Identification de motifs au sein des structures biologiques arborescentes

Anne-Laure Gaillard

N^o d'ordre : 4381

THÈSE

présentée à

L'UNIVERSITÉ BORDEAUX I

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET INFORMATIQUE

par Anne-Laure GAILLARD

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : Informatique

Identification de motifs au sein des structures biologiques arborescentes

---

Soutenue le : 30 novembre 2011
Après avis de :

M.	Alain DENISE	Professeur	**Rapporteur**
M^{me}	Christine GASPIN ...	Directrice de recherche	**Rapporteur**
M^{me}	Marie-France SAGOT	Directrice de recherche	**Rapporteur**

Devant la Commission d'Examen composée de :

M.	Alain DENISE ...	Professeur	**Rapporteur**
M.	Pascal FERRARO	Maître de Conférences (HDR)	**Directeur de thèse**
M^{me}	Christine GASPIN	Directrice de recherche	**Rapporteur**
M.	Mathieu GIRAUD	Chargé de recherche	**Examinateur invité**
M.	David SHERMAN	Directeur de recherche	**Président**

- 2011 -

Identification de motifs au sein des structures biologiques arborescentes

Anne-Laure GAILLARD

Identification de motifs au sein des structures biologiques arborescentes

Résumé :

Avec l'explosion de la quantité de données biologiques disponible, développer de nouvelles méthodes de traitements efficaces est une problématique majeure en bioinformatique. De nombreuses structures biologiques sont modélisées par des structures arborescentes telles que les structures secondaires d'ARN et l'architecture des plantes. Ces structures contiennent des motifs répétés au sein même de leur structure mais également d'une structure à l'autre. Nous proposons d'exploiter cette propriété fondamentale afin d'améliorer le stockage et le traitement de tels objets.

En nous inspirant du principe de filtres sur les séquences, nous définissons dans cette thèse une méthode de filtrage sur les arborescences ordonnées, permettant de rechercher efficacement dans une base de données, un ensemble d'arborescences ordonnées proches d'une arborescence requête. La méthode se base sur un découpage de l'arborescence en graines et sur une recherche de graines communes entre les structures. Nous définissons et résolvons le problème de chaînage maximum sur des arborescences. Nous proposons dans le cas des structures secondaires d'ARN une définition de graines $(l-d)$ centrées.

Dans un second temps, en nous basant sur des techniques d'instanciations utilisées, par exemple, en infographie et sur la connaissance des propriétés de redondances au sein des structures biologiques, nous présentons une méthode de compression permettant de réduire l'espace mémoire nécessaire pour le stockage d'arborescences non-ordonnées. Après une détermination des redondances, nous utilisons une structure de données plus compacte pour représenter notamment l'architecture de la plante, celle-ci pouvant contenir des informations topologiques mais également géométriques.

Mots-clefs : Bioinformatique, algorithmique, arborescence, architecture des plantes, structure d'ARN.

Discipline : Informatique.

Pattern identification in biological tree structure

Abstract:

The explosion of available biological data urges the need for bioinformatics methods. Many biological structures are modeled by tree structures such as RNA secondary structure and plants architecture. These structures contain repeating units within their structure, but also between different structures. We propose to exploit this fundamental property to improve storage and treatment of such objects.

Following the principle of sequence filtering, we define a filtering method on ordered trees to efficiently retrieve in a database a set of ordered trees close from a query. The method is based on a decomposition of the tree into seeds and the detection of shared seeds between these structures. We define and solve the maximum chaining problem on trees. We propose for RNA secondary structure applications a definition of $(l-d)$ centered seed.

Based on instantiation techniques used for instance in computer graphics and the repetitiveness of biological structures, we present a compression method which reduces the memory space required for plant architecture storage. A more compact data structure is used in order to represent plant architecture. The construction of this data structure require the identification of internal redundancies and taking into account both topological and geometrical informations.

Keywords: Bioinformatic, algorithmic, tree, plant architecture, RNA structure.
Field: Computer Science.

Laboratoire Bordelais de Recherche en Informatique (LaBRI)
Université Bordeaux 1
351 cours de la libération
33405 Talence FRANCE

Table des matières

Table des figures

Liste des tableaux

Liste des algorithmes

Introduction générale

Les *données biologiques*, c'est-à-dire les représentations d'un ensemble d'informations biologiques, sont au cœur de la recherche en bioinformatique [LC03]. Hogeweg [Hog11], à l'origine avec Hesper du terme bioinformatique, a récemment écrit que les données guident la bioinformatique (dans le texte « Data-driven bioinformatics »). En effet, la bioinformatique analyse et interprète ces données au moyen de l'outil informatique dans le but d'apporter de nouvelles connaissances en biologie [RQ04]. Dans un premier temps, les scientifiques collectent des données biologiques de types et de précisions variables. Différentes formes de stockage peuvent alors être envisagées, dépendantes, entre autres, de leurs tailles et des besoins d'accès [HK04]. Enfin, elles sont étudiées, par exemple, afin de créer des modèles, de réaliser des analyses statistiques ou encore de formuler des hypothèses [Fry93]. Dans le but de valider les résultats, une nouvelle phase de collecte peut avoir lieu (Fig. 1).

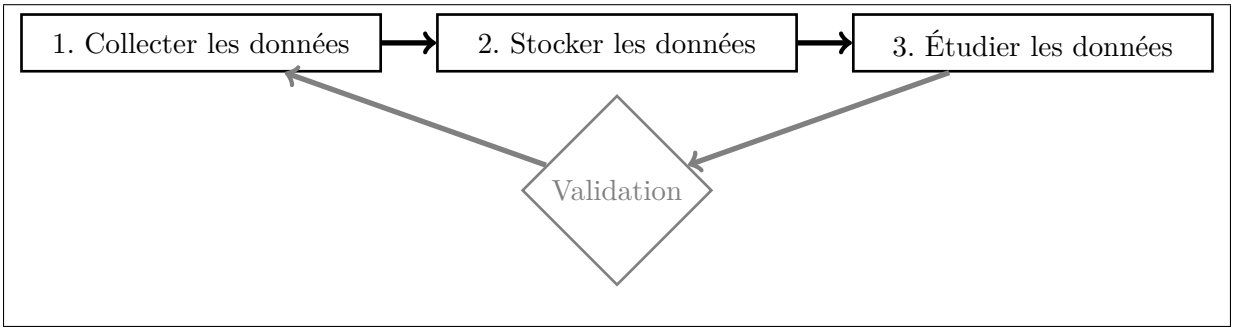


Figure 1 : Schéma d'une méthode scientifique appliquée en bioinformatique.

Collecte des données biologiques

Ces dernières décennies, du niveau de l'écosystème à celui de la molécule, les techniques d'acquisition de données biologiques ont fortement évolué. Tout naturellement, les données biologiques acquises augmentent en qualité et en quantité, nous allons illustrer cette évolution au travers de quelques exemples.

Au niveau de l'écosystème, des approches ont montré que plus de 99% des micro-organismes sur la Terre n'ont pas encore été cultivés en laboratoire [Mar10]. Des travaux ont donc été entrepris, notamment afin de découvrir de nouveaux bactériophages, non pas sur l'individu mais sur l'écosystème. Des milieux entiers ont été étudiés comme les déserts [CCB+06] ou encore les milieux aquatiques [JMN+10] tel que la Seine [LPD+08].

D'un point de vue plus macroscopique au niveau d'un individu, alors que les mesures de plantes ont commencé manuellement, des techniques de mesure à l'aide d'outils tels que les lasers [XGC07, PBF+10] se développent permettant ainsi de collecter plus rapidement des données relatives aux plantes.

Les données concernant les organes et les cellules, unité de base de la structure et de la fonction de la vie [RUC+10], ne font pas exception. Des méthodes permettant d'obtenir des informations quantitatives sur les organes de croissance au niveau cellulaire se sont développées, par exemple en utilisant des techniques d'acquisition et de reconstruction d'images [FDM+10].

Au niveau des données moléculaires, le séquençage du premier génome complet, celui du bactériophage $\Phi X174$, fut réalisé grâce à la technique de Sanger [SAB+77]. Bien que cette technique lui valût le prix Nobel de chimie en 1980, le séquençage de génomes entiers via cette méthode est coûteux en temps et en argent. Au début des années 1990, seulement quelques groupes pouvaient séquencer plus de cent mille bases par an. Suite au projet génome humain [CPJ+98] de nombreux laboratoires ont maintenant la capacité de séquencer dans les 100 millions de bases par an à un coût bien moindre. La Fig. 2 [1] représente le coût du séquençage d'un génome en fonction du temps, passant de plus de 95 millions de dollars en juillet 2001 à moins de 21 mille dollars en janvier 2011.

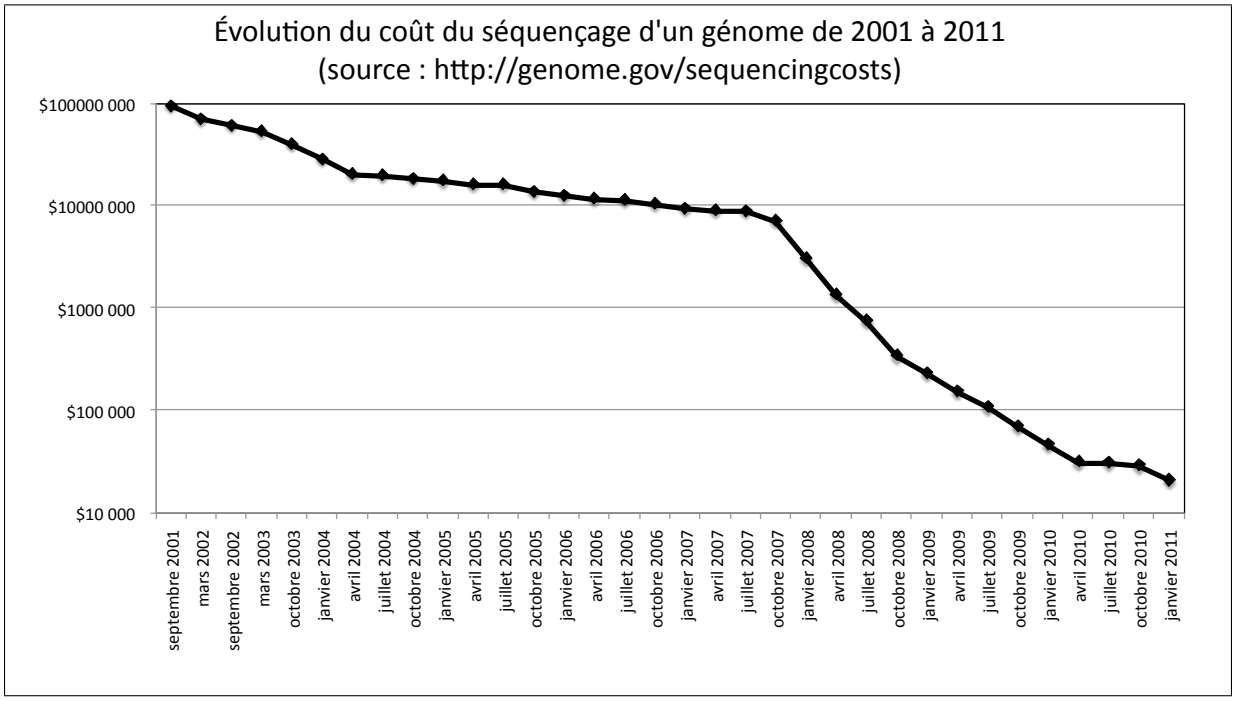


Figure 2 : Graphique représentant la diminution du coût de séquençage d'un génome sur les dix dernières années.

La taille des données biologiques et des bases de données correspondantes ont ainsi explosé. À titre d'exemple, la base de données Rfam [2] comportait en 2003, 25 familles d'ARN [GJBM+03] tandis que la version de juin 2011 en contient 1973. De plus, d'après [GDT+10] la majorité des familles de Rfam vont considérablement augmenter dans un futur proche.

1. Source des valeurs : http ://genome.gov/sequencingcosts
2. Site de la RNA families database of alignments and CMs : http ://www.sanger.ac.uk

Compte tenu de la quantité de données, l'analyse dite in-silico, c'est-à-dire effectuée à l'aide de l'ordinateur, s'est généralisée [Pal00]. Celle-ci étant limitée par deux contraintes que sont l'espace mémoire requis et le temps nécessaire, respectivement liés aux phases de stockage et d'étude. Il est donc devenu primordial de développer des méthodes efficaces du point de vue de ces deux critères.

