que les résultats du matching — aussi bien avec la procédure *clique* que sans — ne dépendent pas de l'ordre d'entrée des séquences, et on peut donc supposer que l'algorithme détecte vraiment toutes les cliques.

Une deuxième idée de ce travail était d'étudier le temps d'exécution du programme en fonction de la dimension du problème, qui dépend des quatres facteurs suivants :

1. Le nombre de séquences choisies, n, ainsi que du nombre de résidus par séquence. L'implémentation du programme permet de choisir jusqu'à 20 séquences comprenant en moyenne 500 résidus.

2. Du paramètre m.

3. Du nombre maximal de "matches" pris en considération, deep.

4. De la valeur du *cutoff* choisie.

Pour nos tests nous avons fait varier les 3 premiers facteurs, mais la valeur du *cutoff* restait inchangée.

Un autre facteurs pouvant influencer le programme est l'élagage. Pour tous mes tests la variable *élagage* valait zéro, donc le programme ne fait pas d'élagage.

Florence Badoux a montré que l'élagage est seulement intéressant si les premières séquences sont moins denses, car sinon on perd plus de temps à mettre l'élagage en route que ce qu'il permet de gagner comme temps. Pour que l'élagage soit intéressant, on devrait réarranger chaque matrice de vérité pour avoir des séquences avec peu de "matches" (donc des séquences moins denses) en premier lieu. Le réarrangement coûterait probablement plus que ce que l'on gagne en faisant l'élagage. Il pourrait cependant être intéressant de tester cela.

On voudrait bien établir une relation entre les variables et le temps utilisé par le programme, donc le CPUtotal, pour avoir une idée du temps nécessaire à l'exécution du programme, avant de lancer le programme. Ceci permettra d'éviter de lancer des programmes qui nécessitent un temps CPU trop grand.

Le CPUtotal est décomposé en différents CPU's correspondant à différentes étapes du programme. On a la relation suivante :

$$\mathbf{CPUtotal = CPUsetup + CPUtrmat + CPUclique + CPUsort}$$

Lorsque deep=1, la valeur de CPUclique et CPUsort vaut 0, pour des raisons expliquées plus haut.

Pour déterminer cette relation, on va essayer de trouver une relation pour chaque CPU séparément. Les figures 6.4, 6.5, 6.6 et 6.7 montrent la répartition des différents CPU pour les différents tests avec deep>1.

$\diamond$ représente $CPUtotal$
$+$ représente $CPUsetup$
$\square$ représente $CPUsort$
$\times$ représente $CPUtrmat$
$\triangle$ représente $CPUclique$

On voit clairement — sauf lorsque m est très petit — que CPUclique est la composante majeure du CPUtotal. CPUsetup est une autre composante importante du CPUtotal, par contre CPUtrmat ainsi que CPUsort sont quasi négligeables (sauf lorsque n=13 CPUtrmat n'est pas négligeabe pour des raisons expliquées à la page 107).

Lorsque deep=1 (Figures 6.8 et 6.9) le CPUtotal ($\diamond$) est déterminé par le CPUsetup ($+$); le CPUtrmat ($\square$) est négligeable.

Nous allons maintenant étudier plus en détail les différents CPU's, mais avant cela nous parlons de la matrice de corrélation et de la régression linéaire.

6.2.1 Matrice de corrélation

Que décrit la corrélation ?

L'analyse de corrélation est utilisée pour décrire le degré de liaison linéaire entre deux variables.

Lorsque deux variables sont corrélées positivement (respectivement négativement), les observations qui ont de fortes valeurs pour une variable ont en général une forte valeur (respectivement une valeur faible) pour l'autre. Par exemple le poids et la taille sont généralement des variables positivement corrélées. Par contre les prix des bons du trésor et le taux d'intérêt sont des variables négativement corrélées, à une chute de l'une correspond une hausse de l'autre.

Lorsque deux valeurs ne sont pas corrélées, il n'y a aucune liaison linéaire entre les valeurs des deux variables.

Figure 6.4: Les différents CPU pour les tests avec n=3 (deep> 1)

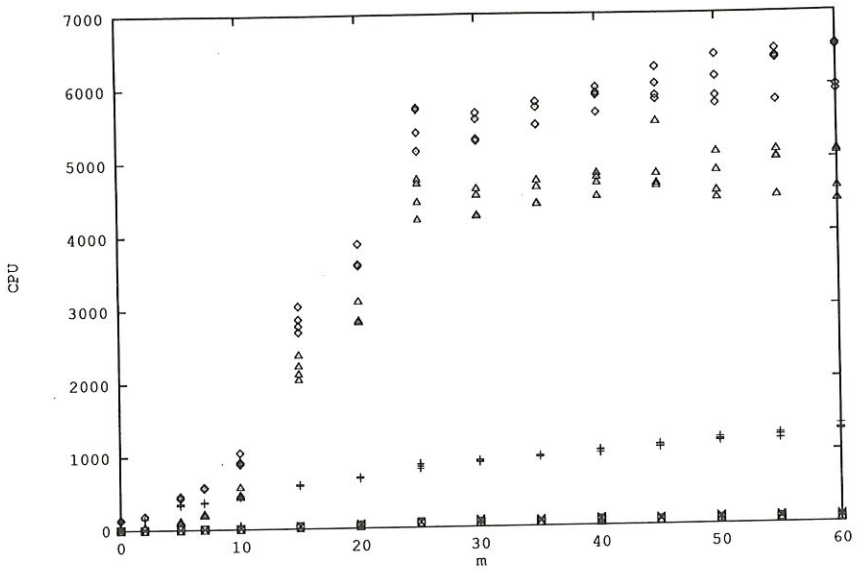


Figure 6.5: Les différents CPU pour les tests avec n=5 (deep> 1)

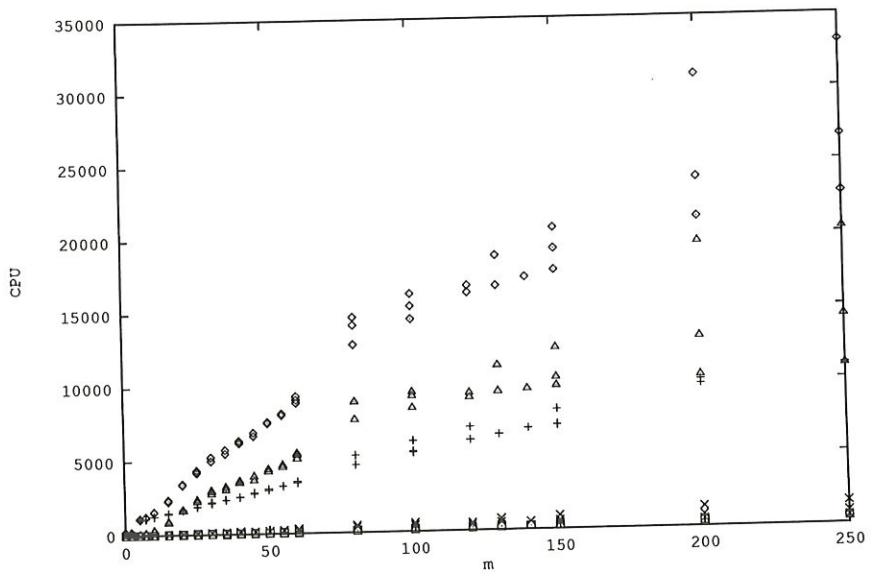


Figure 6.6: Les différents CPU pour les tests avec n=13 (deep> 1)

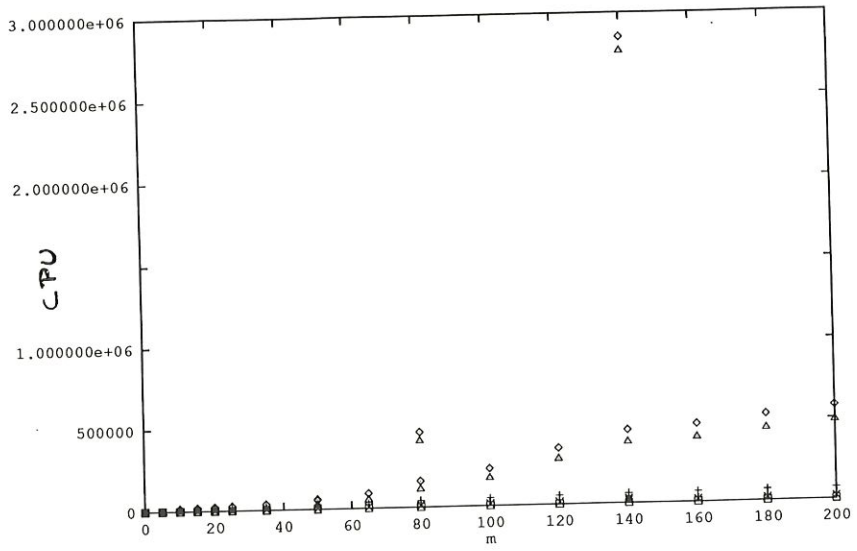


Figure 6.7: Les différents CPU pour les tests avec n=13 (deep> 1)

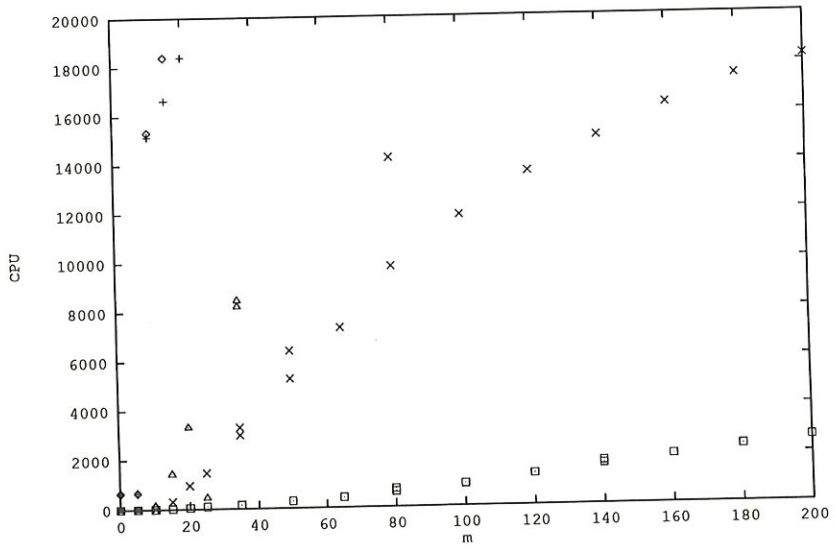


Figure 6.8: Les différents CPU pour tous les tests avec deep = 1

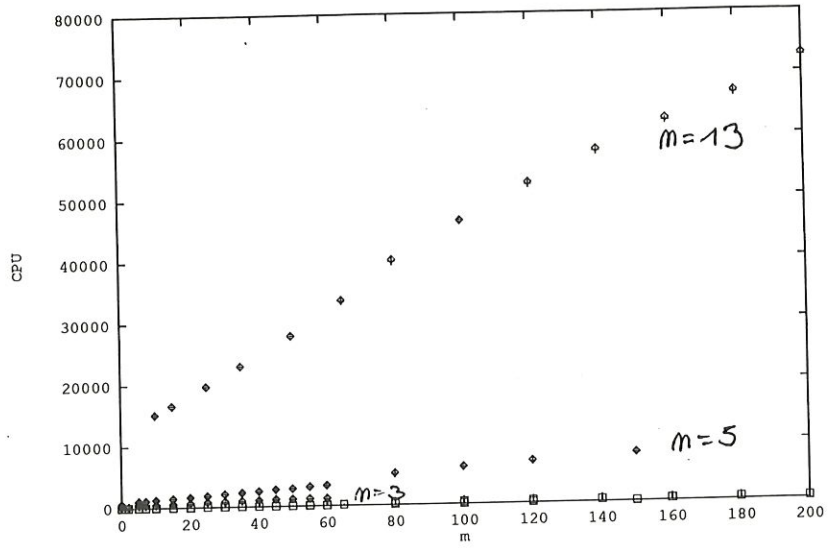


Figure 6.9: Les différents CPU pour tous les tests avec deep = 1

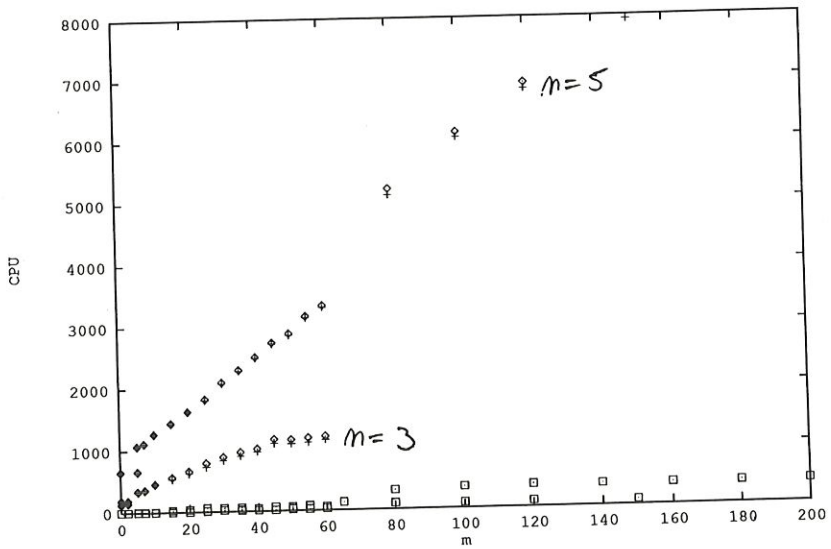


Table 6.1: Matrice de corrélation

	n	m	CPU total	deep	ncomp-mat	CPU setup	CPU sort	CPU trmat	clique	nmatch	nfinit
n	1										
m	0.2963	1									
CPUtotal	0.3305	0.2758	1								
deep	-0.1247	0.0061	0.1410	1							
ncompmat	0.3133	0.2776	0.9946	0.1500	1						
CPUsetup	0.7972	0.5774	0.4901	-0.0827	0.4872	1					
CPUsort	0.4134	0.5425	0.6426	0.1505	0.6994	0.6612	1				
CPUtrmat	0.5120	0.3879	0.8024	0.1632	0.8354	0.7006	0.9149	1			
CPUclique	0.4035	0.4067	0.5990	0.0817	0.6630	0.6535	0.9670	0.8861	1		
nmatch	0.7341	0.6076	0.4955	-0.0817	0.4954	0.9905	0.6740	0.7038	0.6680	1	
nfinit	0.9992	0.3050	0.3274	-0.1281	0.3101	0.7917	0.4118	0.5074	0.3996	0.7293	1

La valeur du coefficient de corrélation se situe entre -1 et +1. Un coefficient de corrélation voisin de +1 (respectivement -1) signifie que les deux variables sont fortement corrélées positivement (respectivement négativement). Un coefficient voisin de zéro signifie que les deux variables sont très faiblement corrélées.