Bien que la collecte de données biologiques soit un problème fondamental, nous ne traitons pas, dans ce manuscrit, cette phase et supposons que les données sont déjà acquises et disponibles.

Stockage des données

En 1965, Gordon Moore postula que la complexité des semi-conducteurs allait continuer à croître [Moo65]. La conjecture de Moore est souvent interprétée, à tort, comme « la capacité de stockage et la puissance de calcul double tous les 18 mois » [Tuo02]. L'augmentation des capacités de stockage et de calcul augmentent alors que leurs prix diminuent, cependant, l'acquisition des données biologiques est considérablement plus rapide. À titre d'exemple, le coût de séquençage présenté Fig. 2 serait, en suivant une telle loi, en millions de dollars en janvier 2011 alors qu'il est en milliers. Trouver des représentations de stockage efficaces est donc un enjeu clé de la recherche en bioinformatique [RUC+10].

Les structures de données sont des outils d'organisation et après codage, de stockage des données. Elles ont pour but de faciliter l'accès à ces données et leur modification [CLRS04]. Il existe de nombreuses structures de données représentant une ou un ensemble de données : les tableaux, les listes, les séquences, les arbres ou encore les graphes [FGS90]. Étant donné qu'il n'y en a aucune qui réponde à tous les besoins, il est important de connaître les forces et limitations de plusieurs de ces structures. Nous nous intéressons dans ce manuscrit aux données biologiques représentées sous forme d'arborescences.

Structure de données arborescente

Une structure arborescente est une manière de représenter un objet qui se présente sous une forme hiérarchique. De nombreuses structures biologiques sont modélisées par des structures arborescentes tels que les vaisseaux sanguins [BA01], les alvéoles pulmonaires, le système lymphatique, les tissus nerveux [Kal99], les structures secondaires d'ARN [ZS84], l'architecture des plantes [GC98], *etc.*

Nous pouvons noter que les structures arborescentes sont également utilisées pour modéliser des objets d'autres domaines que ceux de la biologie tels que la musique [RIMS03] ou encore les codes sources de programmes [CDR09].

Les données biologiques contiennent du *bruit* dû à leur potentielle nature évolutive [Rea84], à leur environnement [FK50], ce bruit peut également être une conséquence inévitable des techniques de récolte de données [SK95, KY07]. Elles possèdent également des redondances intrinsèques à la nature de l'objet étudié sur différentes échelles, allant des écosystèmes [Wal92] à une échelle moléculaire [NBCS97]. Ainsi les structures arborescentes représentant ces données biologiques héritent de ces deux propriétés qui, nous le verrons, peuvent être prises en compte pour la phase de stockage.

Compression des données

En infographie, par exemple pour la génération d'image de plantes, pour des raisons de limitation de mémoire, l'instanciation d'objets est un concept clé [SSdR03]. Le principe consiste à trouver les éléments d'une scène qui partagent des similarités, par exemple géométriques. Ces éléments sont remplacés par un pointeur vers un unique objet et la transformation géométrique nécessaire pour transformer l'objet initial en l'objet instancié [Sut64]. Cette technique appliquée à des objets de type fractal, apporte un gain significatif [Har92].

Comme nous l'avons dit précédemment, le phénomène de répétitions au sein des individus biologiques est bien connu [Man82]. Des motifs répétés sont facilement notables dans la croissance et la structure de nombreux organismes vivants. Par exemple, une organisation modulaire est un élément essentiel du développement de la structure des plantes [Whi79, RH94]. L'instanciation est d'ailleurs largement utilisée pour la création d'image numérique de plantes [SS00]. Les motifs exacts ou approchés se retrouvent donc au sein d'une même structure ou dans un ensemble de structures. Ces redondances présentes au sein d'une même structure ou d'une base de données peuvent être ainsi utilisées afin de réduire les problèmes de temps ou d'espace ; ce qui est identique n'est stocké ou calculé qu'une seule fois.

Des travaux initiaux ont eu pour objectif la compression sans perte des arborescences non-ordonnées en un DAG [GF10]. Cette thèse s'est orientée vers la diminution de l'espace de stockage utilisé pour les données biologiques arborescentes en utilisant des techniques de compression avec pertes mais également vers la prise en compte d'informations supplémentaires attachées aux nœuds.

Recherche dans des bases de données

Les données une fois stockées peuvent alors être étudiées. L'étude des données biologiques regroupe un champ de recherche très large : l'analyse des génomes [Mou04], la modélisation de plantes [PL90], l'analyse d'images médicales [Eps08] ou encore la reconstruction d'arbres phylogénétiques [Gas07]. De nombreux travaux existent sur le traitement de ces données. Les méthodes développées ne sont cependant pas toutes applicables compte tenu de la masse de données désormais disponible et de nouvelles techniques ont vu le jour, par exemple pour l'étude des données issues des méthodes de séquençage nouvelle génération [MBG+10].

Dans de nombreuses applications, il est primordial de pouvoir comparer un ensemble de données biologiques, par exemple pour l'annotation de génomes ou pour l'étude de l'évolution [HP91]. De nombreuses méthodes permettant de comparer deux structures arborescentes existent mais sont au moins quadratiques [Bil05]. Avec l'augmentation de la quantité des données biologiques arborescentes, il est donc devenu indispensable de les adapter.

Une application courante de la comparaison d'objets est la recherche dans une base de données d'un ensemble d'objets proches d'un objet requête. Il est généralement admis que des objets semblables ne le sont pas par hasard, et peuvent, par exemple, avoir des structures, des fonctions ou encore des ancêtres proches.

Une première approche pour cette recherche est de comparer deux à deux chacun des objets de la base de données avec la requête. Un score de similarité est associé à chaque comparaison. Les objets avec un tel score élevé sont qualifiés de proches. Cette technique a cependant ses limites. Les comparaisons deux à deux sont généralement lentes. Ainsi, lorsque la base de données est grande il n'est plus envisageable de l'utiliser.

Des méthodes alternatives ont été développées sur des objets de type séquences. Par exemple, en ce qui concerne les séquences protéiques ou nucléiques des heuristiques, telles que BLAST [AGM+90] ou FASTA [LP85, PJ88], ont vu le jour ; des régions de séquences similaires entre les deux séquences sont calculées. Le principe de la méthode est alors de chercher, sous certaines contraintes, l'ensemble maximum de régions similaires. Un score est associé à cet ensemble et les séquences avec un tel score élevé sont supposées proches. Cette technique a un impact scientifique majeur et ces travaux ont été cités plusieurs milliers de fois.

Partant de l'approche utilisée dans le cas des séquences [LP85, PJ88, AGM+90], nous proposons dans cette thèse la définition de graines, c'est-à-dire de régions similaires, sur des arborescences ordonnées et donnons une solution pour la détermination de l'ensemble maximum de régions similaires compatibles, appelée chaînage des graines. Ces travaux ont pour applications biologiques principales l'étude des structures secondaires d'ARN. Ils peuvent être appliqués à la recherche d'un ensemble de structures secondaires d'ARN issu d'une base de données telle que Rfam [GJBM+03, GJMM+05, DGT+08, GDT+08, GDT+10] proche d'une structure secondaire d'ARN requête.

Plan du mémoire

Ce document est structuré en cinq chapitres.

Nous présentons, dans le Chapitre 1, les notions de théorie des graphes. En partant du concept de graphes, nous introduisons les arborescences d'un point de vue des structures de données, puis les séquences et terminons par des définitions relatives à la complexité des algorithmes.

La Partie I est consacrée aux arborescences et aux méthodes développées pour les comparer, elle regroupe deux chapitres.

Dans le Chapitre 2, les modélisations d'objets biologiques sous forme d'arborescences sont présentées, deux exemples sont principalement détaillés, les structures secondaires d'ARN et l'architecture des plantes.

Après une présentation des méthodes de comparaison de séquences, le Chapitre 3 explore les algorithmes de comparaison de structures arborescentes.

Le travail effectué dans cette thèse est présenté dans la Partie II qui est constituée de deux chapitres.

Les méthodes de filtrage sur les séquences sont étendues, dans le Chapitre 4, aux arborescences ordonnées. Partant d'une base de données de structures arborescentes nous proposons une méthode permettant de rechercher efficacement l'ensemble des structures proches d'une structure requête. Premièrement, le principe de filtrage sur les séquences ainsi que l'état de l'art sont présentés. Une méthode de chaînage générale entre deux arborescences ordonnées est ensuite donnée. Enfin, nous proposons une définition de graines sur des arborescences ordonnées ayant pour principale application les structures secondaires d'ARN.

Le Chapitre 5 décrit un algorithme de compression de données, basé sur un concept d'infographie nous donnons une méthode de compression des redondances permettant de réduire l'espace mémoire nécessaire pour le stockage d'arborescences non-ordonnées. Nous présentons tout d'abord le principe de compression et une méthode de compression sans perte dont nos méthodes sont des extensions. Nous étudions ensuite une méthode de compression avec perte

des arborescences non-ordonnées. La première étape consiste à trouver les similarités au sein de l'objet. Pour cela, nous nous appuyons sur la comparaison de structures arborescentes. Les objets proches sont alors, dans un second temps, représentés par un même objet. Enfin, nous mettons les techniques pour évaluer la méthode et présentons quelques applications relatives à l'architecture des plantes.

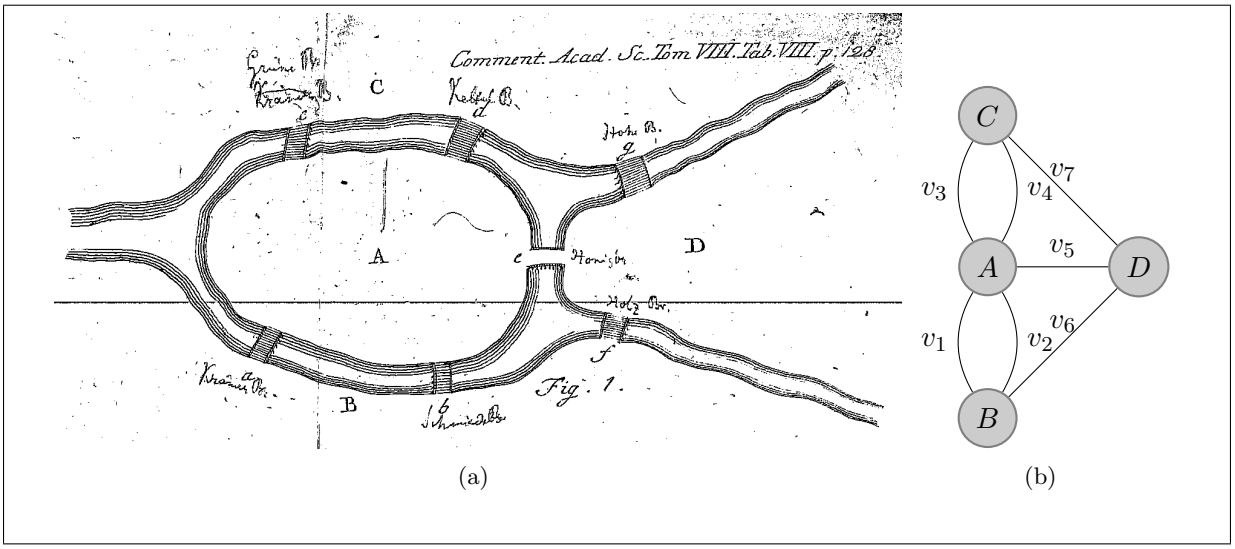
Chapitre 1

Définitions et notations : Concepts de théorie des graphes

Ce chapitre contient les définitions et notations de théorie des graphes, le lecteur familier avec ces notions peut se dispenser de le lire et aller directement à la partie suivante.