On voit par exemple que la corrélation entre n et nfinit vaut 0.9992, les deux variables sont donc positivement corrélées, ce qui est normal puisque nfinit augmente en fonction de n.

6.2.2 Régression linéaire

Pour faire de la régression linéaire ou multilinéaire (multiple), nous nous sommes servis de BMDP [30][2] ainsi que de SAS (**S**tatistical **A**nalysis **S**ystem) [33].

La régression linéaire est une technique statistique uilisée pour étudier les liaisons entre variables. En général, la régression peut être utilisée pour

- vérifier si une variable peut être exprimée en fonction d'une autre.
- prévoir les valeurs d'une variable en fonction des valeurs données à d'autres variables.

Terminologie de la régression

L'équation d'une droite peut s'écrire

$$y = \alpha + \beta\, x$$

où α est le terme constant et β le coefficient de la droite.

Lorsqu'on suppose que la variable y (variable dépendante) dépend de la variable x (variable indépendante), alors on peut faire la régression pour avoir une estimation de l'équation de la droite (c'est-à-dire une estimation de α et β) qui ajuste au mieux les points correspondants aux variables x et y.

Il existe plusieurs techniques de régression, selon le sens de l'expression "ajuster au mieux".

Nous employons la régression linéaire selon les moindres carrés ordinaires qui consiste à minimiser la somme des carrés des écarts entre y réel et $\hat{y}$ ajusté par la formule

$$\hat{y} = \hat{\alpha} + \hat{\beta}\, x$$

où $\hat{\alpha}$ et $\hat{\beta}$ sont les estimations trouvées pour α et β.

Sur les graphiques j'ai repris les droites[3] de régression trouvées.

Pour étudier la dépendance de y par rapport à plusieurs variables, on fera appel à une régression multiple. On aura un coefficient de régression pour chaque variable indépendante (x_i), l'équation s'écrit alors

$$y = \alpha + \beta_1\, x_1 + ... + \beta_n\, x_n$$

[2]Un autre livre de référence peut être [7]

[3]Pour certaines modèles ce sont des courbes et non des droites à cause des transformations des variables initiales

Le principe des moindres carrés

Pour déterminer la droite de meilleur ajustement, nous évaluons pour chaque point le carré de la distance verticale qui existe entre ce point et la droite et ensuite on en fait la somme. *La droite cherchée est celle qui minimise cette somme de carrés.*

Ce critère nous garantit que les points seront aussi proches que possible de la droite en question.

R-square (coefficient de corrélation carré)

Il mesure la quantité de variation de la variable dépendante qui peut être expliquée par le modèle. Ce coefficient, dont la valeur est comprise entre 0 et 1, est obtenu en divisant la somme des carrés expliquée par le modèle par la somme des carrés totale. R-square × 100 indique le pourcentage des valeurs de y expliqué par la connaissance de x et le modèle linéaire.

En général, plus grande est sa valeur, meilleur est l'ajustement du modèle.

Pour nos tests nous avons parfois créé des variables nouvelles, comme par exemple $\log(n-1)$, et considéré ces variables comme variables dépendantes ou indépendantes. J'indiquerai toujours le modèle utilisée, ainsi que R-square (pour avoir une idée de la cohérence du modèle choisi) et les paramètres de régression α, β_1 , ..., β_n.

6.2.3 Nombre de matches (nmatch)

Le nombre de matches dépend directement de m et des séquences choisies. Comme vous le voyez sur les graphiques 6.10 et 6.11, il existe une relation linéaire entre m et le nombre de matches, qui dépend des séquences choisies.

Cette relation n'est pas parfaite. On voit clairement sur la figure 6.11 que à partir de m=150 les points forment plutôt une courbe et non une droite. En effet, le nombre de matches est limité. Il augmente linéairement avec m, mais à partir d'une certaine limite, nmatch n'augmente plus aussi vite, ce qui est normal, car si m = $\frac{1}{2}L_i$, alors, on compare la fenêtre initiale avec toute la séquence i, donc un m plus grand ne permet pas de trouver plus de matches pour cette séquence. Comme nos séquences ont des valeurs entre 294 et 431, les points ne suivent plus une régression linéaire à partir de m=147 et nmatch reste constant à partir de m=215, car lorsque m=$\frac{1}{2}\max_i L_i$, alors on a atteint la limite pour nmatch.

Donc, si on respecte certaines limites, on peut écrire la relation suivante.

$$nmatch = \alpha + \beta \times m$$

Figure 6.10: Le nombre de matches en fonction de m

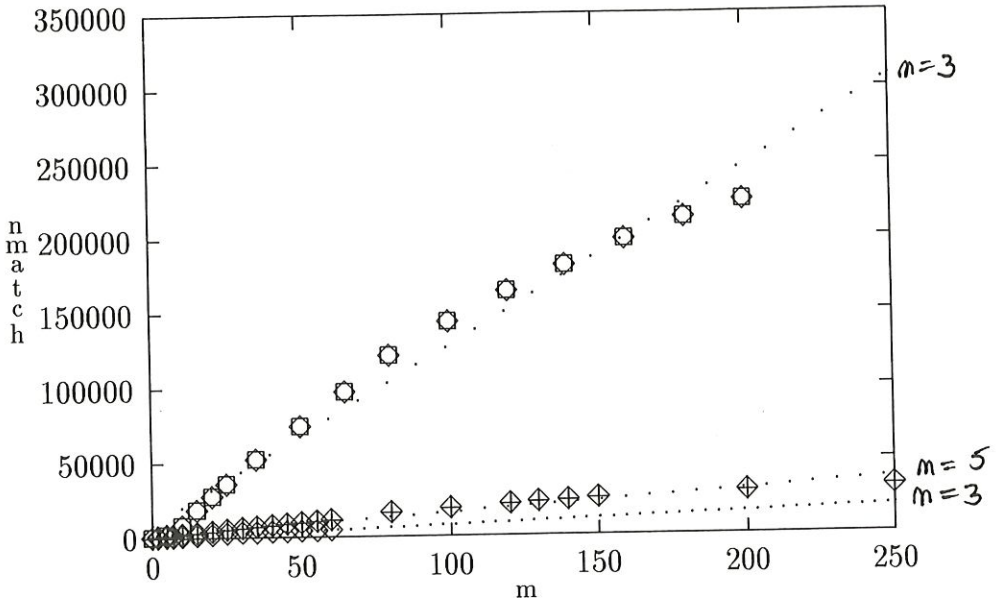
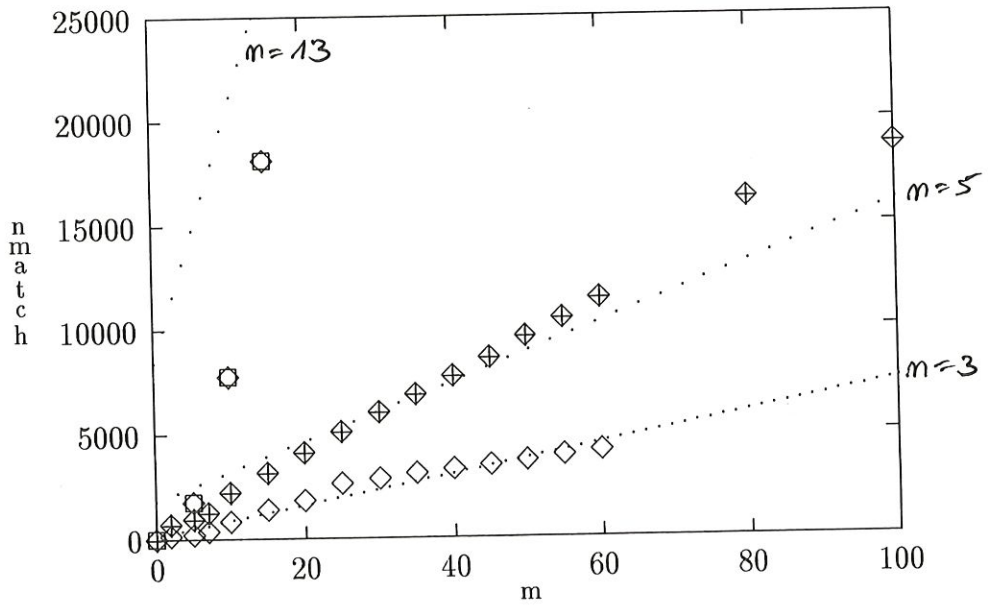


Figure 6.11: Le nombre de matches en fonction de m



Les paramètres α et β dépendent des séquences choisies.

Il suffit donc de déterminer pour deux valeurs de m (ce qui ne prend pas beaucoup de temps si les deux valeurs sont petites) la valeur de nmatch et ensuite on peut estimer la valeur de nmatch pour un m plus grand, car on connait l'équation de la droite $nmatch = \alpha + \beta \times m$. Ensuite, on peut estimer le CPU qu'utilisera le programme en fonction du nmatch.

Pour nos tests, nous avons trouvé les coefficients suivants pour le modèle linéaire :

	α	β	R-square
n=3	177.15686	73.3400	0.9578
n=5	1825.18262	141.7447	0.9612
n=13	8422.17969	1195.3931	0.9779

Comme on exprimera par la suite toutes les graphiques en fonction de nmatch, j'ai construit le tableau suivant, qui permettra d'associer le m correspondant au nombre de matches pour les différents tests.

	n=3	n=5	n=13
nmatch	m	m	m
1000	11	-	-
5000	66	22	-
10000	135	58	-
20000	-	128	10
50000	-	-	35
100000	-	-	77
200000	-	-	160

Il est clair que nmatch dépend de m et de nfinit, mais il n'est pas possible de généraliser l'équation de la relation linéaire, car ce nombre dépend également de n et de la similarité entre les séquences choisies.

6.2.4 Les différents CPU's

CPUsetup

Comme le montrent les figures 6.12 et 6.13, le CPUsetup dépend linéairement de nmatch. On peut écrire la relation suivante :

$$\text{CPUsetup} = \alpha + \beta \text{ nmatch}$$

avec $\alpha = 631.60547$ et $\beta = 0.3232$ et R-square $= 0.9812$.

Figure 6.12: CPU setup en fonction du nombre de matches

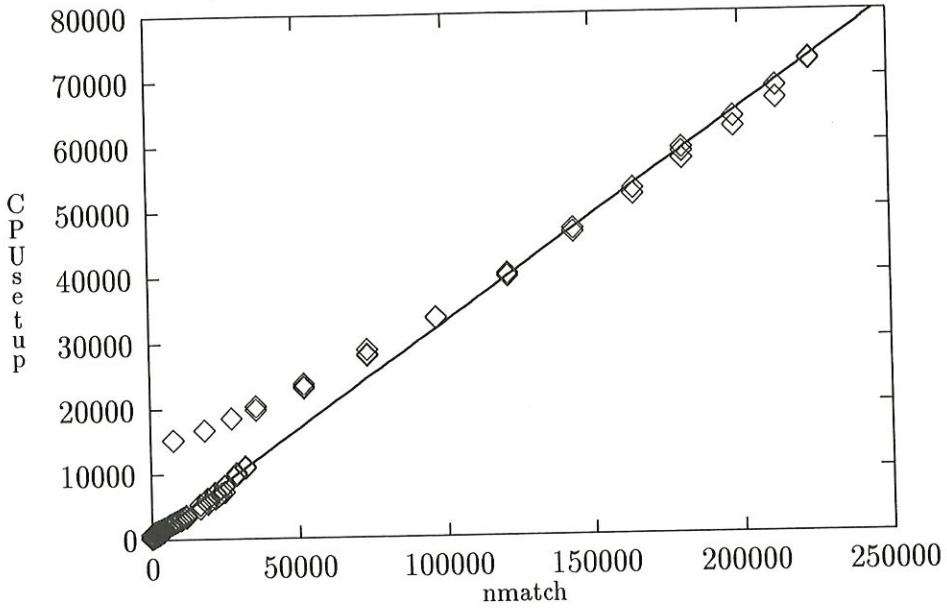
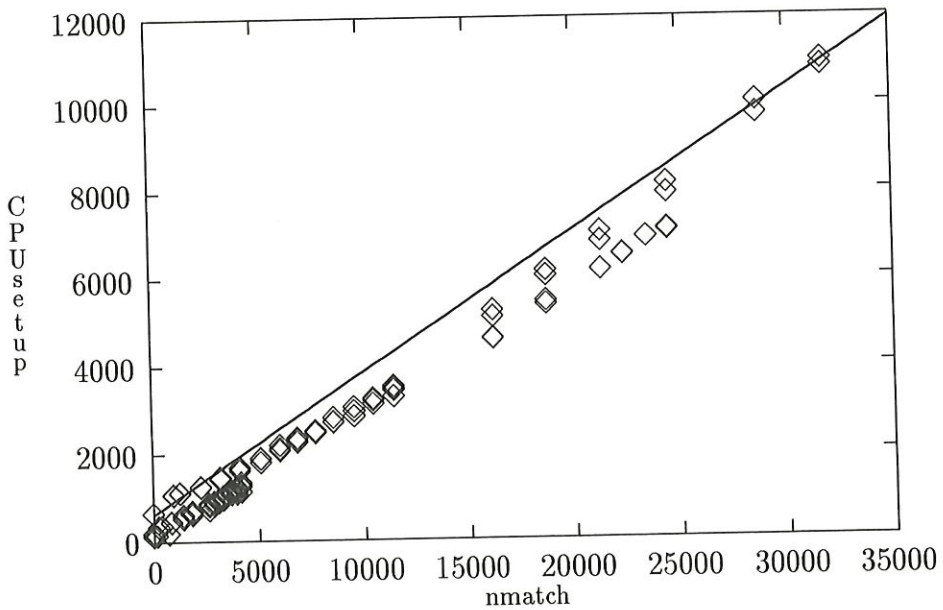


Figure 6.13: CPU setup en fonction du nombre de matches



CPUtrmat

Au début nous avons cru que $CPUtrmat$ était proportionnel au carré de $nmatch$, nous avons donc testé le modèle suivant :

$$CPUtrmat = \alpha + \beta\ nmatch^2$$

et obtenu les résultats suivants :

$$\begin{aligned} \alpha &= 257.7752379 \\ \beta &= 0.0000003 \\ R - square &= 0.455726 \end{aligned}$$

Comme d'une part $R - square$ est insuffisant et qu'en plus β montre clairement que $CPUtrmat$ n'est pas expliqué uniquement en fonction de $nmatch^2$, nous avons abandonnée cette idée.

Comme les résultats sont fort différents suivant que deep>1 ou deep=1, nous avons fait deux analyses différentes.

deep=1

Nous avons supposé que CPUtrmat dépend de nmatch, de la taille ((n-1)×(n-1)) et du nombre ($nfinit$) des matrices de vérité et testé le modèle suivant :

$$\log(CPUtrmat) = \alpha + \beta_1\ \log(nmatch) + \beta_2\ \log(n-1) + \beta_3\ \log(nfinit)$$

Les résultats obtenus étaient très bons :

$$\begin{aligned} \alpha &= 424.9509334 \\ \beta_1 &= 1.7773717 \\ \beta_2 &= 55.5161891 \\ \beta_3 &= -69.1089055 \\ R - square &= 0.944597 \end{aligned}$$

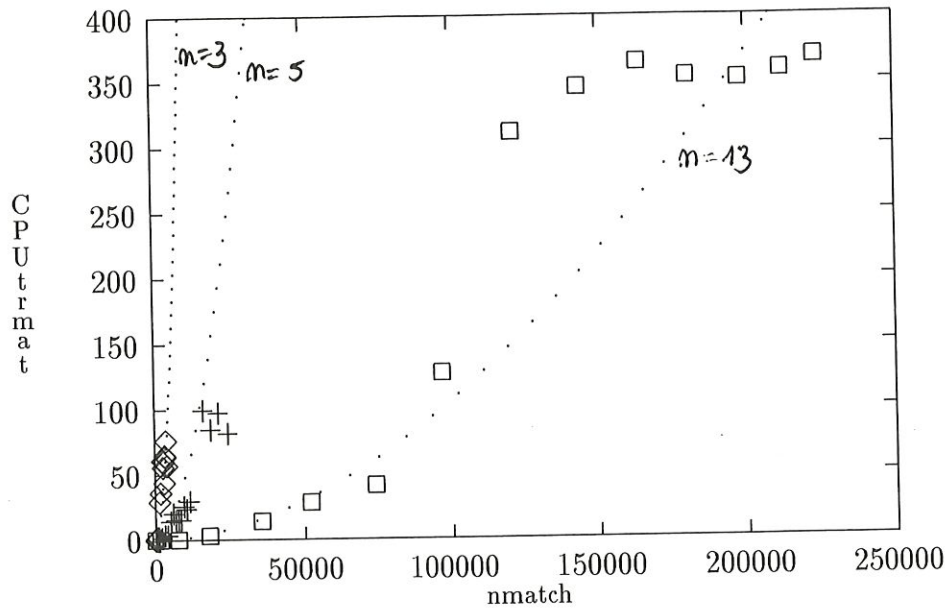
Le modèle s'écrit donc

$$CPUtrmat = \frac{e^{424.95}\ nmatch^{1.78}\ (n-1)^{55.52}}{nfinit^{69.11}} \tag{6.1}$$

On voit qu'on a $\beta_1 \sim 2$, donc l'idée de départ n'était pas si mauvaise. Cependant le coefficient de $nmatch^2$ n'est pas une constante, mais varie en fonction de n et nfinit. Comme il existe une corrélation forte entre n et nfinit, nous avons simplifié le modèle en considérant uniquement n.

$$\log(CPUtrmat) = \alpha + \beta_1\ \log(nmatch) + \beta_2\ \log(n-1)$$

Figure 6.14: CPUtrmat en fonction de nmatch pour deep=1



Les résultats obtenus :

$$\begin{aligned} \alpha &= -8.500034446 \\ \beta_1 &= 1.785722374 \\ \beta_2 &= -3.040337648 \\ R-square &= 0.908786 \end{aligned}$$

Le modèle s'écrit donc

$$CPUtrmat = \frac{nmatch^{1.79}}{e^{8.5}(n-1)^{3.04}} \tag{6.2}$$

Le modèle 6.2 est encore un bon modèle et il est beaucoup plus compréhensible que le modèle 6.1. Les figures 6.14 et 6.15 montrent les courbes de régression ainsi que les points trouvés pour les tests avec deep=1 où

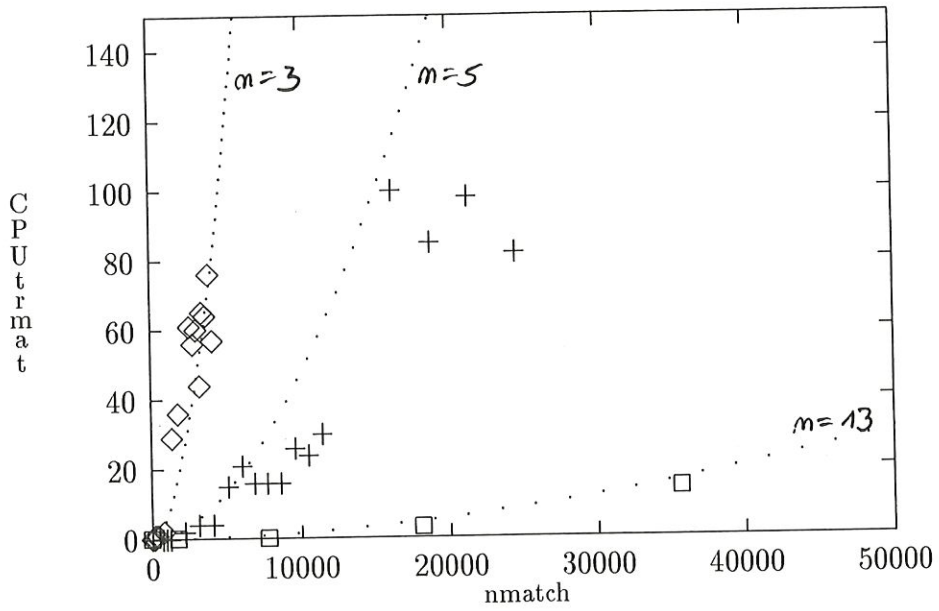
- $\diamond$ représente les points pour $n = 3$
- $+$ représente les points pour $n = 5$
- $\square$ représente les points pour $n = 13$

deep>1

Lorsque deep>1, la taille de la matrice de vérité est

$$((n-1) \cdot deep) \times ((n-1) \cdot deep)$$

Figure 6.15: CPUtrmat en fonction de nmatch pour deep=1



et notre modèle doit donc tenir compte de la variable deep.

$$\log(CPUtrmat) = \alpha + \beta_1 \log(nmatch) + \beta_2 \log(n-1) + \beta_3 \log(deep)$$

Comme les résultats obtenus pour ce modèle étaient très bons, nous n'avons pas continué la recherche et gardé directement ce modèle (Figures 6.16, 6.17 et 6.18).

$$\begin{aligned} \alpha &= -8.752976787 \\ \beta_1 &= 1.603493433 \\ \beta_2 &= -0.782825814 \\ \beta_3 &= 0.335494513 \\ R-square &= 0.950060 \end{aligned}$$

Le modèle s'écrit donc

$$CPUtrmat = \frac{nmatch^{1.6}\,(deep)^{0.34}}{e^{8.75}\,(n-1)^{0.78}} \tag{6.3}$$

CPUsort

deep=1

Lorsque deep=1, CPUsort vaut toujours 0, car on garde toujours que le meilleur "match" par séquence pour chaque fenêtre initiale et par conséquent on ne doit pas trier les "matches".

Figure 6.16: CPUtrmat en fonction de nmatch pour n=3 et deep>1

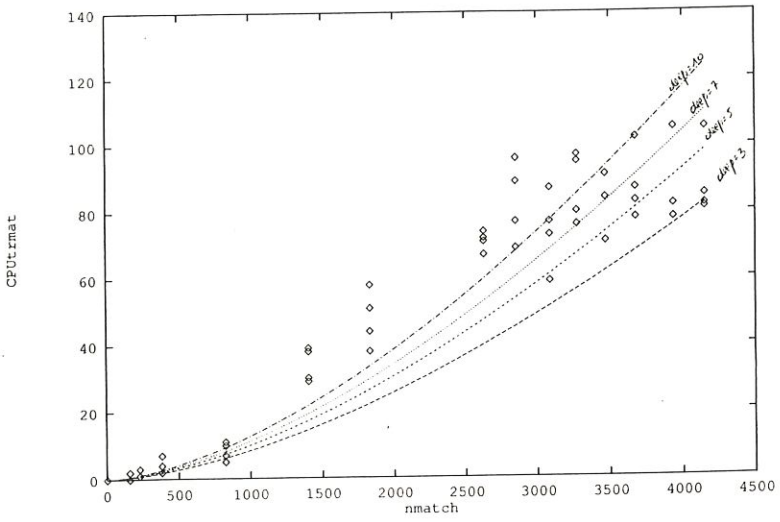


Figure 6.17: CPUtrmat en fonction de nmatch pour n=5 et deep>1

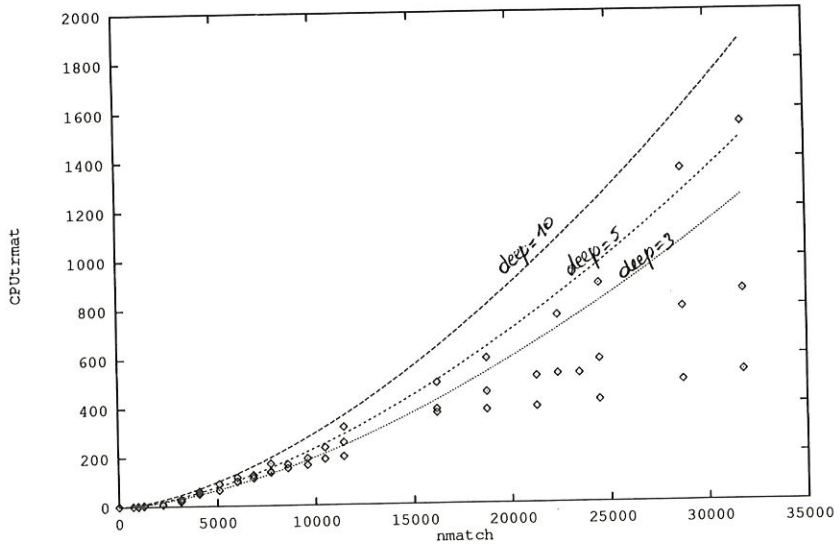
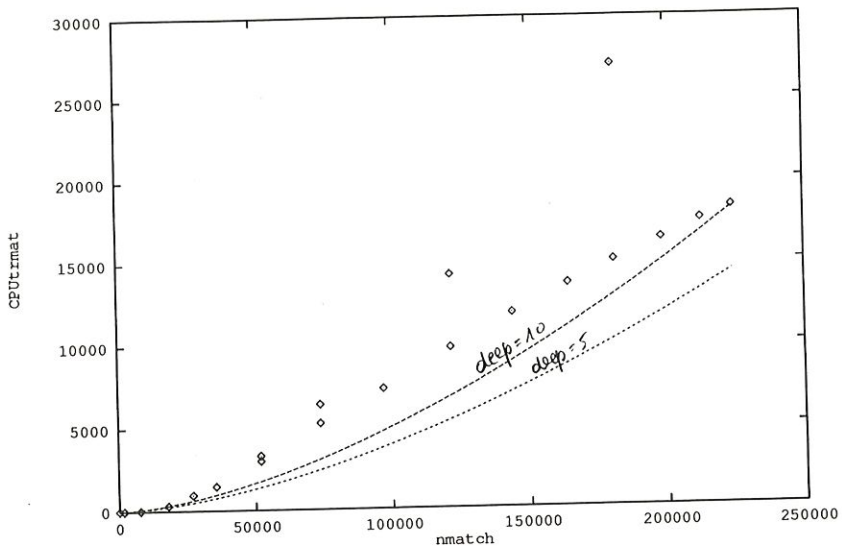


Figure 6.18: CPUtrmat en fonction de nmatch pour n=13 et deep>1



deep > 1

Modèle testé :

$$\log(CPUsort) = \alpha + \beta_1 \log(nmatch) + \beta_2 \log(n-1) + \beta_3 \log(deep)$$

Les résultats obtenus :

$$\begin{aligned} \alpha &= -7.095520248 \\ \beta_1 &= 1.510768415 \\ \beta_2 &= -1.792280959 \\ \beta_3 &= 0.062425747 \\ R-square &= 0.94719 \end{aligned}$$

Le modèle s'écrit donc

$$CPUsort = \frac{nmatch^{1.51}\,(deep)^{0.06}}{e^{7.1}\,(n-1)^{1.8}} \tag{6.4}$$

Comme β_3 est très petit, et que deep a des valeurs comprises entre 1 et 10, on pourra simplifier le modèle comme suit :

$$CPUsort = \frac{nmatch^{1.51}}{e^{7.1}\,(n-1)^{1.8}} \tag{6.5}$$

Les équations 6.4 et 6.5 et les figures 6.19 et 6.20 montrent bien que CPUsort est beaucoup plus petit que le CPUclique, il est négligeable par rapport au CPUtotal, ce que l'on a déjà vu sur les figures 6.4, 6.5, 6.6 et 6.7.