La théorie des graphes est un outil de modélisation utilisé dans un grand nombre de disciplines. Tout d'abord, elle prit son essor dans les mathématiques discrètes. En mathématiques, un *graphe* est une représentation abstraite d'un ensemble d'objets dans lequel certains couples d'objets, appelés *nœuds* sont connectés par des liens dits *arêtes* ou *arcs*. Il est généralement admis que le premier résultat formel de la théorie des graphes remonte au milieu du XVIII^{e} siècle, il est attribué au travail de Leonhard Euler, mathématicien et physicien suisse. Son résultat fut présenté à l'Académie de Saint-Pétersbourg puis publié en latin [Eul36]. Le problème traité est celui des sept ponts de Königsberg, les habitants se demandaient s'il était possible, en partant d'un endroit quelconque de la ville, d'effectuer une promenade en traversant tous les ponts sans passer deux fois par le même et de revenir à leur point de départ. Une modélisation sous la forme d'un graphe consiste à représenter les parties de la ville par des nœuds et les ponts par des arêtes. La Fig. 3, en partie extraite de l'article original, illustre la disposition des ponts dans la ville et une modélisation du problème sous forme de graphe. Précisons pour le lecteur curieux qui ne serait pas familier avec le problème, le latin ou encore la théorie des graphes qu'il est impossible de trouver une telle promenade.

Le terme graphe a été introduit en 1878 par James Joseph Sylvester, un mathématicien anglais dans un article intitulé « Chemistry and Algebra »(en français : Chimie et Algèbre) [Syl78] lorsqu'il fit une analogie entre les liens chimiques des molécules et une représentation algébrique. Depuis la théorie des graphes a évolué, elle constitue désormais une discipline à part entière. Elle est un outil de modélisation très utilisé. Par exemple, en biologie, les réseaux métaboliques sont modélisés par des graphes. Extraite de [LDB11], la Fig. 4 représente un réseau métabolique de l'organisme *Saccharomyces cerevisiae*. Dans cette représentation les zones vertes (les nœuds verts) représentent les réactions tandis que les zones blanches (les nœuds blancs) correspondent aux molécules. Pour une revue sur l'étude des réseaux biologiques modélisés par des graphes, se référer à [MV07].



(a) (b)

Figure 3 : Le problème des sept ponts de Königsberg avec un schéma des ponts et une modélisation du problème sous forme de graphe. (a) Schéma extrait de l'article [Eul36] de Leonhard Euler considéré comme le premier résultat de la théorie des graphes. (b) Modélisation du problème sous forme de graphe.

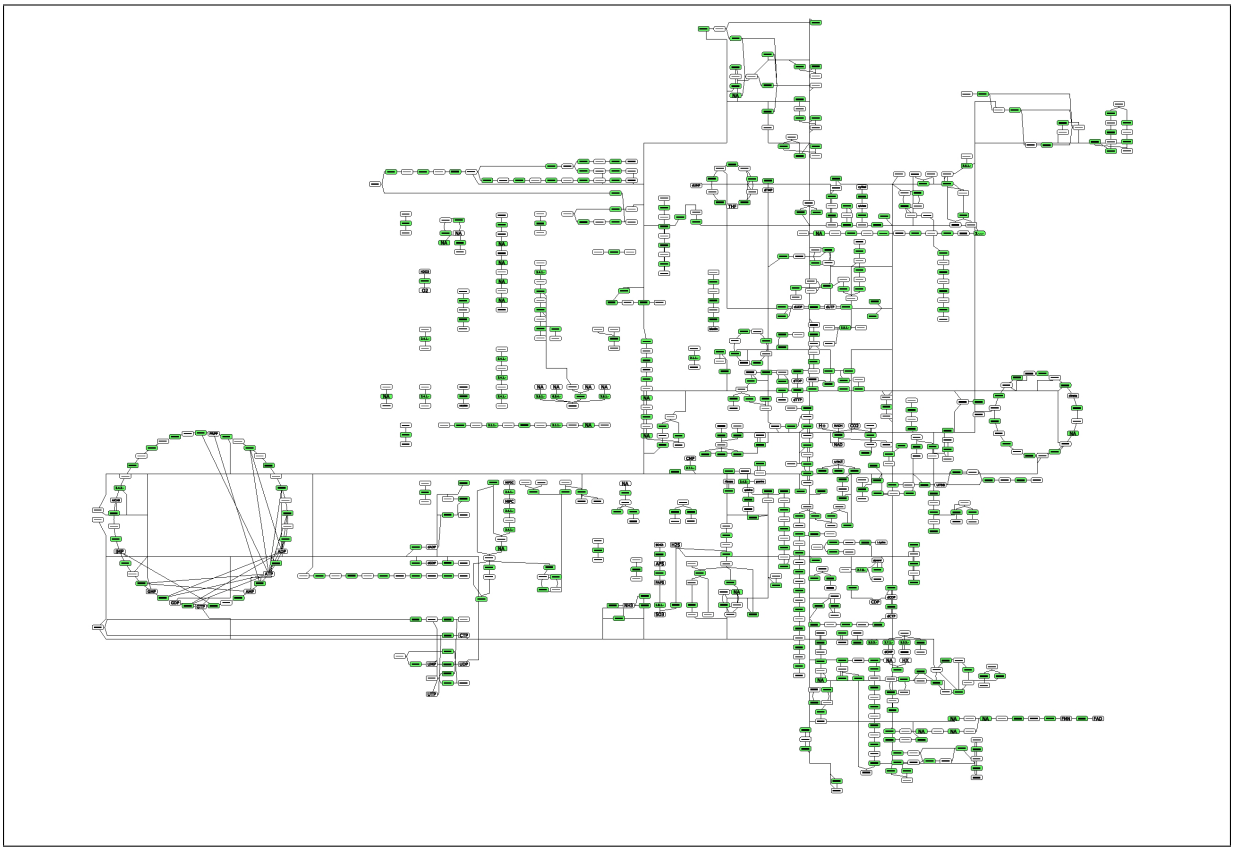


Figure 4 : Modélisation du réseau métabolique de l'organisme *Saccharomyces cerevisiae* (la levure du boulanger), issue de [LDB11].

Nous présentons, dans ce Chapitre 1, le concept de graphe. Ce chapitre présente les concepts de théorie des graphes des plus généraux aux plus spécifiques. Nous donnerons, Section 1.1, des définitions relatives aux graphes. Dans la Section 1.2, nous donnons une définition des arborescences, au cœur des travaux présentés dans ce mémoire. Les séquences également utilisées dans la modélisation sont formalisées Section 1.3. Enfin, dans la Section 1.4, nous présentons la notion de complexité d'un algorithme.

Les concepts présentés ci-après sont indépendants de toute application. Pour une représentation de concepts biologiques sous forme arborescente se référer à la Section 4.3.3, traitant de la représentation des structures secondaires d'ARN et à la Section 2.2 pour celle de l'architecture des plantes.

1.1 Notations et définitions relatives aux graphes

Les définitions suivantes sont en partie issues de [GY03]. Commençons par une définition formelle des termes *graphes*, *nœuds* et *arêtes* (Déf. 1.1).

Définition 1.1 (Graphe) Un graphe G est composé d'un ensemble de *nœuds* (également appelés *sommets*), noté V et d'un ensemble d'*arêtes*, noté E. Chaque arête se compose d'une paire de deux nœuds $\{v_i, v_j\}$ appartenant à V.

La *taille* du graphe est égale au nombre de nœuds dans le graphe (Déf. 1.2).

Définition 1.2 (Taille d'un graphe) Soit $G = (V, E)$ un graphe, notons $|V|$ le nombre d'éléments dans l'ensemble V. La taille du graphe G notée $|G|$ est égale à $|V|$, donc au nombre de nœuds dans le graphe G.

Un exemple de graphe est donné Fig. 5, le graphe contient 7 nœuds et 8 arêtes, l'arête e_3 est un lien entre le nœud v_2 et le nœud v_4, la taille du graphe est de 7.

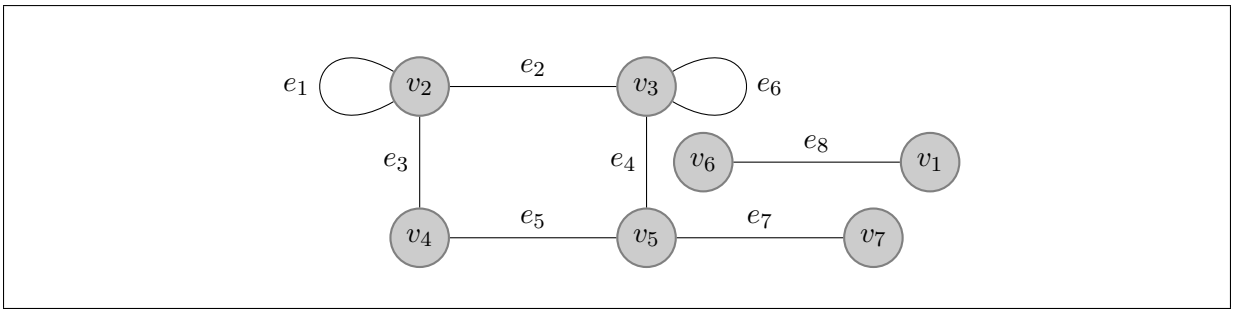


Figure 5 : Un graphe $G = (V, E)$ avec $V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$, $E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8\}$ et $|G| = 7$.

Le lien entre deux nœuds peut-être asymétrique, dans ce cas le lien entre les nœuds v_i et v_j est dans un seul sens, le graphe est alors appelé *graphe orienté* (Déf. 1.3) et les arêtes sont appelées *arcs*. Par convention les arcs sont représentés par des flèches.

Définition 1.3 (Graphe orienté) Un graphe G est composé d'un ensemble de nœuds (également appelés sommets), noté V et d'un ensemble d'arcs, noté E. Chaque arc se compose d'un couple de nœuds (v_i, v_j) appartenant à V.

Dans ce mémoire nous nous intéressons aux graphes orientés. Par la suite, sauf mention particulière, nous considérons $G = (V, E)$ un graphe orienté. Un graphe inclus dans G est un *sous-graphe* (Déf. 1.4).

Définition 1.4 (Sous-graphe) Soit $G' = (V', E')$ un graphe, G' est sous-graphe de G si et seulement si G' est un graphe orienté tel que $V' \subseteq V$ et $E' \subseteq E$.

Un graphe et un de ses sous-graphes sont donnés en exemple Fig. 6.

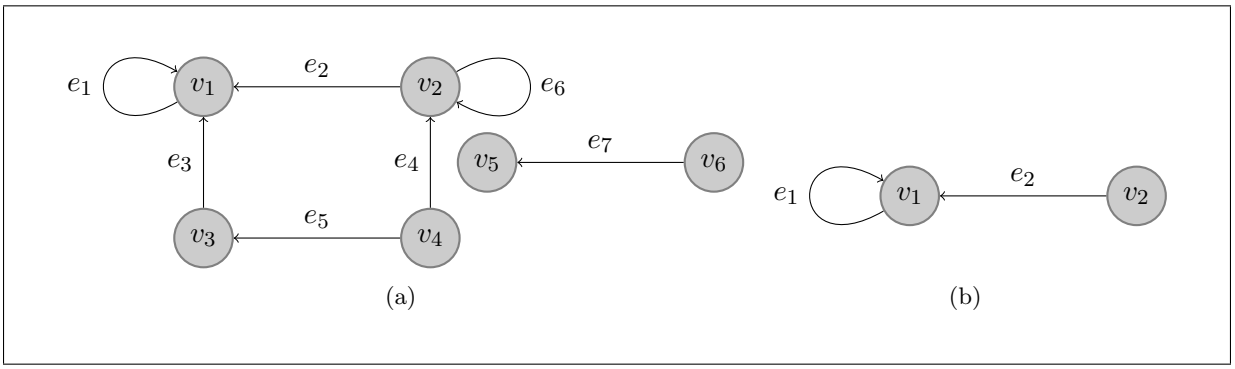


Figure 6 : Un graphe orienté et un de ses sous-graphes. (a) Graphe orienté $G = (V, E)$, e_2 est un arc reliant le nœud v_2 au nœud v_1. (b) Un sous-graphe de G noté $G' = (V', E')$ avec $V' = \{v_1, v_2\}$ et $E' = \{e_1, e_2\}$.

Les nœuds relatifs à un arc sont appelés *extrémités* (Déf. 1.5).

Définition 1.5 (Extrémité) Soit $e = (v_i, v_j)$ un arc, v_i est l'extrémité entrante de l'arc e et v_j l'extrémité sortante de e. Le nœud v_i (*resp.* v_j) est le prédécesseur (*resp.* successeur) du nœud v_j (*resp.* v_i).

Sur le graphe Fig. 6, le nœud v_1 est l'extrémité entrante de l'arc e_1 et le nœud v_2 l'extrémité sortante.