Figure 6.19: CPUsort en fonction de nmatch pour deep>1

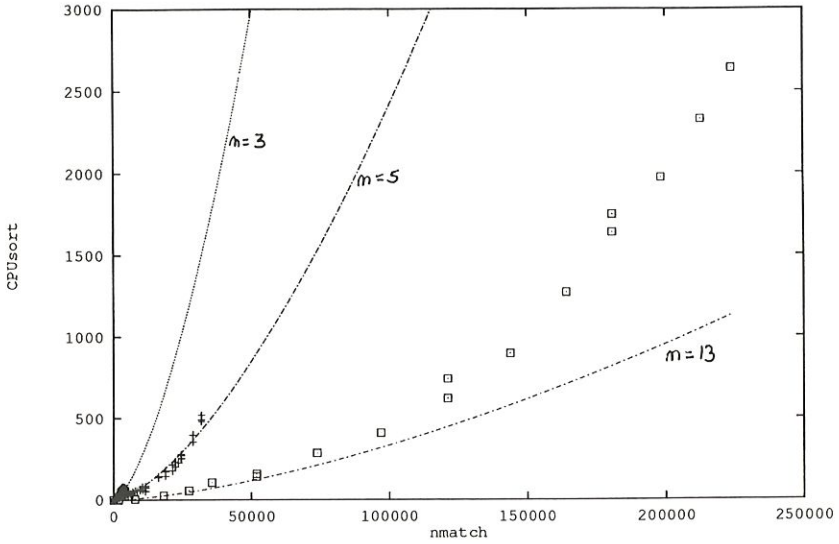
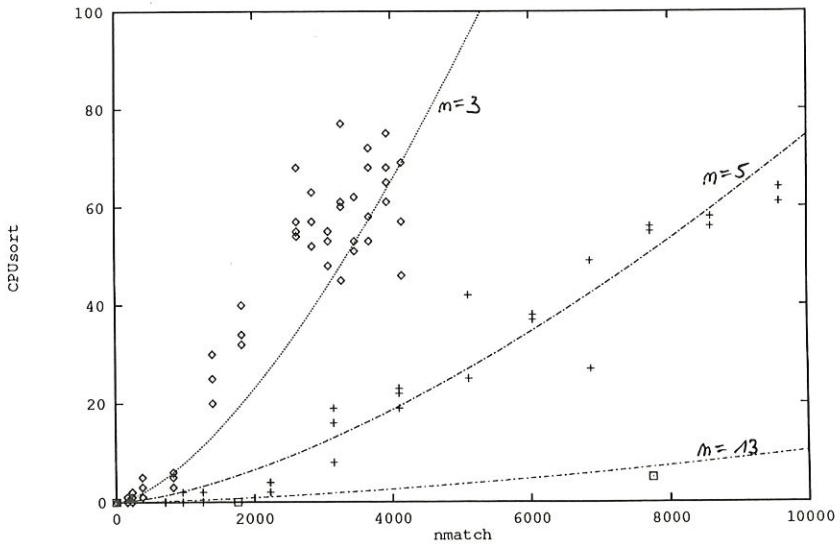


Figure 6.20: CPUsort en fonction de nmatch pour deep>1



CPUclique

deep=1

Lorsque deep=1, CPUclique vaut 0, puisque l'on n'emploie pas l'algorithme de recherche des cliques maximales.

deep > 1

Modèle testé :

$$\log(CPUclique) = \alpha + \beta_1 \log(nmatch) + \beta_2 \log(n-1) + \beta_3 \log(deep)$$

Les résultats obtenus :

$$\begin{aligned} \alpha &= -4.365042695 \\ \beta_1 &= 1.679722445 \\ \beta_2 &= -2.039860824 \\ \beta_3 &= 0.272692107 \\ R-square &= 0.873191 \end{aligned}$$

Le modèle s'écrit donc

$$CPUclique = \frac{nmatch^{1.68}\ (deep)^{0.27}}{e^{4.37}\ (n-1)^{2.04}} \tag{6.6}$$

On voit que l'on a presque les mêmes exposants pour nmatch et deep pour les modèles 6.3 et 6.6 et que l'exposant de (n-1) est d'une unité plus élevée pour CPUclique.
On peut donc écrire la relation suivante :

$$CPUclique \sim \gamma\ \frac{CPUtrmat}{n} \tag{6.7}$$

avec $\gamma = e^{4.38} = 79.8$. Comme γ est très grand, on voit encore une fois que CPUclique est beaucoup plus élevé que CPUtrmat, c'est d'ailleur le terme dominant du CPUtotal.

La relation entre ces deux CPU peut être expliquée par le fait qu'ils expriment toutes les deux le temps d'une partie du programme *MATCH-CLI* qui traite les matrices de vérités. CPUtrmat est le temps pour construire les matrices de vérité et CPUclique est le temps pour y chercher les cliques. Pour réduire le temps CPU du programme, on devrait essayer d'améliorer le temps CPUclique, comme c'est la composante majeure de CPUtotal.

Par la relation 6.7 nous voyons que CPUtrmat est négligeable par rapport à CPUclique et CPUtotal, sauf dans le cas ou n est grand. En effet sur le graphique 6.7 on voit clairement que CPUtrmat est beaucoup plus élevé que sur les figures 6.4 et 6.5, ceci est du au fait que n=13 pour les figures 6.7 et 6.6 et vaut 3 et 5 pour les deux autres.

Figure 6.21: CPUclique pour n=3 et deep>1

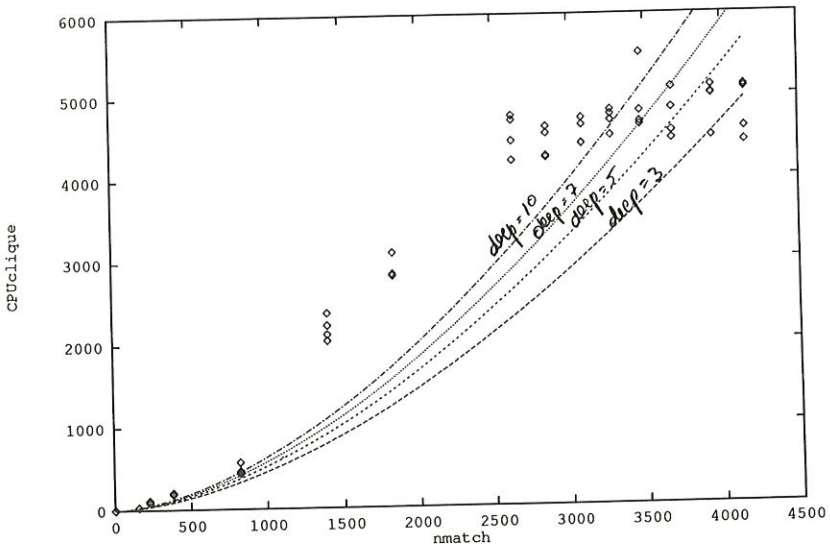


Figure 6.22: CPUclique pour n=5 et deep>1

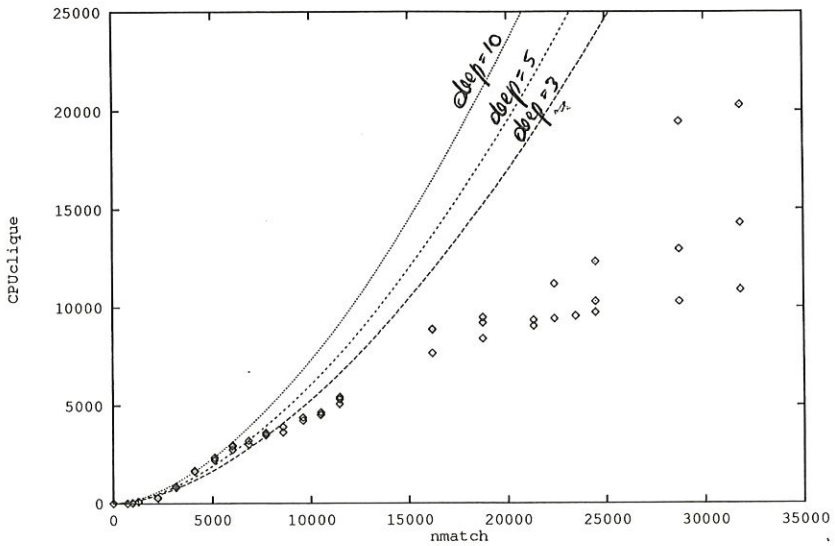
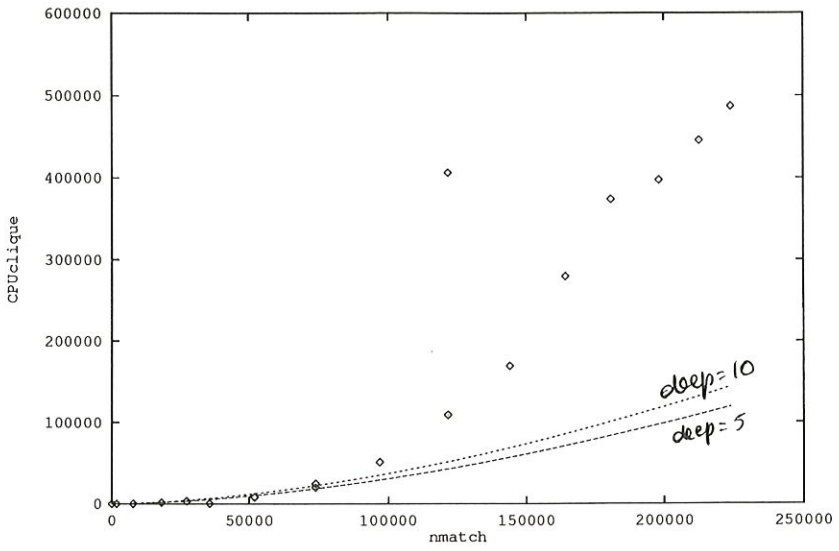


Figure 6.23: CPUclique pour n=13 et deep>1



Revenons au CPUclique. Sur les figures 6.21, 6.22 et 6.23 j'ai représenté le modèle de la formule 6.6 pour n=3, 5 et 13 respectivement. On voit que le modèle ne correspond pas bien aux résultats des tests, le R-square ne valait d'ailleur seulement 0.87.

Comme les courbes pour les différents tests sont fort différentes, il est très dur de trouver une relation générale pour tous les tests.

Après avoir essayé plusieurs modèles, nous avons considéré que le CPUclique est bien approché (en fonction de nmatch) par un modèle de la forme suivante :

$$CPUclique = \delta \ (deep)^{\gamma} \ p_3(nmatch) \tag{6.8}$$

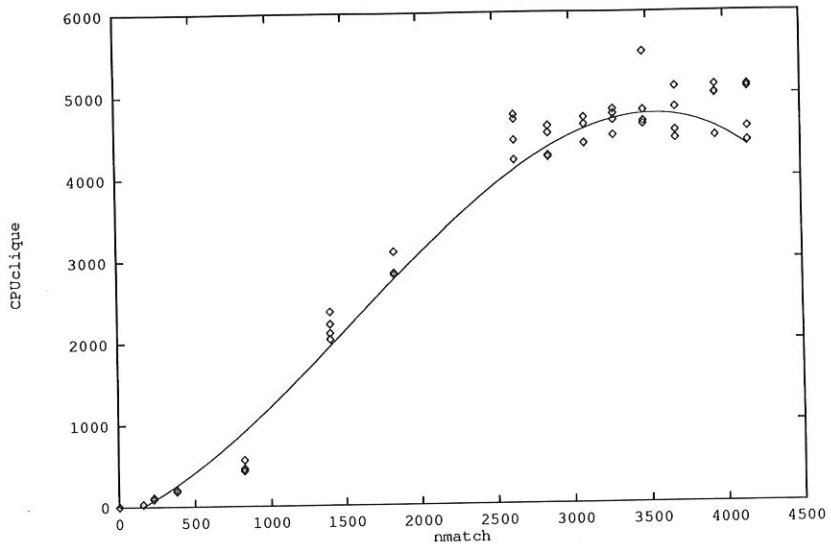
où p_3 un polynôme de degré 3. Les coefficients de p_3 ainsi que δ et γ dépendent des séquences choisies.

Pour tester le modèle de l'équation 6.8 les modèles linéaires ne suffisent pas, car on a une combinaison d'un modèle logarithmique et polynomial de degré 3.

Comme il nous était donc impossible de tester ce modèle pour toutes les données en une fois, nous avons fait des analyses séparément pour n=3, 5 ou 13.

Sur les figures 6.21, 6.22 et 6.23, nous voyons que deep n'influence seulement les résultats lorsque nmatch est élevé. Cela pourrait s'expliquer de la façon suivante : Lorsque nmatch est petit, les matrices de vérité, qui sont de dimension $((n-1) \times deep) \times ((n-1) \times deep)$ ne sont pas toutes remplies. En effet, lorsqu'il n'y a pas beaucoup de "matches", alors chaque fenêtre initiale possède qu'un nombre restreint de fenêtres mobiles qui lui sont similaires et par conséquent, les matrices de vérité pour les différents deep sont les mêmes à des colonnes et lignes vides près, qui n'influencent naturellement pas la recherche des cliques et par conséquent les temps CPUclique ne varient que de peu pour les différents deep. Par

Figure 6.24: Le CPUclique en fonction de nmatch pour n=3 et deep>1



contre si nmatch est très grand, les matrices de vérité peuvent varier fortement pour des tests avec des valeurs différentes de deep (idéalement on pourrait avoir $nfinit$ matrices de vérité de dimension $((n-1) \times deep) \times ((n-1) \times deep)$ remplies), et alors le CPUclique pour les différents deep change fortement et est naturellement beaucoup plus élevé pour des grandes valeurs de deep.

Comme deep n'influence donc que peu le modèle —sauf dans quelques cas que je préciserai par la suite — nous avons testé le modèle suivant

$$CPUclique = \alpha + \beta_1 \ nmatch + \beta_2 \ nmatch^2 + \beta_3 \ nmatch^3 \tag{6.9}$$

CPUclique pour n=3

Pour les tests 16 à 75 qui correspondent au tests avec n=3 et deep>1 nous avons testé le modèle de l'équation 6.9 et obtenu les résultats suivants :

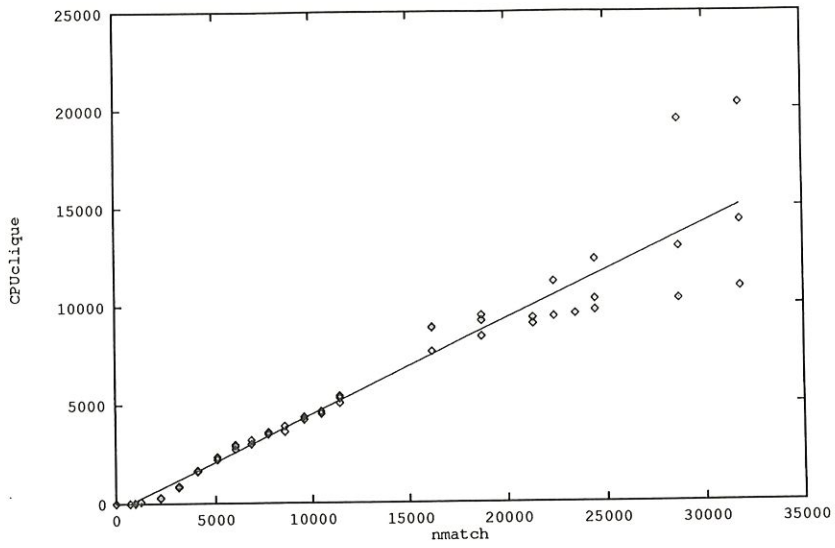
$$\begin{aligned} \alpha &= -136.6714392 \\ \beta_1 &= 0.7673645 \\ \beta_2 &= 0.0007406 \\ \beta_3 &= -0.0000002 \\ \text{R-square} &= 0.984481 \end{aligned}$$

La figure 6.24 montre les résultats de ce modèle.

CPUclique pour n=5

Pour les tests 95 à 152 qui correspondent au tests avec n=5 et deep>1 nous avons testé le modèle de l'équation 6.9 et obtenu les résultats suivants :

Figure 6.25: Le CPUclique en fonction de nmatch pour n=5 et deep>1



$$\begin{aligned} \alpha &= -516.6337879 \\ \beta_1 &= 0.5678382 \\ \beta_2 &= -0.0000070 \\ \beta_3 &= 0.0000000 \\ \text{R-square} &= 0.923570 \end{aligned}$$

Comme β_2 et β_3 sont très petit, nous supposons que nous pouvons simplifier le modèle et nous avons testé le modèle linéaire suivant (Figure 6.25) :

$$CPUclique = \alpha + \beta nmatch$$

Les résultats étaient encore très bons :

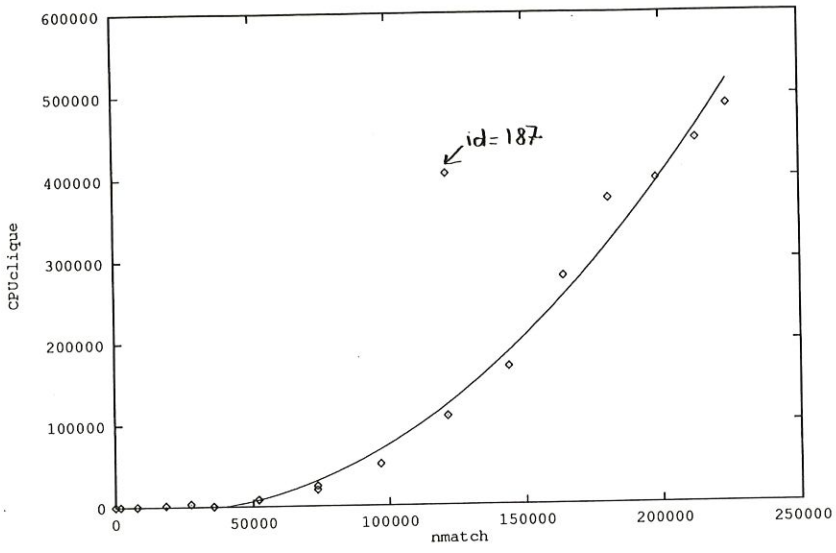
$$\begin{aligned} \alpha &= -345.8093944 \\ \beta &= 0.4847745 \\ \text{R-square} &= 0.923068 \end{aligned}$$

CPUclique pour n=13

Pour les tests 168 à 188 qui correspondent aux tests avec n=13 et deep>1 nous avons testé le modèle de l'équation 6.9 et obtenu les résultats suivants :

$$\begin{aligned} \alpha &= 88065.31180 \\ \beta_1 &= -11.87354 \\ \beta_2 &= 0.00019 \\ \beta_3 &= -0.00000 \\ \text{R-square} &= 0.321155 \end{aligned}$$

Figure 6.26: Le CPUclique en fonction de nmatch pour n=13 et deep>1



Le modèle est inadéquat pour ces données, ce qui est conforme à l'intuition, parce qu'il ne tient pas compte de la variable deep, or comme vous le voyez sur la figure 6.23, le CPUclique dépend fortement de la valeur de deep pour nmatch élevé. En fait il y a deux points correspondant aux tests 187 et 188 qui ont un comportement différents des autres. En réalité, on voit, que le temps CPU a tendance à croire exponentiellement pour les données avec deep=10, l'algorithme a trop de données à gérer.

Nous avons enlevé ces deux données de nos étude, et obtenu un résultat qui correspond beaucoup mieux aux données des 19 tests restant. Comme pour ce résultat β_3 valait zéro, nous avons testé un modèle quadratique et obtenu les résultats suivants (représenté sur la figure 6.26) :

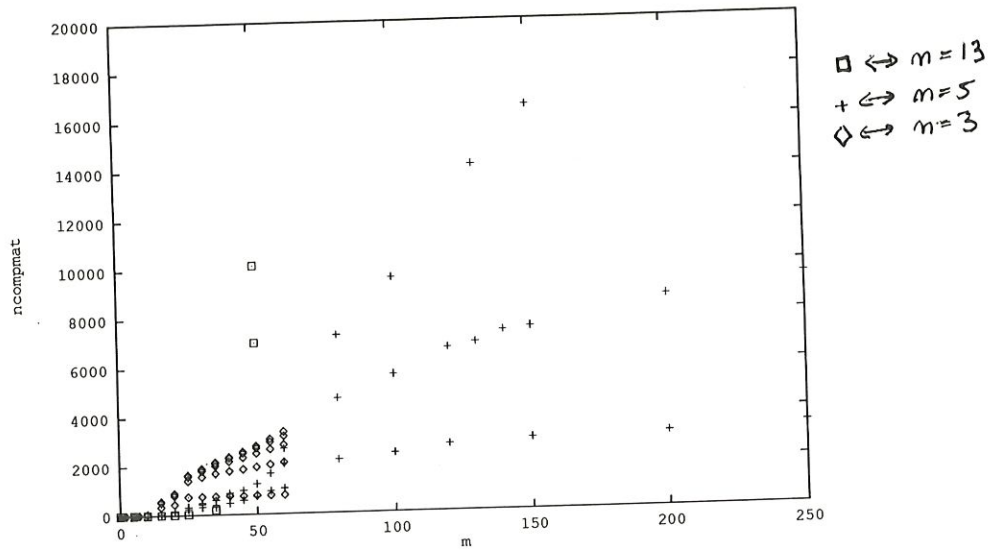
$$\begin{aligned} \alpha &= 651.9036933 \\ \beta_1 &= -0.5156653 \\ \beta_2 &= 0.0000126 \\ \text{R-square} &= 0.989747 \end{aligned}$$

6.2.5 Résultats des programmes "Match-cli" et "Match-Box"

Jusqu'à maintenant nous avons testé le comportement du CPU en fonction des dimensions du problème (m, deep, les séquences choisies). Une question que nous nous posons est de savoir si les résultats du programme *Match-cli* varient également en fonction des dimensions du problème, et s'ils varient proportionnellement au CPU.

Une première idée pour ce voir est de représenter le nombre de "matches" complets trouvés (ncompmat) dans le programme *Match-cli* en fonction de m (Figure 6.27 et 6.28).

Figure 6.27: Nombre de matches complets en fonction de m.



On voit que ce nombre augmente en fonction de m (de même que le CPUtotal), donc à un accroissement du CPU correspond un accroissement des résultats trouvés. Pour mieux encore saisir cette relation, nous réprésentons directement ncompmat en fonction de nmatch (Figure 6.29 et 6.30).

Une autre question que l'on peut se poser est de savoir si le résultat du programme *Match-Box* est mieux avec le programme *Match-cli* qu'avec le programme *Matching* et de savoir si les problèmes du programme *Match-Box* sont résolus (au moins en partie). Il est sûr que le programme initial (avec deep=1) est beaucoup plus économique. On se pose donc la question si la hausse du prix engendré par le choix de l'utilisation du programme *Match-cli* à la place du programme *Matching* correspond effectivement à une amélioration des résultats obtenus.

Nous avons vu (Figure 6.29 et 6.30) que plus nous trouvons de "matches", plus nous trouvons de "matches" complets, donc des segments similaires, mais d'un autre côté on complique le programme *Match-Box* considérablement. Déjà lors de la procédure de Screening, qui recherche les boîtes significatives, l'utilisateur sera confronté à des problèmes énormes si les données du problème initial sont trop importantes. Un autre désavantage est que si le nombre de "matches" est grand, on trouvera certainement plusieurs fois les mêmes boîtes, correspondant à des fenêtres initiales différentes, ce qui n'apportera pas de résultats supplémentaires. En plus il est clair, que l'on trouvera beaucoup de "matches" complets impossibles à aligner, surtout lorsque m est très grand (et par conséquent nmatch).

Il est impossible — au moins pour l'implémentation actuelle du programme *Screening* — de faire tourner le programme avec des données trop élevées, car le programme ne

Figure 6.28: Nombre de matches complets en fonction de m.

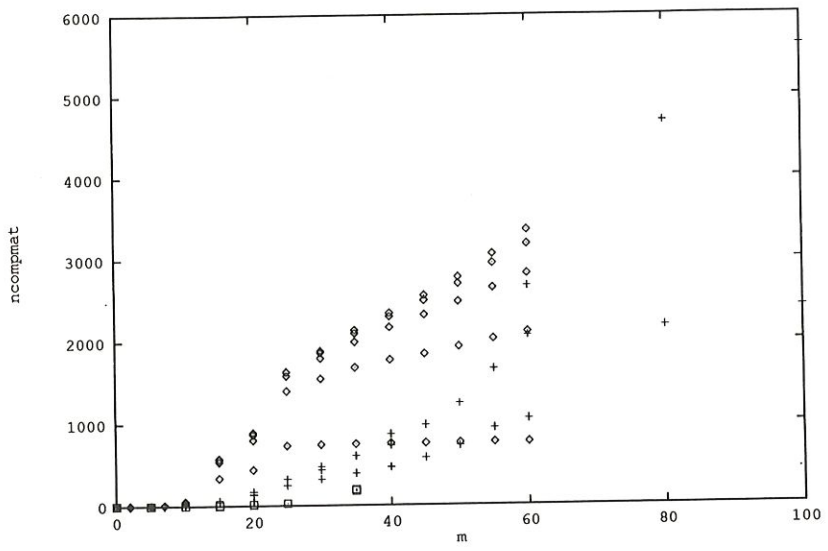


Figure 6.29: Nombre de matches complets en fonction de nmatch.

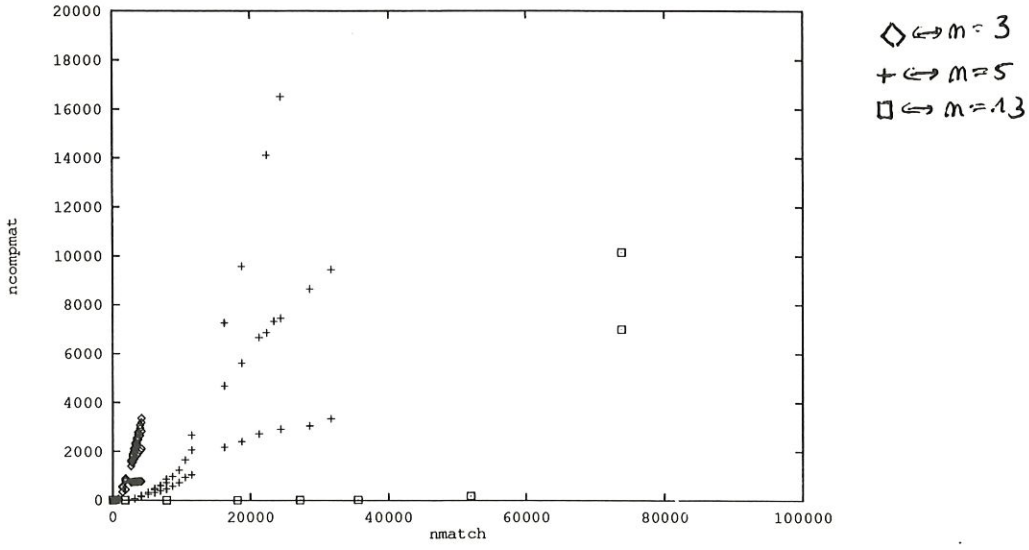
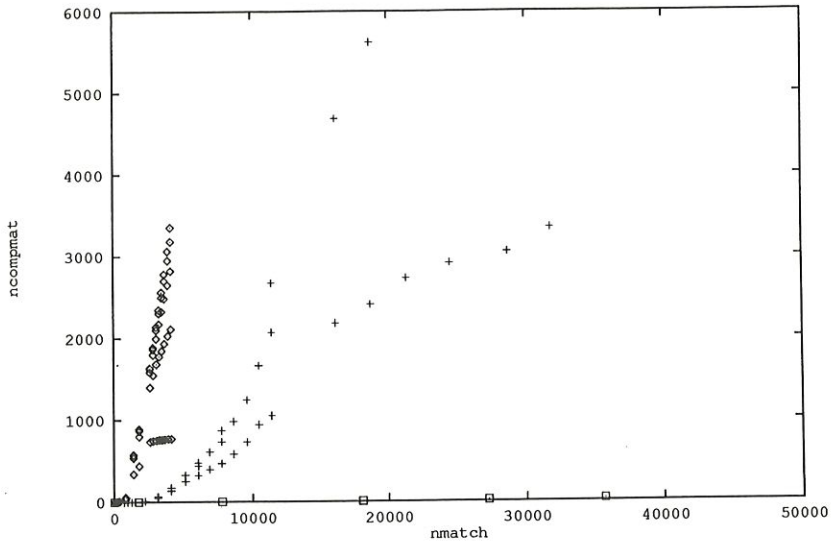


Figure 6.30: Nombre de matches complets en fonction de nmatch.



peut pas traiter des fichiers énormes. D'un autre côté, même si le programme *Screening* serait capable de traiter des fichiers énormes, il ne serait peut-être pas toujours souhaitable d'exécuter des problèmes avec des données trop importantes, à cause du temps CPU du programme *Match-Box*. En effet, si on pense que nous avons déjà obtenu des temps d'une heure ou plus pour l'exécution du programme *Match-cli*, on peut s'imaginer facilement que pour exécuter tout le programme Match-Box il faudrait un temps de dimension exponentielle.

Il serait naturellement intéressant de voir — au moins pour des données raisonnables — si l'alignement est vraiment meilleur avec deep>1, qu'avec deep=1, pour voir si la hausse du prix correspond effectivement à une amélioration des résultats.