De nombreux concepts sont relatifs aux nœuds. Nous allons présenter ceux qui seront utilisés dans la suite du mémoire, tels que les *degrés entrants* et *sortants* (Déf. 1.6) dont nous pouvons en déduire le *degré* (Déf. 1.7).

Définition 1.6 (Degré entrant et sortant d'un nœud) Le degré entrant (*resp.* sortant) d'un nœud $v \in V$ est le nombre d'arcs $e \in E$ dont l'extrémité entrante (*resp.* sortante) est v. Il est noté $deg_G^-(v)$ (*resp.* $deg_G^+(v)$).

Définition 1.7 (Degré d'un nœud) Le degré d'un nœud $v \in V$ est noté $deg_G(v)$ et est défini comme suit : $deg_G(v) = deg_G^-(v) + deg_G^+(v)$.

Sur la Fig. 6(a), le degré entrant du nœud v_2 est 2, son degré sortant également, son degré est donc 4.

Un nœud possède un *voisinage* (Déf. 1.8) qui nous permet de définir les *nœuds isolés* (Déf. 1.9) et les *nœuds jumeaux* (Déf. 1.10).

Définition 1.8 (Voisinage d'un nœud) Le voisinage du nœud $v_i \in V$ est l'ensemble $N(v_i) = \{v_j | (v_i, v_j) \in E \text{ ou } (v_j, v_i) \in E\}$. Comme pour le degré, le voisinage entrant du nœud v_i est l'ensemble $N^-(v_i) = \{v_j | (v_j, v_i) \in E\}$ et le voisinage sortant $N^+(v_i) = \{v_j | (v_i, v_j) \in E\}$.

Remarque 1 Nous notons que $N^+(v_i) \cup N^-(v_i) = N(v_i)$.

Le voisinage du nœud v_3 sur le graphe G de la Fig. 6(a) est $N(v_3) = \{v_1, v_4\}$, son voisinage entrant $N^-(v_3) = \{v_4\}$ et son voisinage sortant $N^+(v_3) = \{v_1\}$.

Définition 1.9 (Nœud isolé) Un nœud v tel que $N(v) = \varnothing$ est appelé nœud isolé.

Définition 1.10 (Nœuds jumeaux) Deux nœuds v_i et v_j tels que $N^+(v_i) \backslash v_j = N^+(v_j) \backslash v_i$ sont dits jumeaux.

Une illustration de ces deux définitions est donnée Fig. 7, le nœud v_5 a pour voisinage $N(v_5) = \varnothing$, il est donc isolé. Les nœuds v_2 et v_4 ont respectivement pour voisinage sortant $N^+(v_2) = \{v_1\}$ et $N^+(v_4) = \{v_1, v_2\}$, les deux ensembles $N^+(v_2) \backslash v_4$ et $N^+(v_4) \backslash v_3$ sont égaux, ces nœuds sont donc jumeaux.

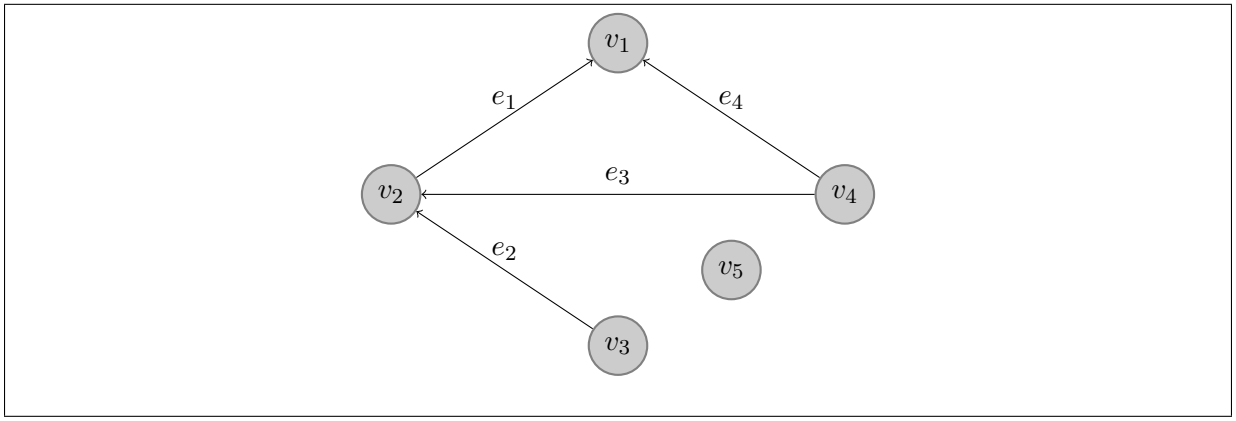


Figure 7 : Un graphe orienté $G = (V, E)$, le nœud v_5 est isolé et les nœuds v_2 et v_4 sont jumeaux.

Un *chemin* (Déf. 1.11) permet de relier un ensemble de nœuds avec des arcs.

Définition 1.11 (Chemin) Un chemin dans G est une suite alternée de nœuds dans V et d'arcs dans E de la forme $(v_0,\ e_1,\ v_1,\ e_2,\ v_2,\ \ldots,\ e_n,\ v_n)$ telle que pour tout arc e_i appartenant au chemin, v_i est l'extrémité sortante de e_i mais également l'extrémité entrante de v_{i+1}. La *longueur* du chemin est le nombre d'arcs qui le composent.

Définition 1.12 (Boucle) Une boucle est un arc $e = (v_i, v_j) \in E$ dans lequel $v_i = v_j$.

Définition 1.13 (Arête parallèle) Soient deux arêtes $e_1 = (x_1, y_1)$ et $a_2 = (x_2, y_2)$, elles sont dites parallèles si et seulement si $x_1 = x_2$ et $y_1 = y_2$.

Définition 1.14 (Arête multiple) Ensemble d'arêtes parallèles relatives à un couple de nœuds.

Sur la Fig. 8 nous pouvons observer une boucle, l'arc e_7 sur le nœud v_5. Les arêtes parallèles sont $\{e_4, e_5\}$, $\{e_3, e_5\}$ et $\{e_3, e_4\}$. L'ensemble $\{e_3, e_4, e_5\}$ est un ensemble d'arêtes multiples relatif aux nœuds v_2 et v_4.

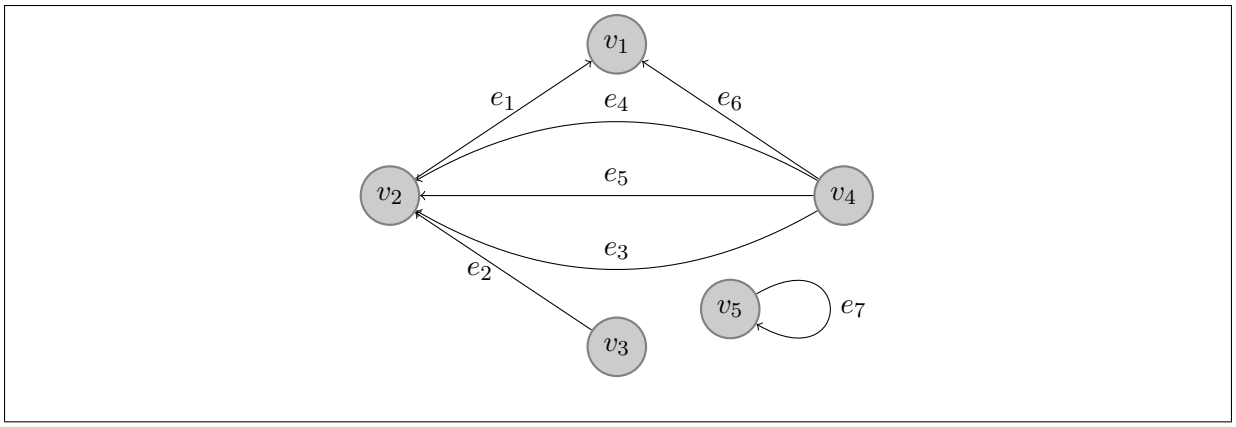


Figure 8 : Le graphe G possède l'arc e_7 est une boucle, les arcs e_4 et e_5 sont parallèles, l'ensemble $\{e_3, e_4, e_5\}$ constitue les arêtes multiples entre les nœuds v_2 et v_4.

Définition 1.15 (Graphe connexe) Un graphe G est dit connexe si et seulement si quels que soient les nœuds v_1 et v_2 de l'ensemble de nœuds, il existe un chemin de v_1 à v_2 ou de v_2 à v_1.

Remarque 2 Un graphe connexe ne peut pas contenir de nœud isolé.

Nous pouvons remarquer que les graphes Fig. 6(a), Fig. 7 et Fig. 8 ne sont pas connexes, dans les trois cas il n'existe pas de chemin du nœud v_5 au nœud v_1. En revanche, le sous-graphe Fig. 6(b) est connexe.

Définition 1.16 (Graphe planaire) S'il existe un moyen de dessiner un graphe dans un plan sans croiser deux arêtes, le graphe est dit planaire.

Un exemple de graphe non planaire est donné Fig. 9(a), les autres graphes précédemment illustrés sont planaires.

Une notion celle de *cycle* (Déf. 1.17) permet de définir un autre type de graphe particulier les *graphes acycliques orientés* (Déf. 1.18).

Définition 1.17 (Cycle) Un graphe G contient un cycle si et seulement s'il existe un nœud v appartenant à l'ensemble des nœuds, tel qu'il existe un chemin de longueur strictement positif de v à v.

La Fig. 9(a) possède plusieurs cycles par exemple les cycles $[v_1,\ e_7,\ v_5,\ e_8,\ v_2,\ e_1,\ v_1]$ et $[v_1,\ e_7,\ v_5,\ e_{10},\ v_4,\ e_6,\ v_3,\ e_3,\ v_2,\ e_1,\ v_1]$.

Définition 1.18 (Graphe acyclique orienté) Un graphe qui ne contient pas de cycle orienté est appelé graphe acyclique orienté. Ce type de graphe est noté *DAG* (« *directed acyclic graph* » en anglais).

Le graphe Fig. 7 est un graphe acyclique orienté.

Des informations supplémentaires peuvent être attachées aux nœuds et arcs d'un graphe, il est alors appelé *graphe étiqueté* (Déf. 1.19).

Définition 1.19 (Graphe étiqueté) Soient Σ un alphabet, c'est-à-dire un ensemble de symboles, et $G = (V, E)$ un graphe, s'il existe une fonction injective λ_V (*resp.* λ_E) telle que $\lambda_V : V \to \Sigma$ (*resp.* $\lambda_E : E \to \Sigma$) alors le graphe est dit étiqueté.

Remarque 3 L'ensemble λ_V (*resp.* λ_E) peut être vide.

Définition 1.20 (Isomorphe) Deux graphes $G = (V, E)$ et $H = (V', E')$ sont isomorphes si et seulement s'il existe une fonction bijective $f : V \to V'$ telle que pour tout $v_1, v_2 \in V$, $(v_1, v_2) \in E$ si seulement si $(f(v_1), f(v_2)) \in E'$. L'isomorphisme défini une relation d'équivalence entre les nœuds de ces deux graphes, nous noterons $G \equiv H$.

Remarque 4 Si deux graphes sont isomorphes leurs nombres de nœuds et d'arêtes doivent être identiques.

Les deux graphes G et H de la Fig. 9 sont isomorphes.

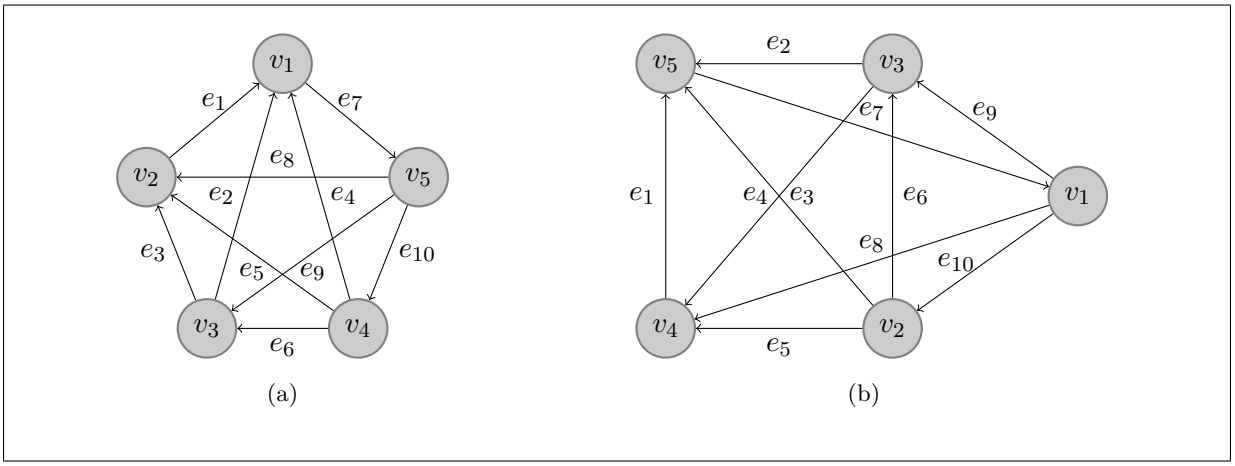


Figure 9 : Deux graphes isomorphes. (a) Un graphe G non planaire avec 5 nœuds. (b) Un graphe H isomorphe au graphe G.