6.3 Les résultats des tests

6.3.1 Données

id	n	m	cpu total	deep	ncomp mat	CPU setup	CPU sort	CPU trmat	CPU clique	nmatch	nfinit
1	3	0	130	1	0	130	0	0	0	2	948
2	3	2	149	1	0	149	0	0	0	158	948
3	3	5	338	1	4	337	0	1	0	229	948
4	3	7	359	1	8	358	0	1	0	383	948
5	3	10	458	1	37	456	0	2	0	827	948

6	3	15	564	1	337	535	0	29	0	1403	948	
7	3	20	660	1	438	624	0	36	0	1832	948	
8	3	25	789	1	733	728	0	61	0	2629	948	
9	3	30	886	1	743	830	0	56	0	2853	948	
10	3	35	960	1	749	900	0	60	0	3088	948	
11	3	40	1010	1	756	966	0	44	0	3278	948	
12	3	45	1160	1	756	1095	0	65	0	3472	948	
13	3	50	1151	1	761	1087	0	64	0	3678	948	
14	3	55	1178	1	768	1102	0	76	0	3937	948	
15	3	60	1200	1	768	1143	0	57	0	4151	948	
16	3	0	135	3	0	135	0	0	0	2	948	
17	3	2	181	3	0	152	1	0	28	158	948	
18	3	5	455	3	4	342	2	1	110	229	948	
19	3	7	565	3	10	372	1	4	188	383	948	
20	3	10	1046	3	54	461	5	11	569	827	948	
21	3	15	3045	3	534	605	30	38	2372	1403	948	
22	3	20	3888	3	797	698	40	51	3099	1832	948	
23	3	25	5732	3	1397	851	55	71	4755	2629	948	
24	3	30	5298	3	1547	914	52	89	4243	2853	948	
25	3	35	5488	3	1681	944	53	87	4404	3088	948	
26	3	40	5915	3	1773	1033	45	76	4761	3278	948	
27	3	45	5820	3	1844	1057	62	71	4630	3472	948	
28	3	50	5755	3	1932	1162	53	83	4457	3678	948	
29	3	55	6388	3	2027	1238	65	78	5007	3937	948	
30	3	60	5987	3	2108	1268	46	81	4592	4151	948	
31	3	0	129	5	0	129	0	0	0	2	948	
32	3	2	183	5	0	154	0	0	29	158	948	
33	3	5	434	5	4	334	2	1	97	229	948	
34	3	7	571	5	10	377	5	2	187	383	948	
35	3	10	892	5	54	443	6	10	433	827	948	
36	3	15	2691	5	568	591	25	39	2036	1403	948	
37	3	20	3592	5	866	691	32	44	2825	1832	948	
38	3	25	5400	5	1581	834	54	67	4445	2629	948	
39	3	30	5570	5	1795	898	63	77	4532	2853	948	
40	3	35	5797	5	1992	963	55	59	4720	3088	948	
41	3	40	5885	5	2170	1028	77	97	4683	3278	948	
42	3	45	6032	5	2321	1089	53	91	4799	3472	948	
43	3	50	6124	5	2483	1146	58	87	4833	3678	948	

44	3	55	5789	5	2648	1158	68	78	4485	3937	948
45	3	60	5922	5	2822	1348	69	82	4423	4151	948
46	3	0	137	7	0	137	0	0	0	2	948
47	3	2	187	7	0	154	1	2	30	158	948
48	3	5	458	7	4	354	1	1	102	229	948
49	3	7	572	7	10	363	3	4	202	383	948
50	3	10	901	7	54	446	3	5	447	827	948
51	3	15	2773	7	573	602	30	29	2112	1403	948
52	3	20	3596	7	886	686	34	58	2818	1832	948
53	3	25	5144	7	1627	809	57	72	4206	2629	948
54	3	30	5277	7	1863	887	63	69	4258	2853	948
55	3	35	5485	7	2096	945	55	77	4408	3088	948
56	3	40	5647	7	2302	992	60	95	4500	3278	948
57	3	45	5869	7	2496	1069	51	84	4665	3472	948
58	3	50	5857	7	2700	1135	68	102	4552	3678	948
59	3	55	6363	7	2950	1209	75	82	4997	3937	948
60	3	60	6519	7	3182	1292	57	85	5075	4151	948
61	3	0	137	10	0	137	0	0	0	2	948
62	3	2	182	10	0	153	0	0	29	158	948
63	3	5	437	10	4	344	0	3	90	229	948
64	3	7	578	10	10	365	3	7	203	383	948
65	3	10	917	10	54	443	6	7	461	827	948
66	3	15	2864	10	573	592	20	30	2222	1403	948
67	3	20	3608	10	887	700	32	38	2838	1832	948
68	3	25	5713	10	1630	872	68	74	4699	2629	948
69	3	30	5655	10	1883	885	57	96	4617	2853	948
70	3	35	5722	10	2135	967	48	73	4634	3088	948
71	3	40	5991	10	2343	1036	61	80	4814	3278	948
72	3	45	6260	10	2559	1105	62	84	5509	3472	948
73	3	50	6421	10	2781	1189	72	78	5082	3678	948
74	3	55	6490	10	3063	1220	61	105	5104	3937	948
75	3	60	6546	10	3353	1282	57	105	5102	4151	948
76	5	0	192	1	0	192	0	0	0	4	1722
77	5	2	191	1	0	191	0	0	0	708	1722
78	5	5	1071	1	0	1071	0	0	0	964	1722
79	5	7	1112	1	0	1112	0	0	0	1256	1722
80	5	10	1266	1	5	1264	0	2	0	2229	1722
81	5	15	1436	1	25	1432	0	4	0	3152	1722

82	5	20	1628	1	69	1624	0	4	0	4099	1722	
83	5	25	1825	1	109	1810	0	15	0	5111	1722	
84	5	30	2094	1	147	2073	0	21	0	6040	1722	
85	5	35	2291	1	165	2275	0	16	0	6870	1722	
86	5	40	2500	1	172	2484	0	16	0	7737	1722	
87	5	45	2728	1	185	2712	0	16	0	8606	1722	
88	5	50	2875	1	205	2849	0	26	0	9600	1722	
89	5	55	3152	1	253	3128	0	24	0	10485	1722	
90	5	60	3323	1	282	3293	0	30	0	11438	1722	
91	5	80	5209	1	675	5109	0	100	1	16200	1722	
92	5	100	6120	1	665	6035	0	85	1	18752	1722	
93	5	120	6914	1	672	6816	0	98	0	21312	1722	
94	5	150	8005	1	673	7923	0	82	0	24472	1722	
95	5	0	199	3	0	199	0	0	0	4	1722	
96	5	2	196	3	0	196	0	0	0	708	1722	
97	5	5	1097	3	0	1076	2	0	19	964	1722	
98	5	7	1220	3	0	1124	2	3	91	1256	1722	
99	5	10	1551	3	8	1265	4	7	275	2229	1722	
100	5	15	2285	3	52	1430	19	20	816	3152	1722	
101	5	20	3423	3	136	1698	22	61	1642	4099	1722	
102	5	25	4202	3	247	1899	25	64	2214	5111	1722	
103	5	30	5242	3	321	2197	37	101	2907	6040	1722	
104	5	35	5718	3	393	2358	49	113	3198	6870	1722	
105	5	40	6186	3	466	2504	56	136	3490	7737	1722	
106	5	45	6653	3	580	2813	56	152	3632	8606	1722	
107	5	50	7508	3	728	3049	64	164	4231	9600	1722	
108	5	55	8138	3	938	3246	60	188	4644	10485	1722	
109	5	60	8861	3	1049	3524	83	197	5067	11438	1722	
110	5	80	14648	3	2174	5246	143	372	8887	16200	1722	
111	5	100	16200	3	2405	6162	173	382	9483	18752	1722	
112	5	120	16699	3	2725	7046	215	393	9045	21312	1722	
113	5	150	19145	3	2910	8151	279	418	10297	24472	1722	
114	5	200	21171	3	3048	10028	355	495	10293	28703	1722	
115	5	250	22724	3	3344	10809	491	533	10891	31794	1722	
116	5	0	195	5	0	195	0	0	0	4	1722	
117	5	2	191	5	0	191	0	0	0	708	1722	
118	5	5	1105	5	0	1080	0	2	23	964	1722	
119	5	7	1225	5	0	1125	0	4	96	1256	1722	

120	5	10	1564	5	8	1261	2	8	293	2229	1722
121	5	15	2359	5	64	1451	16	25	867	3152	1722
122	5	20	3373	5	170	1682	19	49	1623	4099	1722
123	5	25	4353	5	325	1882	42	90	2339	5111	1722
124	5	30	4996	5	436	2119	38	99	2740	6040	1722
125	5	35	5488	5	606	2329	27	125	3007	6870	1722
126	5	40	6317	5	730	2529	55	139	3594	7737	1722
127	5	45	6866	5	977	2731	58	168	3909	8606	1722
128	5	50	7586	5	1241	2965	61	191	4369	9600	1722
129	5	55	8043	5	1658	3210	79	235	4519	10485	1722
130	5	60	9068	5	2065	3447	52	255	5314	11438	1722
131	5	80	12785	5	4682	4602	136	387	7660	16200	1722
132	5	100	14471	5	5618	5464	145	454	8408	18752	1722
133	5	120	16213	5	6668	6168	178	517	9350	21312	1722
134	5	130	16654	5	6867	6510	202	526	9416	22379	1722
135	5	140	17214	5	7333	6906	227	527	9554	23465	1722
136	5	150	17686	5	7455	7111	254	584	9737	24472	1722
137	5	200	23877	5	8652	9733	398	793	12953	28703	1722
138	5	250	26634	5	9446	10967	519	863	14285	31794	1722
139	5	0	198	10	0	198	0	0	0	4	1722
140	5	5	1086	10	0	1066	0	1	19	964	1722
141	5	10	1549	10	8	1249	2	7	291	2229	1722
142	5	15	2360	10	64	1464	8	29	859	3152	1722
143	5	20	3419	10	172	1676	23	52	1668	4099	1722
144	5	30	5235	10	476	2120	38	116	2961	6040	1722
145	5	40	6319	10	866	2505	56	171	3587	7737	1722
146	5	60	9306	10	2670	3491	74	318	5423	11438	1722
147	5	80	14104	10	7264	4614	143	494	8853	16200	1722
148	5	100	15359	10	9576	5381	178	591	9209	18752	1722
149	5	130	18717	10	14121	6528	233	764	11192	22379	1722
150	5	150	20565	10	16504	7081	271	892	12321	24472	1722
151	5	200	30926	10	20723	9727	397	1356	19446	28703	1722
152	5	250	33111	10	23077	10808	482	1542	20279	31794	1722
153	13	0	653	1	0	653	0	0	0	2	4330
154	13	5	658	1	0	658	0	0	0	1762	4330
155	13	10	15146	1	0	15146	0	0	0	7750	4330
156	13	15	16584	1	4	16581	0	3	0	18119	4330
157	13	25	19687	1	22	19673	0	14	0	35647	4330

158	13	35	22948	1	70	22920	0	28	0	52015	4330
159	13	50	27790	1	97	27749	0	41	0	73773	4330
160	13	65	33538	1	254	33411	0	127	0	96936	4330
161	13	80	40083	1	683	39772	0	311	0	121313	4330
162	13	100	46367	1	773	46367	0	345	0	143842	4330
163	13	120	52428	1	767	52064	0	364	0	164154	4330
164	13	140	57694	1	765	57341	0	353	0	180753	4330
165	13	160	62519	1	760	62168	0	351	0	198180	4330
166	13	180	66842	1	759	66484	0	358	0	212515	4330
167	13	200	72815	1	757	72447	0	368	0	223867	4330
168	13	0	648	5	0	648	0	0	0	2	4330
169	13	5	661	5	0	661	0	0	0	1762	4330
170	13	10	15318	5	0	15147	5	22	144	7750	4330
171	13	15	18382	5	4	16620	26	313	1423	18119	4330
172	13	25	26334	5	29	20156	105	1467	460	35647	4330
173	13	35	34888	5	180	23561	142	2979	8206	52015	4330
174	13	50	54270	5	6996	28664	286	5254	20066	73773	4330
175	13	65	91456	5	34032	33551	409	7314	51182	96939	4330
176	13	80	159732	5	84024	40134	620	9779	109199	121313	4330
177	13	100	228764	5	145250	46894	896	11849	169125	143842	4330
178	13	120	346777	5	256602	52863	1269	13603	279042	164154	4330
179	13	140	448544	5	329861	58442	1634	15017	373451	180753	4330
180	13	160	479411	5	386134	63646	1968	16319	397178	198180	4330
181	13	180	533528	5	425582	68362	2324	17455	445387	212515	4330
182	13	200	580238	5	443530	72288	2637	18225	487088	223867	4330
183	13	0	660	10	0	660	0	0	0	2	4330
184	13	20	22713	10	14	18386	55	951	3321	27247	4330
185	13	35	35025	10	191	23112	158	3298	8457	52015	4330
186	13	50	59339	10	10147	27958	285	6395	24630	73773	4330
187	13	80	460852	10	303464	40135	740	14213	405764	121313	4330
188	13	140	2853740	10	1755444	59058	1744	26997	2765941	180753	4330