Lorsque deux graphes sont isomorphes nous pouvons remarquer que les étiquettes des nœuds et des arcs peuvent être différentes. Lorsque les étiquettes sont prises en compte on parle d'*isomorphisme avec étiquettes* (Déf. 1.21).

Définition 1.21 (Isomorphisme avec étiquettes) Deux graphes étiquetés $G = (V, E)$, $H = (V', E')$ et leurs étiquettes λ_V, λ'_V, λ_E et λ'_E, G et H sont isomorphes avec étiquette si et seulement s'il existe une fonction bijective $f : V \to V'$ telle que pour tout $v_1 \in V$, $\lambda_V(v_1) = \lambda'_V(f(v_1))$ et pour tout $v_1, v_2 \in V$, $(v_1, v_2) \in E$ si et seulement si $(f(v_1), f(v_2)) \in E'$ et $\lambda_E(v_1, v_2) = \lambda_E(f(v_1), f(v_2))$.

Dans le cas où l'on considère que les noms des nœuds et des arcs sont des étiquettes alors les deux graphes G et H de la Fig. 9 sont ne sont pas isomorphes avec étiquettes.

Il existe de nombreux graphes particuliers tels que les graphes *simples* (Déf. 1.22), *connexes* (Déf. 1.15) et *planaires* (Déf. 1.16).

Définition 1.22 (Graphe simple) Un graphe est dit simple si et seulement s'il ne contient ni boucles, ni arêtes multiples.

Ainsi le graphe Fig. 8 n'est pas simple alors que les graphes Fig. 7 et Fig. 9 sont simples.

1.2 Arborescences

Au milieu du XIX^{e} siècle, Arthur Cayley, mathématicien anglais, s'intéressa à des graphes particuliers sans cycles, les arbres. L'étude de ces structures fût continuée par George Pólya, mathématicien hongrois, au début des années 1900. Ils présentaient tous deux des applications à la chimie. De nos jours, les arbres sont utilisés pour modéliser de nombreux phénomènes ou problèmes, par exemple pour modéliser les structures secondaires d'ARN et les architectures des plantes. Le vocabulaire de théorie des graphes sur les arbres est d'ailleurs très inspiré du vocabulaire biologique.

Avant de donner la définition d'un *arbre* (Déf. 1.24) et d'une *arborescence* (Déf. 1.26), nous allons voir les définitions d'une *forêt non-orientée* (Déf. 1.23) et d'une *forêt orientée* (Déf. 1.25).

Définition 1.23 (Forêt non-orientée) Un graphe G non-orienté est une forêt si et seulement si G ne contient aucun cycle.

Définition 1.24 (Arbre) Un arbre est une forêt connexe.

Remarque 5 Une forêt peut-être vue comme un ensemble d'arbres.

Définition 1.25 (Forêt orientée) Un graphe G orienté est une forêt si et seulement si G ne contient aucun cycle et que tous les nœuds de G sont de degré entrant au plus 1.

Remarque 6 Par la suite, nous utiliserons le terme forêt pour référer à une forêt orientée.

Définition 1.26 (Arborescences) Une arborescence T est un arbre orienté tel que tous les nœuds de T sont de degré entrant 1, à l'exception d'un nœud de degré entrant 0.

Des nœuds particuliers sont présents dans les arborescences tels que la *racine* (Déf. 1.27), les *nœuds internes* (Déf. 1.28) et les *feuilles* (Déf. 1.29).

Définition 1.27 (Racine) La racine de l'arborescence est le nœud de l'arborescence qui n'est extrémité sortante d'aucun arc de V.

Définition 1.28 (Nœud interne) Soit v un nœud d'une arborescence, si $deg^{+}(v) > 0$ alors v est un nœud interne.

Définition 1.29 (Feuille) Soit v un nœud d'une arborescence, si $deg^{+}(v) = 0$ alors v est une feuille.

Définition 1.30 (Profondeur) La profondeur d'un nœud v est la longueur du chemin de la racine au nœud v.

Ainsi sur la Fig. 10(a) le nœud v_1 est la racine, les nœuds v_2 et v_5 sont des nœuds internes, enfin les nœuds v_3, v_4, v_6, v_7 et v_8 sont des feuilles. La profondeur du nœud v_7 est de 2.

Définition 1.31 (Parent) Soit v_i un nœud d'une arborescence qui est l'extrémité sortante d'un arc e. Le parent de v_i est le nœud v_j tel que v_j est extrémité entrante de e.

Définition 1.32 (Enfant) Soit v_i un nœud d'une arborescence, les enfants de v_i sont les nœuds qui ont pour parent v_i, ils sont notés $enf(v_i)$.

Définition 1.33 (Fratrie) Deux nœuds d'une arborescence sont dans la même fratrie s'ils ont un même nœud parent.

Sur l'arborescence Fig. 10(a) le nœud v_2 est parent du nœud v_3, le nœud v_4 est un enfant du nœud v_2 et les nœuds v_6, v_7 et v_8 sont dans la même fratrie.

Définition 1.34 (Sous-arborescence) Soit T une arborescence, une sous-arborescence est un sous-graphe de T connexe. La sous-arborescence complète notée $T[v]$ avec $v \in V$ est la sous-arborescence maximale enracinée en v, c'est-à-dire la sous-arborescence contenant v et l'ensemble de ces descendants.

Définition 1.35 (Sous-forêt) Soit T une arborescence, une sous-forêt est un sous-graphe de T noté $F[v] = (V', E')$ avec $v \in V$ telle que V' est l'ensemble des nœuds de $T[v] \backslash \{v\}$ et E' l'ensemble des arcs de $T[v]$ n'ayant pas pour extrémité entrante ou sortante v.

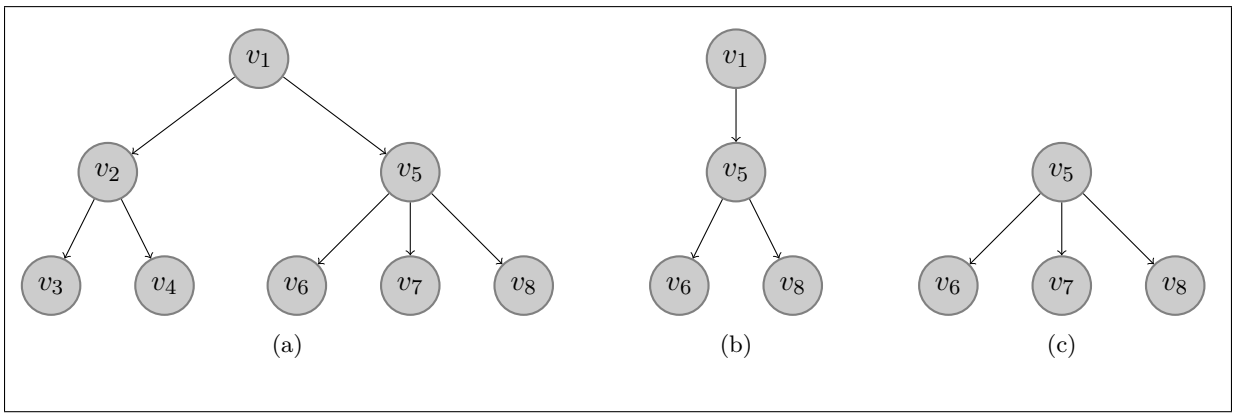


Figure 10 : Une arborescence, une de ces sous-arborescences et une de ces sous-arborescences complètes. (a) Représentation d'une arborescence T avec pour racine le nœud v_1, le nœud v_2 est parent du nœud v_3, le nœud v_4 est une enfant du nœud v_2, les nœuds v_6, v_7 et v_8 sont dans la même fratrie, les nœuds v_2 et v_5 sont des nœuds internes, enfin les nœuds v_3, c_4, v_6, v_7 et v_8 sont des feuilles. (b) Une des sous-arborescences de T. (c) Une de ses sous-arborescences complètes, qui est enracinée en v_5.

Soit l'arborescence Fig. 10(a), une sous-arborescence est donnée Fig. 10(b) et une sous-arborescente complète Fig. 10(c).

Afin de définir les arborescences ordonnées, semi-ordonnées et non-ordonnées, nous devons définir les relations d'ordre total et de semi-ordre total.

Définition 1.36 (Relation binaire) Une relation binaire $\mathcal{R}$ d'un ensemble E vers un ensemble F est définie par un sous-ensemble G de $E \times F$. Pour tout couple $(x, y) \in G$, x est en relation avec y et est noté $x\mathcal{R}y$.

Définition 1.37 (Relation de semi-ordre total) Une relation binaire $\preceq$ est une relation de semi-ordre total si pour tous x, y et z éléments de l'ensemble E :

- $x \preceq x$ (réflexivité),
- $x \preceq y$ et $y \preceq z \Rightarrow x \preceq z$ (transitivité),
- $\forall (x, y) \in E^2$ $x \mathcal{R} y$ ou $y \mathcal{R} x$ (totale).

Définition 1.38 (Relation d'ordre totale) Une relation binaire $\leq$ est une relation d'ordre totale si pour tous x et y éléments de l'ensemble E :

- $\leq$ est une relation de semi-ordre total,
- $x \leq y$ et $y \leq x \Rightarrow x = y$ (antisymétrie).

Définition 1.39 (Arborescence non-ordonnée) Une arborescence non-ordonnée est une arborescence dans laquelle pour tout nœud il n'existe pas de relation d'ordre sur l'ensemble des enfants [ZSS92].

Définition 1.40 (Arborescence ordonnée) Une arborescence ordonnée est une arborescence dans laquelle il existe une relation d'ordre total sur les nœuds de l'arborescences [ZS89].

Définition 1.41 (Arborescence semi-ordonnée) Une arborescence semi-ordonnée est une arborescence dans laquelle il existe une relation de semi-ordre total sur les nœuds de l'arborescences [Oua07], c'est-à-dire muni d'une relation d'ordre pour certains couples.

Remarque 7 Un isomorphisme sur des arborescences ordonnées ou semi-ordonnées doit respecter la relation d'ordre sur les nœuds.

Dans le cas des arborescences non-ordonnées, les arborescences T_1 et T_2 de la Fig. 11 sont isomorphes. Si les nœuds sont ordonnés selon leur position graphique avec $v_i < v_j$ si v_i est situé à gauche de v_j alors ces arborescences ne le sont pas isomorphes dans le cas ordonné.

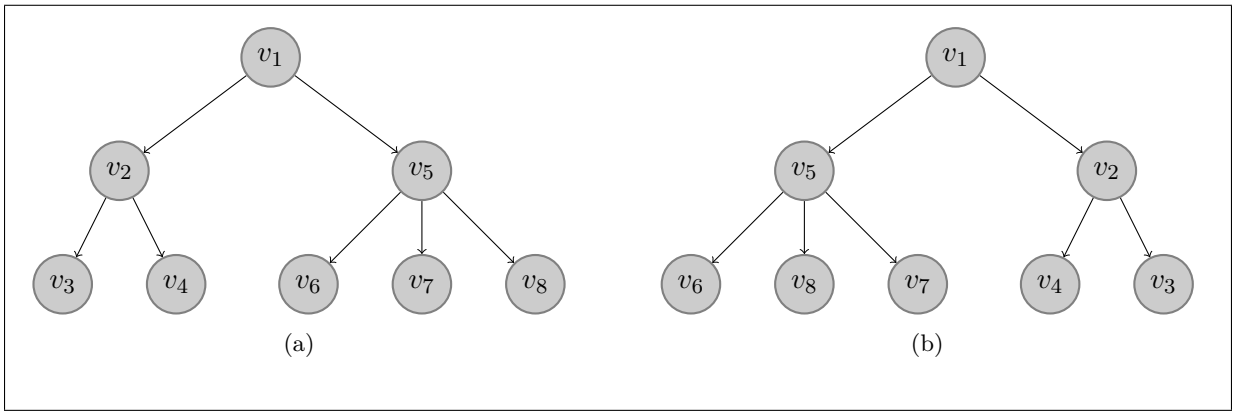


Figure 11 : Dans le cas non-ordonné les arborescences T_1 et T_2 sont isomorphes, pas dans le cas ordonné. (a) Arborescence T_1. (b) Arborescence T_2

Afin d'accéder aux informations contenues dans l'arborescence, il existe plusieurs manières de le parcourir. Le *parcours en profondeur* consiste en un parcours récursif des nœuds.

Nous allons présenter un ordre sur ce parcours. Dans l'ordre postfixe les descendants sont parcourus avant leurs parents en suivant l'Algorithme 1.

Remarque 8 Dans l'ordre préfixe chaque nœud est visité avant ses descendants.

Algorithme 1 *ParcoursPostFixe(A)* : Algorithme de parcours postfixe d'une arborescence.

Entrées : A une arborescence ayant pour racine r.
Sorties : L'ordre postfixe de A.

1: **Si** $A = \varnothing$ **Alors**
2: **Retourner** λ
3: **Sinon**
4: **Pour tout** s enfant de r **Faire**
5: $ParcoursPostFixe(A[s])$
6: **Fin pour**
7: **Retourner** r
8: **Fin si**

Le parcours postfixe de l'arborescence T_1, présentée Fig. 11(a), donne $(v_3, v_4, v_2, v_6, v_7, v_8, v_5, v_1)$.

Les *arbres binaires* sont des arbres particuliers dans lesquels chaque nœud possède au plus deux enfants, qui sont nommés enfant gauche et enfant droit.

Les *arbres binaires de recherche* Fig. 1.42 permettent une recherche rapide des valeurs associées aux nœuds (voir Algorithme 2).

Définition 1.42 (Arbre binaire de recherche) Un arbre binaire de recherche, abrégé en ABR est un arbre binaire ordonné pour lequel à chaque nœud est assigné une clé. La clé associée à chaque nœud v est plus grande (*resp.* plus petite) que les clés de l'ensemble des nœuds présents dans le sous-arbre enraciné en v_g (*resp.* v_d). Ce type d'arbre est également noté BST (« *binary search tree* » en anglais).

Algorithme 2 *RechercheABR(A, c)* : Algorithme de recherche dans un arbre binaire.

Entrées : A un arbre et c une clé cible.
Sorties : Un nœud s de A ayant pour clé c ou NUL si c n'est pas trouvé.

1: $s \leftarrow$ racine de A
2: **Tant que** ($s \neq$ NUL) et ($c \neq$ clé associée à s) **Faire**
3: **Si** $c >$ clé associée à s **Alors**
4: $s \leftarrow$ enfant droit de s
5: **Sinon**
6: $s \leftarrow$ enfant gauche de s
7: **Fin si**
8: **Fin tant que**
9: **Retourner** s

1.3 Séquences

La séquence peut être vue comme une arborescence particulière dans laquelle les nœuds sont représentés comme des caractères. De nombreux objets sont modélisés par des *séquences* (Déf. 1.43), nous pouvons citer les séquences pour les correcteurs orthographiques ou encore les séquences de bases d'ADN. Une séquence particulière est la *séquence vide* (Déf. 1.44).

Définition 1.43 (Séquence) Une séquence de *longueur* n est une suite de caractères $S = S[0], S[1], \ldots, S[n-2], S[n-1]$ tels que $\forall i \in [\![0, n-1]\!], S[i] \in \Sigma$ avec Σ un alphabet. La *taille* de la séquence est notée $|S|$.

Définition 1.44 (Séquence vide) Si et seulement si la *longueur* de S est de 0, la séquence est dite vide, elle est notée λ.

Remarque 9 Soit S une séquence, nous notons $S[i, j]$ avec $0 \leq i \leq j \leq |S|$ une sous-séquence de S contenant la suite de caractères $S[i], \ldots, S[j]$.

Il est possible de représenter certaines arborescences ordonnées par des *séquences arc-annotées* (Déf. 1.45) introduites par [Eva99]. La Fig. 12 présente (a) une arborescence, (b) une représentation possible de l'arborescence par une séquence arc-annotée.

Définition 1.45 (Séquence arc-annotée) Une séquence arc-annotée de taille n sur un alphabet Σ est un couple (S, P) tel que S est une séquence de taille n sur Σ et P un ensemble de couples tel que pour chaque couple $(i, j) \in P$ un arc relie les lettres $S[i]$ et $S[j]$ de la chaîne S avec $0 \leq i \leq j \leq n-1$.

Remarque 10 Le $i^{ème}$ caractère de S est noté $S[i-1]$.

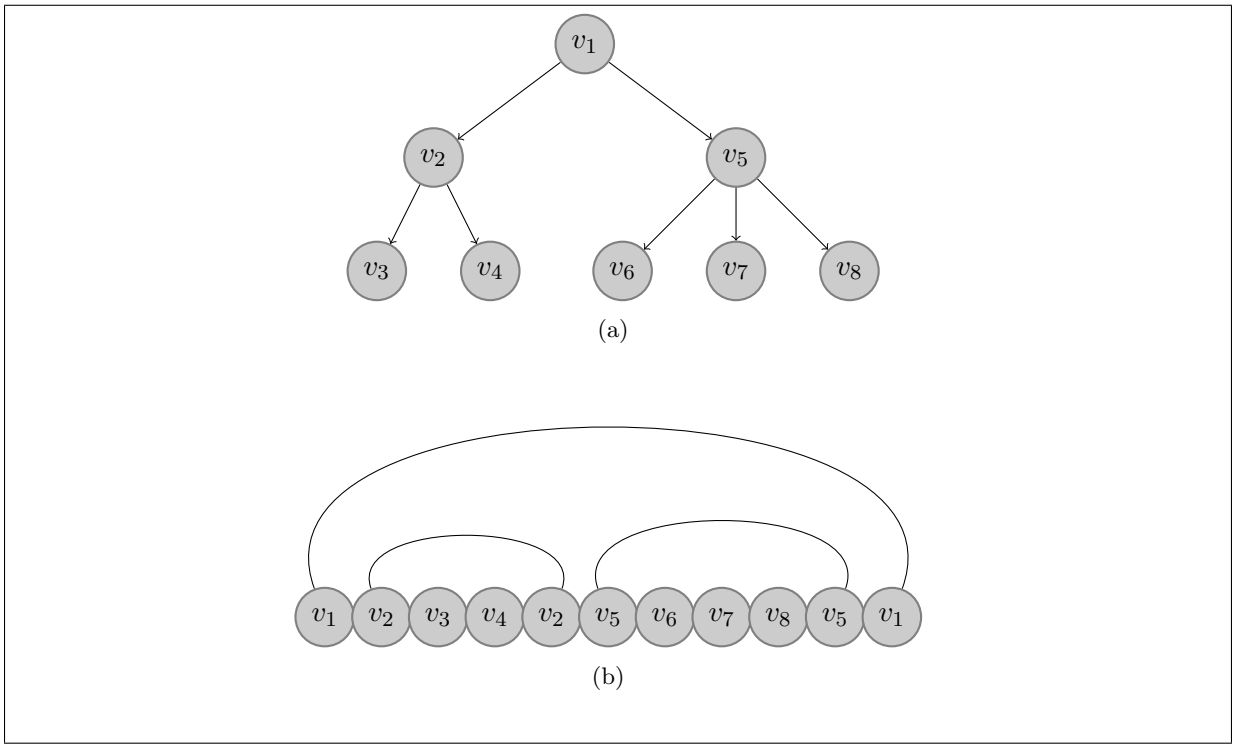


Figure 12 : Représentation sous forme d'une séquence arc-annotée de l'arborescence A. (a) Arborescence A. (b) Séquence arc-annotée correspondant à l'arborescence A.

1.4 Définitions relatives aux algorithmes

Le terme *algorithme* est apparu bien avant les ordinateurs, vers le IX^{e} siècle. Un algorithme désigne un processus répondant à une question en suivant une suite d'opérations. Une mesure d'évaluation des algorithmes est sa complexité. Le modèle de calcul basé sur les machines de Turing est l'un des plus utilisés. Une machine de Turing, introduite par Turing [Tur37], est un automate abstrait disposant de mémoire infinie. À partir de cela nous pouvons définir le calcul des *complexités en temps* (Déf. 1.46) et en *espace* (Déf. 1.47) [HS65, CLRS04].

Définition 1.46 (Complexité en temps) La complexité en temps d'un algorithme est une fonction $f : \mathbb{N} \to \mathbb{N}$ telle que $f(n)$ est le nombre d'opérations élémentaires nécessaires à réaliser sur une machine de Turing pour une entrée quelconque de taille n.

Définition 1.47 (Complexité en espace) La complexité en espace d'un algorithme est une fonction $f : \mathbb{N} \to \mathbb{N}$ telle que $f(n)$ est le nombre cellules différentes de la bande qui sont utilisées par une machine de Turing pour une entrée quelconque de taille n.

Dans cette thèse, nous ne nous intéresserons qu'à la borne supérieure asymptotique du temps d'exécution ou de l'espace utilisé notée $O(f(n))$ (Déf. 1.48).

Définition 1.48 (Grand O) Soit une fonction $f : \mathbb{N} \to \mathbb{R}$, l'ensemble $O(f)$ est défini comme suit : $O(f) : \{g : \mathbb{N} \to \mathbb{R} \mid \exists\ c > 0\ \exists\ n_0 \in \mathbb{N}\ \forall n \geq n_0 : g(n) \leq c \times f(n)\}$.

Remarque 11 Une complexité en $O(1)$ est dite constante, en $O(n)$ linéaire et en $O(n^2)$ quadratique.

Les problèmes peuvent être classés selon leurs difficultés. La classe de complexité **NP** (« *non-deterministic polynomial* » en anglais) caractérise les problèmes pour lesquels la réponse, vérifiable en un temps polynomial, est « oui » ou « non ». Les problèmes inclus dans **NP** pour lesquels il existe un algorithme déterministe en $O(n^k)$ où k est une constante sont dits *polynomiaux*.

Nous avons présenté dans ce chapitre la structure de données principalement concernée par ce manuscrit que sont les structures arborescentes mais également deux autres structures qui sont les graphes et les séquences. Nous allons voir par la suite la modélisation de deux objets biologiques sous forme d'arborescences.

Première partie

Comparaisons de structures arborescences

Introduction

L'étude de l'ensemble des données biologiques étant très large, dans cette thèse, nous nous focalisons sur deux objets biologiques présents à deux échelles différentes que sont l'échelle moléculaire avec la structure secondaire d'ARN et l'échelle macroscopique de l'individu au travers de l'architecture des plantes.

Un premier exemple de structures biologiques modélisées par des structures arborescentes est la structure secondaire d'ARN [ZS84, Sha88]. Miescher remarqua en 1868 une substance avec des propriétés inhabituelles, qu'il appela « nucléine » [Dah08]. Il réalisa ainsi la première purification d'acide désoxyribonucléique, abrégé en ADN, cette découverte permit de faire une avancée importante dans la connaissance du vivant. Ce terme est d'ailleurs, deux siècles plus tard, passé dans le langage courant. L'acide ribonucléique ou ARN est une molécule chimiquement très proche de l'ADN. Mais il n'en demeure pas moins qu'il est fondamental pour la compréhension de la vie. Selon le dogme central de la biologie, l'ADN est le support de l'information génétique, il est transcrit en ARN qui est traduit en protéine. Cette dernière ayant un rôle fonctionnel dans la cellule. La fonction de l'ARN ne se limite cependant pas à cela. En effet de nombreux travaux ont permis de mettre en évidence son rôle en tant que composant essentiel de la cellule. Les ARN messagers et de transferts ont été découverts dans les années soixante [HSS+58, Cri58, BJM61]. En 1986, Gilbert [Gil86] a posé l'hypothèse selon laquelle l'ARN serait apparu dans l'évolution avant l'ADN et les protéines. Il a fait naître la théorie dite du « RNA World » (en français : Monde à ARN). En 1989, Tom Cech et Sidney Altman [CB86, Alt89] se sont vus remettre le prix Nobel de chimie pour leur découverte des ribozymes. Ces derniers sont des ARN capables de catalyser des réactions. L'hypothèse du monde à ARN s'est alors renforcée. En 2006, le prix Nobel de médecine a été attribué à Andrew Fire et Craig C. Mello [FXM+98], pour la découverte d'un ARN interférant avec un ARN messager spécifique conduisant à sa dégradation et à la diminution de sa traduction en protéine. Encore plus récemment, le prix Nobel de chimie a été attribué en 2009 à Venkatraman Ramakrishnan, Thomas A. Steitz et Ada E. Yonath pour leurs études de la structure et de la fonction des ribosomes. L'étude des molécules d'ARN étant importante pour la compréhension du vivant, il est essentiel de les modéliser et de les manipuler.