6.3.2 L'alignement des 13 séquences

```
MATCH-BOX 3.0 feb 1994                                  3-MAY-94   15:19:56

      1      1      G3P1_CAEEL.SW
      2      2      G3P1_ECOLI.SW
      3      3      G3P1_ESCVU.SW
```

```
 4     4     G3P1_HUMAN.SW
 5     5     G3P1_YEAST.SW
 6     6     G3PA_MAIZE.SW
 7     7     G3PC_TOBAC.SW
 8     8     G3PG_TRYBB.SW
 9     9     G3PG_TRYCR.SW
10    10     G3P_BACST.SW
11    11     G3P_HOMAM.SW
12    12     G3P_THEAQ.SW
13    13     G3P_THEMA.SW
```

```
          10        20        30        40        50        60        70
           +         +         +         +         +         +         +
 1 ------------------------------------------------------------MSK
 2 ------------------------------------------------------------MT
 3 ------------------------------------------------------------
 4 ------------------------------------------------------------GK
 5 ------------------------------------------------------------
 6 MASSMLSATTVPLQQGGGLSEFSGLRSSASLPMRRNATSDDFMSAVSFRTHAVGTSGGPRRAPTEAKLKV
 7 ------------------------------------------------------------
 8 ----------------------------------------------------------TIKVGING
 9 ---------------------------------------------------------MPIKVGING
10 ------------------------------------------------------------A
11 ------------------------------------------------------------
12 ------------------------------------------------------------
13 ------------------------------------------------------------AR
```

```
          80        90       100       110       120       130       140
           +         +         +         +         +         +         +
 1 ANVGINGFGRIGRLVLRAAVEKDTvqvvavnDPFITIDYMVY------lfkydsthGQFKGTVTYDGDFL
 2 IKVGINGFGRIGRIVFRAAQKRSDieivainDLLDADYMAY-------mlkydsthGRFDGTVEVKDGHL
 3 ------------IVFRAAQKRSDieivainDLLDAEYMAY-------mlkydsthGRFDGTVEVKDGHL
 4 VKVGVDGFGRIGRLVTRAAFNSGKvdivainDPFIDLHYMVY------mfqydsthGKFHGTVKAEDGKL
 5 VRVAINGFGRIGRLVMRIALQRKNvevvalnDPFISNDYSAY------mfkydsthGRYAGEVSHDDKHI
 6 AINGFGRIGRNFLRCWHGRGDASPldviainDTGGVKQASH-------llkydstlGIFDADVKPVGDNA
 7 --------GRIGRLVARVALQRDDvelvavnDPFISTDYMTY------mfkydsvhGQWKHHELKVKDEK
 8 FGRIGRMVFQALCDDGLLGNEIDVvavvdmnTDARYFAY---------qmkydsvhGKFKHSVSTTKSKP
 9 FGRIGRMVFQALCEDGLLGTEIDVvavvdmnTDAEYFAYQMRYDTVHGkfkyevttTKSSPSVAKDDTLV
```

```
10  VKVGINGFGRIGRNVFRAALKNPDievvavnDLTANADGLAH------llkydsvhGRLDAEVVVNDGVS
11  -SKIGIDGFGRIGRLVLRAALSCGaqvvavnDPFIALEYMVY------mfkydsthGVFKGEVKMEDGAL
12  -MKVGINGFGRIGRQVFRILHSRGvevalinDLTDNKTLAH-------llkydsiyHRFPGEVAYDDQYL
13  VAINGFGRIGRLVYRIIYERKNPDievvainDLTDTKTLAH-------llkydsvhKKFPGKVEYTENSL

              150       160       170       180       190       200       210
               +         +         +         +         +         +         +
 1  IVQKDGKSSHKIKVFNSKDPAAIAWGSVKA----dfvvestgvfttkEKASAHLqggakkviisAPSADA
 2  IVNGKKIRVTAERDPANLKWDEVGV---------dvvaeatglfltdETARKHItagakkvvmtGPSKDN
 3  VVNGKKIRVTAERDPANLKWDEVGV---------dvvaeatgifltdETARKHItagakkvvltGPSKDN
 4  VIDGKAITIFQERDPENIKWGDAGT---------ayvvestgvfttmEKAGAHLkggakrivisAPSADA
 5  IVDGHKIATFQERDPANLPWASLNI---------diaidstgvfkelDTAQKHIdagakkvvitAPSSTA
 6  ISVDGKVIKVVSDRNPSNLPWGELGI--------dlviegtgvfvdrEGAGKHIqagakkvlitAPGKGD
 7  TLLFGEKSVRVFGIRNPEEIPWAEAGA-------dfvvestgvftdkDKAAAHLkggakkvvisAPSKDA
 8  SVAKDDTLVVNGHRILCVKAQRNPADLPWGKLGVeyviestglftvkSAAEGHLrggarkvvisAPASGG
 9  VNGHRILCVKAQRNPADLPWGKLGV---------eyviestglftakAAAEGHLrggarkvvisAPASGG
10  VNGKEIIVKAERNPENLAWGEIGV----------divvestgrftkrEDAAKHLeagakkviisAPAKVE
11  VVDGKKITVFNEMKPENIPWSKAGA---------eyivestgvfttiEKASAHFkggakkvvisAPSADA
12  YVDGKAIRATAVKDPKEIPWAEAGV---------gvviestgvftdaDKAKAHLeggakkviitAPAKGE
13  IVDGKEIKVFAEPDPSKLPWKDLGV---------dfviestgvfrnrEKAELHLqagakkviitAPAKGE

              220       230       240       250       260       270       280
               +         +         +         +         +         +         +
 1  PMYVVGVNHEKYDASND-hvvsnascttnclaplakviNDNFGIIE-glmttvhavtatqKTVDGPSGKL
 2  TPMFVKGANFDKYAGQ--divsnascttnclaplakviNDNFGIIE-glmttvhattatqKTVDGPSHKD
 3  TPMFVRGANFDTYAGQ--divsnascttnclaplakviNDNFGIVE-glmttvhattatqKTVDGPSHKD
 4  PMFVMGVNHFKYANSL--kiisnascttnclaplakviHDHFGIVE-glmttvhaitatqKTVDSPSGKL
 5  PMFVMGVNEEKYTSDL--kivsnascttnclaplakviNDAFGIEE-glmttvhsmtatqKTVDGPSHKD
 6  IPTYVVGVNADQYNPDE-piisnascttnclapfvkvlDQKFGIIK-gtmttthsytgdqRLLDASHRDL
 7  PMFVVGVNEKEYKPEY--divsnascttnclaplakviNDRFGIVE-glmttvhsltatqKTVDGPSMKD
 8  AKTFVMGVNHNNYNPREQhvvsnascttnclaplvhvlVKEGFGISTglmttvhsytatqKTVDGVSVKD
 9  AKTLVMGVNHHEYNPSEHhvvsnascttnclapivhvlVKEGFGVQTglmttihsytatqKTVDGVSVKD
10  NITVVMGVNQDKYDPKAHhvisnascttnclapfakvlHQEFGIVR-gmmttvhsytnnqRILDLPTHKD
11  PMFVCGVNLEKYSKDM--tvvsnascttnclapvakvlHENFEIVE-glmttvhavtatqKTVDGPSAKD
12  DITIVMGVNHEAYDPSRHhiisnascttnslapvmkvlEEAFGVEK-almttvhsytndqRLLDLPHKDL
13  DITVVIGCNEDQLKPEH-tiiscascttnsiapivkvlHEKFGIVS-gmlttvhsytndqRVLDLPHKDL

              290       300       310       320       330       340       350
```

```
                +         +         +         +         +         +         +
1  WRDGRGagqniipastgaakavgkvipelngkltgmafrvptpdvsvvdltvRLEKPASMDDIKKVVKAA
2  WRGGRGasqniipsstgaakavgkvlpelngkltgmafrvptpnvsvvdltvRLEKAATYEQIKAAVKAA
3  WRGGRGaaqniipsstgaakavgkvlpelngkltgmafrvptpnvsvvdltvRLEKAASYEEIKKAIKAA
4  WRGGRGaaqnlipastgaakavgkvipeldgkltgmafrvptanvsvldltcRLEKPAKYDDIKKVVKEA
5  WRGGRTasgniipsstgaakavgkvlpelqgkltgmafrvptvdvsvvdltvKLNKETTYDEIKKVVKAA
6  RRARA-aalnivptstgaakavslvlpnlkgklngialrvptpnvsvvdlvvQVSKKTLAEEVNQAFRDA
7  WRGGRAtsfniipsstgaakavgkvlpalngkltgmafrvptvdvsvvdltvRLEKEASYDDIKAAIKEE
8  WRGGRAaalniipsttgaakavgmvipstqgkltgmafrvptadvsvvdltfIATRDTSIKEIDAALKRA
9  WRGGRAaavniipsttgaakavgmvipstqgkltgmsfrvptpdvsvvdltfTAARDTSIQEIDAALKRA
10 LRGARAaaesiiptttgaakavalvlpelkgklngmamrvptpnvsvvdlvaELEKEVTVEEVNAALKAA
11 WRGGRGaaqniipsstgaakavgkvipeldgkltgmafrvptpdvsvvdltvRLGKECSYDDIKAAMKTA
12 RRARA-aainiiptttgaakatalvlpslkgrfdgmalrvptatgsisditaLLKREVTAEEVNAALKAA
13 RRARA-aavniiptttgaakavalvvpevkgkldgmairvptpdgsitdltvLVEKETTVEEVNAVMKEA

          360       370       380       390       400       410       420
           +         +         +         +         +         +         +
1  ADGPMKGILAYTEDQVVSTDFVSDPHssifdagACISLNPNFVKLVSWYDNEYGYSNRVVDLIGYIATRG
2  AEGEMKGVLGYTEDDVVSTDFNGEVCtsvfdgkGGMGLNDNFVKLVSWYDNETGYSNKVLDLIAHISK--
3  SEGAMKGVLGYTEDDVVSTDFNGEVCtsvfdakAGIALNDNFVKLV----------------------
4  SEGPLKGILGYTEDEVVSDDFNGSNHssifdagAGIELNDTFVKLVSWYDNEFGYSERVVDLMAHMASKE
5  AEGKLKGVLGYTEDAVVSSDFLGDSNssifdaaAGIQLSPKFVKLVSWYDNEYGYSTRVVDLVEHVAKA-
6  AANELTGILEVCDVPLVSVDFRCSDVsstidasLTMVMGDDMVKVISWYDNEWGYSQRVVDLADICANQW
7  SEGKLKGILGFTEDDVVSTDFVGDSRssifdakAGIALSKNFVKLVSWYDNEWGYSSRVIDLICHMASVA
8  SKTYMKNILGYTDEELVSADFISDSRssiydskATLQNNLPNERRFFKIVSWYDNEWGYSHRVVDLVRHM
9  SKTYMKGILGYTDEELVSADFINDNRssiydskATLQNNLPKERRFFKIVSWYDNEWGYSHRVVDLVRHM
10 AEGELKGILAYSEEPLVSRNYNGSTVsstidalSTMVIDGKMVKVVSWYDNETGYSHRVVDLAAYINAKG
11 SEGPLQGFLGYTEDDVVSSDFIGDNRssifdakAGIQLSKTFVKVVSWYDNEFGYSQRVIDLLKHMQKVD
12 AEGPLKGILAYTEDEIVLQDIVMDPHssivdakLTKALGNMVKVFAWYDNEWGYANRVADLVELVLRKGV
13 TEGRLKGIIGYNDEPIVSSDIIGTTFsgifdatITNVIGGKLVKVASWYDNEYGYSNRVVDTLELLLKM-

          430       440       450       460       470       480       490
           +         +         +         +         +         +         +
1  ---------
2  ---------
3  ---------
4  ---------
5  ---------
```

```
 6  K--------
 7  ---------
 8  AARDRAAKL
 9  ASKDRSARL
10  L--------
11  SA-------
12  ---------
13  ---------
```

Conclusion

Dans ce mémoire nous avons décrit la réalisation d'un programme appelé *Clique* de recherche de toutes les cliques maximales d'un graphe donné. Pour ce faire, nous nous sommes basés sur le mémoire de Florence Badoux, auteur de cet algorithme, en y apportant les modifications nécessaires.

L'idée fondatrice de cet algorithme a été d'améliorer le programme *Match-Box* créé par E. Depiereux et E. Feytmans de l'Unité de Biologie quantitative des FUNDP de Namur; ce programme a pour objectif de prédire des régions structurellement conservées au sein des protéines.

Le point de départ du programme *Clique* était la constatation suivant laquelle le programme *Match-Box* échouait parfois dans la prédiction des régions structurellement conservées.

Une idée d'amélioration du programme *Match-Box* fut alors de prendre plus qu'un "match" similaire par séquence pour chaque fenêtre initiale, ce qui augmente la complexité de l'algorithme. Pour l'algorithme original, qui ne gardait que la fenêtre mobile la plus similaire de chaque séquence pour une fenêtre initiale donnée, on pouvait trouver au maximum un "match" complet pour cette fenêtre initiale, correspondant aux fenêtres mobiles choisies pour cette fenêtre initiale. Pour le nouveau algorithme, on peut garder plusieurs "matches" par séquence pour chaque fenêtre initiale et par conséquent on peut trouver plus qu'un "match" complet par fenêtre initiale. Le problème consiste alors à trouver tous ces "matches" complets et ceci correspond au problème de *recherche de cliques maximales dans un graphe donné.*

Une première partie de notre travail consistait à la rédaction bibliographique de l'algorithme *clique* ainsi que du problème biologique associé. Une deuxième partie fut alors de tester l'efficacité du programme *clique*, qui a été implémenté et incorporé dans l'ensemble du programme *Match-Box* par E. Depiereux, à partir des situations réelles. De ces tests nous avons essayé de dégager une relation entre le CPU nécessaire à l'exécution du programme et les données du problème.

Malheureusement, par manque de temps, nous n'avons pas entrepris dans le cadre de ce mémoire tous les tests possibles du programme *Match-Box*. Pour un travail futur, ce mémoire pourrait être complété sur un certain nombre de points.

Premièrement, il y a probablement moyen de faire des analyses supplémentaires à partir des résultats que nous avons obtenus pour nos tests, en particulier pour étudier la relation entre les résultats obtenus et l'augmentation du coût de calcul due à l'utilisation du nouveau programme *Match-cli* (qui contient le programme *clique*) à la place du programme existant *Matching*. Il serait intéressant de voir les limites du programme *Match-Box* et pas seulement d'un de ses sous-programmes, et la relation entre les résultats obtenus et le coût de calcul.

Nous nous sommes limités à l'étude de certaines variables pour nos tests. Comme nous l'avons déjà proposé au chapitre 6, il serait intéressant de faire tourner le programme en changeant également d'autre données, comme par exemple la variable d'élagage et les valeurs de *cutoff* choisies. Comme l'alignement dépend des séquences choisies et de leurs similarités, on pourrait faire des tests avec d'autres séquences et essayer de dégager une relation qui dépendra certainement des variables que nous avons considérées, mais en plus de la similarité entre les séquences.

On a vu au chapitre 6, que pour diminuer le temps du programme *Match-cli*, on doit diminuer le CPUclique (comme c'est la composante majeure du CPUtotal), une idée pour améliorer directement l'alignement Match-Box serait donc d'essayer d'améliorer — dans la mesure du possible — le programme *clique*.

Nous avons seulement testé le programme par rapport au temps CPU utilisé, on pourrait le tester au niveau de l'espace disque utilisé. Nous avons eu la chance d'avoir eu presque 250 000 blocs à notre disposition ainsi qu'un temps CPU pratiquement illimité, mais il est clair qu'il n'est pas toujours possible de faire tourner un programme dont l'exécution d'une partie ("Matching") nécessite déjà une heure de CPU ou plus et d'un espace disque énorme. Pour mieux saisir la grandeur du problème posé, voici la grandeur en blocs du fichier clique.dat pour quelques tests réalisés

n	*deep*	m	nombre de blocs de clique.dat
13	5	120	42 102
13	5	140	51 030
13	5	200	69 684
13	10	140	126 246

On comprend donc aisément que la suite du programme *Match-Box* dans son entièreté ne "marchera" pas avec des gros fichiers pareils.

Annexe A

Pseudo-code de l'algorithme de C.Bron et J.Kerbosch

A.1 Première version

```
COMPSUB← ∅
CLIQUE(V, ∅)
procédure CLIQUE(CANDIDATES, NOT)
    if CANDIDATES ∪ NOT = ∅ then output COMPSUB, which is a clique
        else if CANDIDATES ≠ ∅ then
            begin [[explore first subtree]]
                f ← vertex in CANDIDATES
                EXPLORE(f)
                [[explore remaining subtrees, if any, not precluded by theorem 1 (voir
                théorème 1 dans "La recherche exhaustive et le problème de clique".]]
                while CANDIDATES ∩ (V - Adj(f)) ≠ ∅do
                    begin
                        v ← vertex in CANDIDATES ∩ (V-Adj(f))
                        EXPLORE(v)
                    end
                end
                [[si CANDIDATES = ∅et NOT ≠ ∅ alors le théorème 2 (voir ) dit
                qu'on ne trouvera plus de nouvelle clique]]
            end
return

procedure EXPLORE(u)
    CANDIDATES ← CANDIDATES - {u}
    COMPSUB ← COMPSUB ∪ {u}
    CLIQUE(CANDIDATES ∩ Adj(u), NOT ∩ Adj(u))
    COMPSUB ← COMPSUB - {u}
    NOT ← NOT ∪ {u}
return
```

A.2 Deuxième version

```
COMPSUB ← ∅
CLIQUE(V, ∅)
procédure CLIQUE(CANDIDATES, NOT)
    if CANDIDATES ∪ NOT = ∅ then output COMPSUB, which is a clique
        else if CANDIDATES ≠ ∅ then
            begin
                f ← vertex in CANDIDATES ∪ NOT, which maximize
    | CANDIDATES - Adj(f)|
                if f ∈ CANDIDATES then [[explore first subtree]]
                EXPLORE(f)
                [[explore remaining subtrees, if any, not precluded by theorem 1 (voir
                théorème 1 dans "La recherche exhaustive et le problème de clique".]]
                while CANDIDATES ∩ (V - Adj(f)) ≠ ∅ do
                    begin
                        v ← vertex in CANDIDATES ∩ (V-Adj(f))
                        EXPLORE(v)
                    end
                end
                [[si CANDIDATES = ∅ et NOT ≠ ∅ alors le théorème 2 (voir ) dit
                qu'on ne trouvera plus de nouvelle clique]]
            end
return

procedure EXPLORE(u)
    CANDIDATES ← CANDIDATES - {u}
    COMPSUB ← COMPSUB ∪ {u}
    CLIQUE(CANDIDATES ∩ Adj(u), NOT ∩ Adj(u))
    COMPSUB ← COMPSUB - {u}
    NOT ← NOT ∪ {u}
return
```

Annexe B

Solution algorithmique de l'algorithme clique

B.1 Structure de données

B.1.1 Constantes du problème

- **Une matrice d'adjacence MAT** de dimension N×N, avec N le nombre de "matches" trouvés, donc le nombre de sommets du graphe.

$$MAT_{ij} = \begin{cases} 1 & \text{si les "matches" i et j sont} \\ & \text{issus de séquences de protéines} \\ & \text{différentes et s'ils sont similaires.} \\ 0 & \text{sinon.} \end{cases}$$

	0		n_1			n_2			$N-1$	
	A_1	...	A_{m_A}	B_1	...	B_{m_B}	...	Z_1	...	Z_{m_Z}
A_1		...			...		...		...	
...		...			...		...		...	
A_{m_A}		...			...		...		...	
B_1		...			...		...		...	
...		...			...		...		...	
B_{m_B}		...			...		...		...	
...		...			...		...		...	
Z_1		...			...		...		...	
...		...			...		...		...	
Z_{m_Z}		...			...		...		...	