Un second exemple de structures biologiques modélisées par des structures arborescentes est l'architecture des plantes. Nous utilisons comme définition des plantes celle biologique qui décrit les plantes comme des êtres vivants pluricellulaires, formant avec les animaux, champignons et protistes les quatre grandes subdivisions du domaine des eucaryotes [RUC+10]. Nous ferons la

distinction entre les végétaux et les plantes, ces dernières sont des végétaux mais les végétaux ne sont pas tous des plantes. Le nombre d'espèces de plantes connues, incluant donc les arbres, est estimé à plusieurs centaines de milliers sur plusieurs millions d'espèces eucaryotes. Les plantes jouent un rôle central sur notre planète. Les biologistes contribuant à la connaissance des plantes le font depuis le niveau moléculaire, jusqu'aux écosystèmes. La Fig. 13 présente un exemple au niveau microscopique et un au niveau macroscopique. Les applications de l'étude des plantes sont nombreuses. Nous pouvons citer le domaine de l'agroalimentaire avec la création de nouvelles plantes [SM02] par hybridation, sélection ou modification génétique. Ces nouvelles plantes ont pour propriété, par exemple, d'être plus nutritives [DJO+07] ou encore de résister aux variations climatiques ou aux insectes qui leur sont nuisibles [CP07]. Leur création est donc un enjeu majeur pour l'ensemble de la planète. Nous pouvons également parler de la sylviculture, c'est-à-dire la culture des forêts [Ada92] et surtout la sylviculture durable [GPT00] ayant pour but d'optimiser durablement leurs potentiels sans surexploiter le milieu. Dans le domaine de l'horticulture, que cela soit pour la production alimentaire [DH00] ou d'agrément par la mise en place de scénarios promouvant la qualité des produits et le respect de l'environnement, mais également en infographie [dREF+88], pour la création d'images numériques plus réalistes.

(a) (b)

Figure 13 : Exemples d'échelles des plantes. (a) Un méristème (organe de la plante), extraite de [BSM+09]. (b) Un chêne d'Allouville-Bellefosse, France, domaine public.

De nombreuses représentations des objets biologiques existent, nous nous intéressons à des représentations discrètes qui les décomposent en entités élémentaires. Les décompositions s'organisent sous forme d'objets mathématiques pouvant être transformés en une structure logique stockant des informations organisées afin d'en faciliter leur traitement [FGS90, CLRS04]. Ces structures de données peuvent être des séquences de caractères (Déf. 1.43), par exemple les suites de nucléotides de l'ADN, ou des structures plus complexes s'appuyant sur la théorie des

graphes telles que les structures arborescentes, voire des graphes (Déf. 1.1) plus généraux pour les réseaux métaboliques [CJ10] ou les réseaux cellulaires [AS06]. Les scientifiques se sont intéressés aux structures arborescentes en premier lieu pour des applications à la chimie. Les représentations sous forme arborescentes de concepts biologiques ont fait leur apparition avec la modélisation de l'évolution des organismes vivants sous forme d'un arbre de la vie par Charles Darwin [Dar72]. La Fig. 14 extraite de [Hae79] représente un des premiers arbres phylogénétiques classifiant les espèces conduisant à l'espèce humaine.

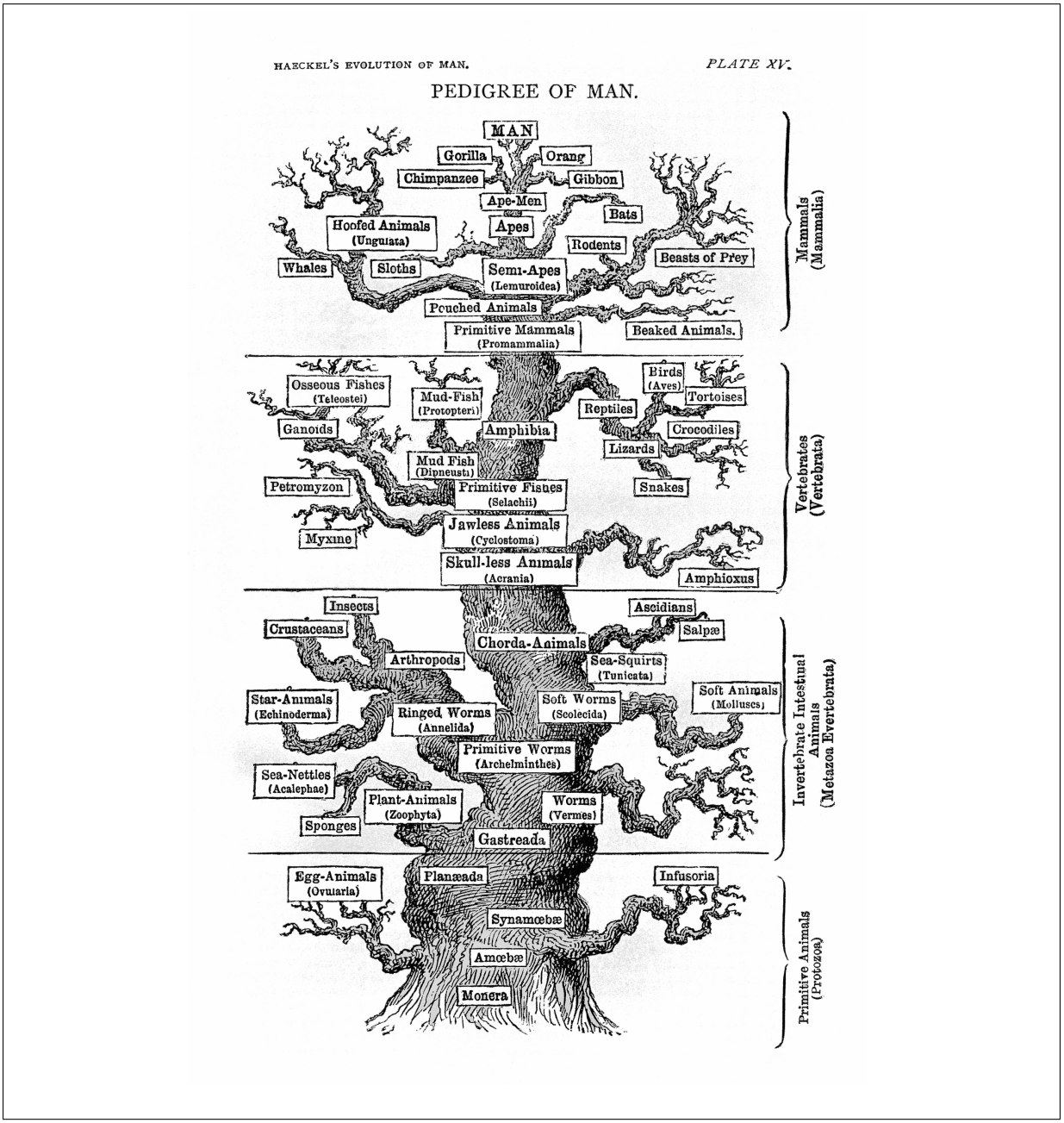


Figure 14 : Un des premiers arbres phylogénétiques datant de 1879, partiellement extraite de [Hae79], domaine public.

L'analyse des modèles représentant des objets biologiques nous conduit à les comparer deux à deux. À partir de ces comparaisons, il est par exemple possible de réaliser des arbres phylogénétiques représentant les relations de parenté entre les organismes en comparant les gènes, d'annoter automatiquement des génomes ou encore de déterminer si deux organismes sont de la même espèce [Mou04]. Une large littérature existe dans ce domaine. Nous nous concentrons sur les méthodes qui comparent les différentes entités élémentaires à l'aide d'un ensemble de transformations permettant de passer d'un objet à un autre, les méthodes d'édition. Ces méthodes permettent de prendre en compte les contraintes biologiques spécifiques à chaque objet.

Deux exemples d'objets biologiques modélisés par des structures arborescentes sont présentés au Chapitre 2, les structures secondaires d'ARN et l'architecture des plantes. Comparer les structures est fondamental, nous explorons dans le Chapitre 3 les algorithmes de comparaison en détaillant des contraintes liées à l'objet modélisé.

Chapitre 2

Représentations arborescentes de structures secondaires des ARN et d'architecture de plantes

De nombreux modèles peuvent être proposés pour chaque objet biologique, nous nous intéressons aux modélisations basées sur une décomposition élémentaire. La détermination de ces unités élémentaires dépend de la nature de l'objet, cela peut, par exemple, être les nucléotides ou les motifs structuraux dans le cas des structures secondaires d'ARN et les unités de croissance ou les branches pour l'architecture des plantes.

Nous nous focalisons sur les structures arborescentes, très utilisées pour la résolution de problèmes pratiques et théoriques. Cependant pour mettre en place des modèles pertinents une connaissance approfondie de l'objet biologique est indispensable. Ainsi les modèles des structures secondaires d'ARN et des architectures des plantes possèdent des contraintes différentes. Par exemple, les structures d'ARN sont généralement modélisées par des arborescences ordonnées [ZS84] (Déf. 1.40) alors que les architectures des plantes peuvent être représentées par des arborescences ordonnées [Oua07], semi-ordonnées [Oua07] (Déf. 1.41) ou non-ordonnées [FG00] (Déf. 1.39).

Nous allons voir, tout d'abord, Section 2.1, une définition de l'ARN à travers sa structure chimique, ses fonctions au sein de la cellule et ses structures, et plus particulièrement, la structure secondaire ainsi que ses diverses représentations. Dans la Section 2.2, nous donnerons une définition plus complète de la plante et de son architecture ainsi qu'une présentation de la représentation de l'architecture des plantes.

2.1 Modélisation de la structure secondaire d'ARN

2.1.1 Définition de l'ARN

Structure chimique

L'ARN, est un polymère composé d'une succession de nucléotides. Il est structurellement comparable à l'ADN. Chaque nucléotide est constitué d'un sucre (pentose), d'un groupement phosphate et d'une base azotée. Ces deux acides nucléiques se distinguent principalement par la nature du pentose. En effet, dans l'ARN le sucre est le ribose tandis que pour l'ADN il s'agit du désoxyribose. Nous pouvons observer Fig. 15 les deux pentoses.

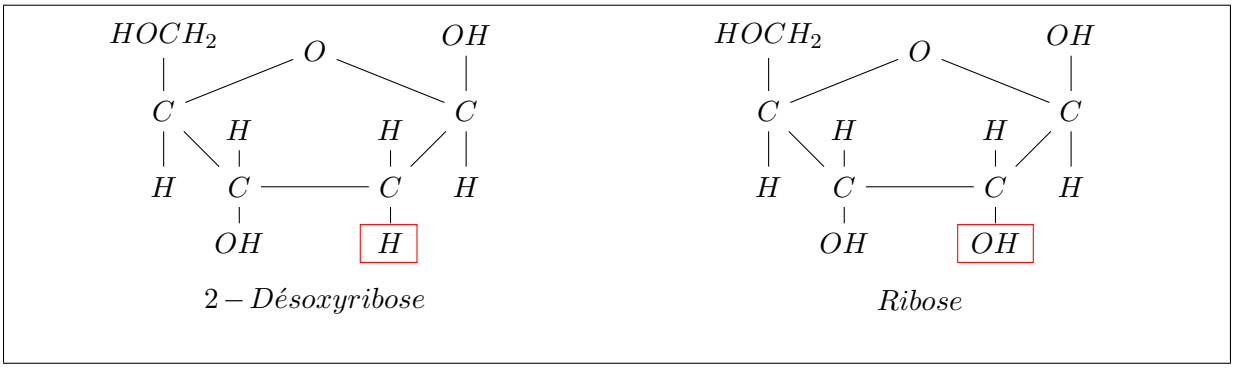


Figure 15 : Comparaison du 2-désoxyribose, composant de l'ADN, et du ribose, composant de l'ARN.