Chaque fois que nous rentrons dans FINDCLI, cette variable est initialisée par le premier sommet du groupe suivant, et à l'intérieur de cette procédure, une boucle le fait progresser jusqu'au dernier sommet du groupe.

Donc si on traite le $x^{ième}$ groupe, J est tel que

$$IFINGPE(X)+1 \leq J \leq IFINGPE(X+1)$$

Variables locales de EXISTCANDEXT

EXISTCANDEXT vérifie s'il existe encore de candidats possibles dans chacun des groupes. La procédure utilise les variables locales suivantes :

- **Une variable Z** qui comporte l'indice du groupe suivant celui du "match" que nous essayons d'étendre, donc $2 \leq Z \leq S$. Z permet de passer de groupe en groupe tant qu'ils comprennent encore des candidats possibles.

- **Une variable Y** qui pointe vers le sommet pour lequel nous vérifions s'il sera un candidat possible lorsque le sommet que nous étendons sera ajouté à CLIEXT.

$$IFINGPE(Z) \leq IFINGPE(Z+1)-1$$

- **Une variable booléenne OK** initialisé à faux lorsque nous changeons de groupe et valant vrai dès que l'on a trouvé un candidat possible dans ce groupe.

- **Un tableau d'indices IFINGPE** qui donne l'indice de la colonne du dernier "match" des différents groupes. Si S est le nombre de groupes, c'est-à-dire le nombre de séquences de protéines, alors IFINGPE est de dimension 1×(S+1).

	0	1		S
$IFINGPE$:	-1	n_1	...	n_S

Soit m_i le nombre de "matches" trouvés dans la séquence i, on peut écrire la relation suivante :

$$m_i = IFINGPE(i) - IFINGPE(i-1)$$

on a donc

$$\begin{aligned} m_A &= n_1 - (-1) = n_1 + 1 \\ m_B &= n_2 - n_1 \\ &\dots \end{aligned}$$

N est la somme des m_i.

B.1.2 Variables globales du problème

- **Une variable X** qui indique le groupe courant qui est étendu en pointant vers l'indice de la colonne correspondante au dernier "match" du groupe précédent. Si nous sommes au premier groupe, x vaut -1.

- **Un vecteur CLIEXT** de dimension 1×S contenant une clique en extension.

	0		$S-1$
$CLIEXT$:		...	

- **Une variable K** qui indique la position de la prochaine entrée dans le vecteur CLIEXT. Lors de l'initialisation K=0, le programme est donc prêt à écrire le premier sommet de la clique dans CLIEXT(0).

  Lorsque K=S, nous avons trouvé une clique maximum, tout le tableau est rempli.

B.1.3 Variables locales du problème

Variables locales de FINDCLI

FINDCLI est une procédure récursive qui cherche une clique maximum à partir de l'extension d'une clique donnée. Elle utilise la variable locale suivante :

- **Une variable J** qui indique avec quel sommet du groupe nous étendons la configuration actuelle de CLIEXT.

B.2 Pseudo-code du programme de recherche de cliques

a. *Procédure : existcandext*

Paramètres d'entrée :
candext : pointeur sur entier;
candidat : pointeur sur entier;
j : pointeur sur entier;
Valeur renvoyée de type entier.
Variables locales :
y, z, ok : entier;

```
DEBUT
  z ← x + 1;
  ok ← FAUX;
  y ← ifingpe[z] + 1;
  SI y = N ALORS
    renvoyer VRAI et sortir;
  SINON
    REPETER
      SI y <> N ET y <> ifingpe[z + 1] + 1 ALORS
        candext[y] ← mat[pointeur sur j * N + y] * candidat[y];
        ok ← ok OU candext[y];
        y ← y + 1;
      SINON
        SI ok = FAUX ALORS
          renvoyer FAUX et sortir;
        SINON
          SI y = N ALORS
            renvoyer VRAI et sortir;
          SINON
            ok ← FAUX;
            z ← z + 1;
          FIN_SI
        FIN_SI
      FIN_SI
    FIN_REPETER
  FIN_SI
FIN
```

b. *Procédure : findcli*

Paramètres d'entrée :
candidat : pointeur sur entier;
Pas de valeur renvoyée.
Variables locales :
j : entier;

```
DEBUT
  j ← ifingpe[x] + 1;
  TANT_QUE j <> N  ET j <> ifingpe[x + 1] + 1 FAIRE
    appel de : traitersommet(adresse de j, candidat);
  FIN_TANT_QUE
FIN
```

c. *Procédure : traitersommet*

Paramètres d'entrée :
j : pointeur sur entier,
candidat : entier.
Pas de valeur retournée.
Variables locales :
candext : pointeur sur entier.

```
DEBUT
  SI candidat[pointeur sur j] = 1 ALORS
    allouer mémoire à candext de dimension N;
    SI existcandext(candext, candidat, j) ALORS
      cliext[k] ← pointeur sur j;
      SI k = S - 1 ALORS
        appel de : memoriserclique;
      SINON
        x ← x + 1;
        k ← k + 1;
        appel de : findcli(candext);
        x ← x - 1;
        k ← k - 1;
      FIN_SI
    FIN_SI
    appel de : free(candext);
  FIN_SI
  pointeur sur j ← (pointeur sur j) + 1;
FIN
```

d. *Procédure : trt_clique*

Paramètres d'entrée :
candidat : pointeur sur entier;
j : pointeur sur entier;
Pas de valeur renvoyée.
Variables locales :
z : entier;

```
DEBUT
  cliext[k] ← pointeur sur j;
  k ← k + 1;
  DE z ← ifingpe[x] + 1 A z < N PAR z++ FAIRE
    candidat[z] ← mat[pointeur de j * N + z];
  FIN_DE
FIN
```

e. *Corps principal*

Pas de paramètres d'entrée.
Variables locales :
j : entier;
candidat : pointeur sur entier;
Pas de valeur renvoyée.

```
DEBUT
  SI NON  read_matrix() ALORS
    SI NON read_igroup() ALORS
      appel de : view_matrix();
      result ← ouvrir fichier RESULTAT.DAT;
      j ← 0;
      appel de : init_all();
      allouer mémoire à candidat, de dimenssion N;
      TANT_QUE j <> ifingpe[x] + 1 FAIRE
        appel de : trt_clique(candidat, adresse de : j);
        appel de : findcli(candidat);
        k ← k - 1;
        j ← j + 1;
      FIN_TANY_QUE
      appel de : fclose(result);
      appel de : free(candidat);
      appel de : free(igroup);
      appel de : free(ifingpe);
      appel de : free(cliext);
    FIN_SI
    appel de : free(mat);
    appel de : free(column);
  FIN_SI
FIN
```

Annexe C

Un exemple du fichier match.log

Le fichier match.log suivant correspond au test numéro 137.

```
MATCH-BOX 3.0 feb 1994                                    24-MAR-94    10:52:09
    5      0      0  Number of sequences
  334      1        G3P1_HUMAN.SW
  331      2        G3P1_YEAST.SW
  403      3        G3PA_MAIZE.SW
  326      4        G3PC_TOBAC.SW
  358      5        G3PG_TRYBB.SW
    7      0      0  w : window length
  200      0 1 5 0  m(b)  m(w) take gaps (0/1) deep(0-10) ELAGAGE(0/1)
    4      0      0   unlike number of identities in 7 pairs
   400  usual cutoff for file   dfk_phys.sco
 ********************************
 ********************************


 Number of initial windows:                        1722
 Number of comparisons performed:                1817464
 Number of matches:                                28703
 Number of best matches:                            6165
 Averaged random noise                  78.0%

 Number of complete matches found:                  8652
 CPU used
SET-UP                 9733
```

```
SORT                    398
TRUTH MATRIX            793
CLIQUE                  12953
TOTAL CPU               23878        23877
Frequence of non match, by sequence
   84   45   56   41   29    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0
```

Références

[1] L. Babel. Finding maximum cliques in arbitrary and in special graph. *Computing*, (n. 46):pp. 321–341, 1991.

[2] F. Badoux. Elaboration d'un algorithme efficace de recherche de cliques. *mémoire en informatique*, 1992-1993.

[3] F. Bafort. Recherche de scores permettant, par l'alignement multiple, de prédire les zones structurellement conservées d'un ensemble de séquences protéiques. *mémoire en biologie*, septembre 1991.

[4] C. Berge. *Graphs and Hypergraphs.* North-Holland, Amsterdam, 1973.

[5] P.H.L. Bovy and E. Stern. *Route Coice: Wayfinding in Transport Networks.* Kluwer Academic Publishers, Dordrecht, 1990.

[6] C. Bron and J. Kerbosch. Finding all cliques of an undirected graph [h]. *Communication of the A.C.M.*, vol. 16(n. 9):pp. 575–577, 1973.

[7] M.B. Brown and W.J. Dixon. University of California Press, ULondon, 1979.

[8] D. Burton. On the inverse shortest path problem. *Doctoral dissertation to obtain a PH.D.Degree in Science*, FNDP Namur, 1993.

[9] C.R. Calladine and H. R. Drew. *Understanding DNA.* Academic press inc., San Diego, 1992.

[10] R. Carraghan and P.M. Pardalos. An exact algorithm for the maximum clique problem. *Operations Research Letters*, (n. 9):pp. 375–382, 1990.

[11] A. Cayley. On the theory of the analytical forms call trees. *Philos.Mag.*, vol. 13:pp. 172–176, 1857.

[12] N. Deo. *Graph Theory with Applications to Engineering and Computer Sciences.* Prentice-Hall, Engewood Cliffs, New Jersey, 1974.

[13] E. Depiereux and E. Feytmans. Simultaneous and multivariate alignment of protein sequences : Correspondence between physicochemical profiles and structurally conserved regions (scr). *Protein Engineering*, vol. 4(n. 6):pp. 603–613, 1991.

[14] E. Depiereux and E. Feytmans. Match-box : A fundamentally new algorithm for the simultaneous alignment of several sequences. *CABIOS*, vol. 8(n. 5):pp. 501–509, 1992.

[15] L. Euler. The konigsgerg bridges. *Sci Amer*, vol. 189:pp. 66–70, 1953.

[16] M.R. Garey and D.S. Johnson. *Computers and Intractability - A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, 1979.

[17] M. Grötschel, L. Lovasz, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization.* Springer-Verlag, Berlin Heidelberg, 1988.

[18] W.H. Hamilton. Account of the icosian calculus. *Proc.Roy.Irish.Acad.*, vol. 6:pp. 415–416, 1853.

[19] D.Y. Handler and P.B. Mirchandani. *Location on Networks : Theory and Algorithms.* The MIT Press, Cambridge, Massachusetts, 1979.

[20] I.D. Hill and P. Griffiths. *Applied Statistics Algorithms.* John Wiley, New York, 1985.

[21] J.-F. Hubermont. Evaluation de l'efficacité et de la spécificité de scores empiriques calculés à partir de l'observation de structures protéiques conservées. *mémoire en biologie*, 1992-1993.

[22] A.K. Kevorkian and J. Snoek. *Decomposition in Large-Scale Systems : Theory and application of Structural Analysis in Partitioning, Disjointing and Constructing Hierarchical Systems.* D.Himmelbaum,North-Holland, Amsterdam, 1973.

[23] A. Kidera, Y. Konishi, M. Oka, and T. Ooi. Statistical analysis of the physical properties of the 20 naturally occuring amino acids. *J.Prot.Chem.*, (n. 4):pp. 23–53, 1985.

[24] G. Kirchhoff. Uber die auflosung der gleichungen, auf welche man bei der untersuchung der linearen verteilung galvanischer strome gefuhrt wird. *Ann.Phys.Chem.*, vol. 72:pp. 497–508, 1847.

[25] A.M. Leontovich. On the optimality of the dayhoff matrix for computing the similarity score between fragments of biological sequences. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 8, 1992.

[26] D.G. Luenberger. *Introduction to Linear and Nonlinear Programming.* Addison-Wesley Reading, MA, 1973.

[27] K. Melhorn. *Data Structure and Algorithms 2 : Graph Algorithms and NP-Completeness.* Springer-Verlag, Berlin Heidelberg-New York Tokyo, 1984.

[28] R.M. Miura. *Some Mathematical Questions in Biology. DNA Sequence Analysis.* American Mathematical Society, USA, 1986.

[29] G. Nolet. *Seismic Tomography.* D.Reidel Publishing Company, Dordrecht, 1987.

[30] Departement of Biomathematics. *B.M.D.P. Statistical Software.* University of California Press, University of California, Los Angeles, 1981.

[31] F.S. Roberts. *Graph Theorie and Its Applications to Problems of Society.* SIAM,CBMSNSF,Regional Conference Series in Applied Mathematics, Philadelphia, Pennsylvania, 1978.

[32] B. Roy. *Algèbre Moderne et Théorie des Graphes.* Dunod, Paris, 1970.

[33] J.P. Sall. Régression. *Guide d'introduction a SAS, SAS Institute Inc.*, pages pp. 67–80, 1992-1993.

[34] T.PH.D. Schlick. *Modeling and Minimization Techniques for Predicting Tree-Dimensional Structures of Large Biological Molecules.* U.M.I., Ann Arbor, USA, 1987.

[35] R. Sedgewick. *Algorithms in C.* Publishing Company, Inc, Addison-Wesley, 1990.

[36] P.H.A. Sneath and R.R. Sokal. *Numerical Taxonomy.* W.H.Freeman, San Francisco, 1973.

[37] R.E. Tarjan and A.E. Trojanowski. Finding a maximum independent set. *SIAM J.Comput.*, vol. 6(n. 3), Sept.1977.