Les bases azotées, dans l'ADN sont l'adénine, la cytosine, la guanine et la thymine, respectivement notées A, C, G et T. Cette dernière n'existe pas dans l'ARN, elle est remplacée par l'uracile noté U. De plus, l'ARN peut être constitué de nucléotides inhabituels, les bases sont alors dites modifiées. Par exemple, la base modifiée pseudouridine, notée Ψ est présente dans les ARN de transfert. La Fig. 16 schématise la structure biochimique de l'ARN. Ainsi chaque nucléotide est constitué d'un ribose (R), dont les carbones sont numérotés de 1' à 5', une liaison phosphodiester est formée entre le carbone 3' et la base azotée (A, C, G, U dans l'exemple) et une autre entre le carbone 5' et le groupement phosphate (P).

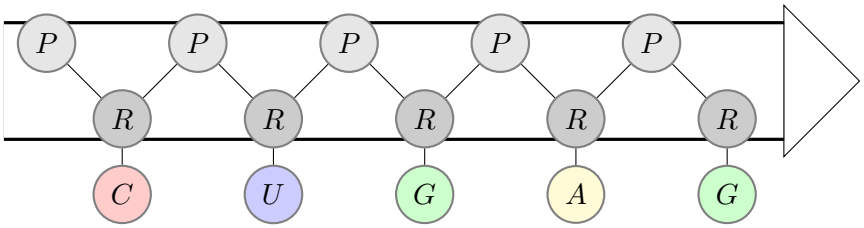


Figure 16 : Schéma de la structure biochimique de l'ARN, P étant l'acide phosphorique, R le ribose, A, C, G et U les bases azotées.

Une molécule d'ARN est, en général, constituée de 50 à 5000 nucléotides contre plusieurs centaines de milliers à plusieurs centaines de millions pour l'ADN et est souvent monocaténaire, c'est-à-dire simple brin.

Les ARN dans la cellule

L'ARN a d'abord été connu pour son rôle dans les processus qui aboutissent à la synthèse de protéines. Les ARN messagers, notés ARNm, servent d'intermédiaire en étant le support de l'information génétique lors de la traduction entre l'ADN et la protéine. Ils sont dits codants. Lors de la traduction, des ARN de transfert (ARNt) spécifiques assurent l'apport d'acides aminés pour la formation de longues chaînes protéiques, synthétisées à l'aide d'un complexe, le ribosome constitué d'ARN ribonucléiques (ARNr) et de protéines.

Récemment, de nouvelles familles d'ARN ont été découvertes. Ces ARN interviennent dans de nombreux processus. Citons les ARN nucléaires (ARNsn) présents dans les cellules eucaryotes

qui participent à l'épissage de l'ARN pré-messager, servent d'amorce lors de la réplication, *etc.* Les petits ARN sont impliqués dans la régulation et y jouent un grand rôle. Rfam est une base de données contenant des informations sur les familles d'ARN non-codants. La version 10.1 de juin 2011 de la base de données Rfam [GJMM+05, GDT+08] regroupe 1446 familles d'ARN non codants contre 379 pour la version 6.1 de mars 2005. Les connaissances des familles fonctionnelles de l'ARN évoluent donc très rapidement.

Structures des ARN

Nous avons vu que les ARN sont généralement monocaténaires. Ils se replient alors sur eux-mêmes. L'ARN adopte une structure en trois dimensions qui lui permet d'accomplir des fonctions complexes. Ainsi la connaissance de la structure de l'ARN est nécessaire bien que non suffisante pour caractériser sa fonction. Comme pour les protéines, un cadre de description de leur structure a été défini. Pour l'ARN, celui-ci est composé de trois niveaux hiérarchiques [LL52]. Il existe également une structure quaternaire qui décrit les interactions entre ARN ou avec des protéines que nous ne détaillerons pas ici.

Structure tertiaire La structure tertiaire est représentée par le graphe de toutes les liaisons chimiques entre les nucléotides. Une liaison chimique caractérise l'échange ou le partage d'au moins un électron entre des atomes.

Structure secondaire La structure secondaire est un sous-graphe (Déf. 1.4) de la structure tertiaire [HSS96, Bon09]. Seules les liaisons de type *Watson-Crick* [WC53], formation de paires entre bases complémentaires c'est-à-dire A-U ou G-C sont gardées. Éventuellement, les liaisons dites *Wobble* [Cri66] entre les bases G et U peuvent être laissées. La structure secondaire représente alors des motifs issus des appariements internes à la séquence.

Structure primaire La séquence ordonnée des nucléotides dans la molécule d'ARN constitue la structure primaire, également appelée séquence. Elle est donnée, par convention, en partant du 5' phosphate vers le 3' OH.

Motifs dans des structures secondaires L'étude des ARN peut se faire à différents niveaux. L'information contenue dans la structure primaire est souvent insuffisante, en effet, deux ARN avec des structures secondaires et tertiaires proches peuvent avoir des structures primaires différentes. L'étude de la structure tertiaire implique généralement une complexité élevée. L'analyse de la structure secondaire est donc un bon compromis.

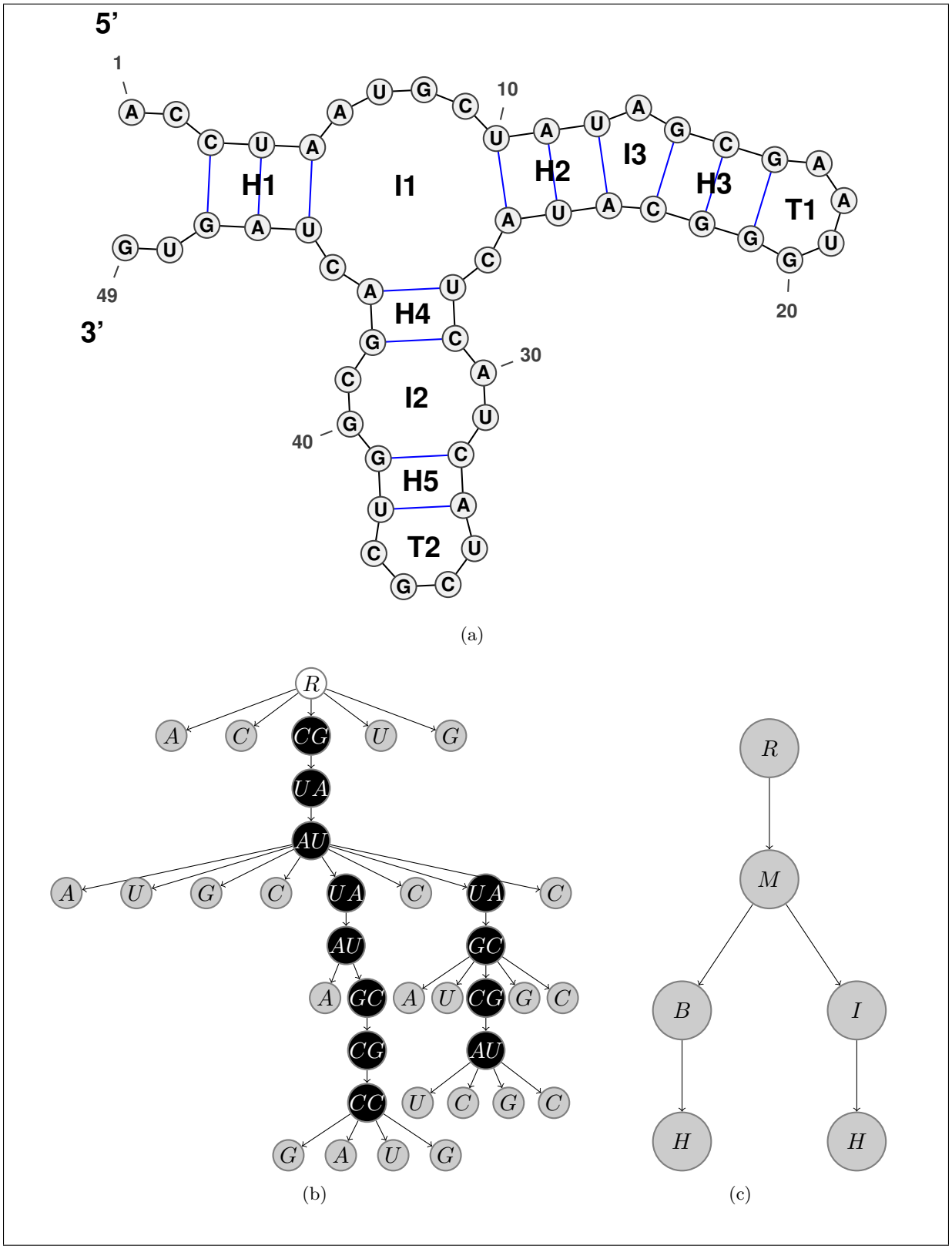


Figure 17 : Représentations d'une structure secondaire d'ARN. (a) Les motifs de structure secondaire d'ARN, illustration réalisée avec le logiciel VARNA [DDP09]. (b) Représentation de Zucker [ZS84] sous forme d'arborescence. (c) Représentation de Shapiro [Sha88] sous forme d'arborescence des éléments de structure.

Plusieurs motifs particuliers ont été remarqués sur les structures secondaires, nous décrirons ceux présentés dans [SZ90]. Un ARN théorique donné Fig. 17(a) les illustre. Tout d'abord les *hélices* qui sont constituées d'une succession de paires de bases appariées, $H1$, $H2$, $H3$, $H4$ et $H5$ sur l'exemple. La *tige boucle* désigne une hélice se terminant par une *boucle terminale*, les boucles $T1$ et $T2$. Une *boucle interne* relie deux hélices, $I3$ et $I2$. Un cas particulier de la boucle interne est le *renflement*, les deux hélices sont liées, d'un côté et de l'autre et possèdent des bases non appariées qui forment le renflement, $I3$. Enfin, la *boucle multiple* relie au moins trois hélices, $I1$. L'appellation *pseudo-noeud* [PRB85] est donnée à une structure qui comporte une interaction entre une boucle d'une structure secondaire et un élément situé en dehors de la boucle. Une structure secondaire est dite sans pseudo-nœud si le graphe la modélisant est planaire (Déf. 1.16), dans la suite de ce document nous considérons les structures secondaires sans pseudo-nœud.

2.1.2 Représentations des structures secondaires d'ARN

Nous allons voir comment ces structures secondaires d'ARN sont modélisées.

Arborescences ordonnées

De nombreux modèles ont été étudiés pour représenter la structure des ARN. Une des représentations les plus courantes est une arborescence ordonnée étiquetée (Déf. 1.19), introduite par Zuker et Sankoff [ZS84]. Chaque feuille (Déf. 1.29) représente une base non appariée et chaque nœud interne (Déf. 1.28) une paire de bases. L'Algorithme 3 explique comment passer de la structure secondaire d'un ARN à une arborescence étiquetée. Le principe est le suivant, un premier nœud P est créé, il constitue la racine (Déf. 1.27). Ensuite pour chaque base de la séquence du $5'$ vers le $3'$:

1. si la lettre est une base appariée telle que la base à laquelle elle est reliée a été traitée alors on remonte dans l'arborescence, P prend alors pour valeur son parent (Déf. 1.31) et la lettre est ajoutée à l'étiquette de P ;
2. si la lettre est une base non appariée un nœud est créé, enfant (Déf. 1.32) de P ;
3. enfin si la lettre est une base appariée telle que la base à laquelle elle est reliée n'a pas été traitée, un enfant est ajouté à droite des enfants de P et à provisoirement pour étiquette la lettre. Le nœud créé devient P.

Chaque paire est alors représentée par un nœud interne et chaque base par une feuille. La Fig. 17(b) montre un exemple d'arborescence construite selon cet algorithme. Il est également possible de ne pas créer un premier nœud racine, dans ce cas le graphe obtenu est une forêt (Déf. 1.25).

D'autres représentations sous formes arborescentes existent, par exemple, celle proposée par Shapiro [Sha88] dans laquelle les éléments de structures secondaires sont représentés par des nœuds étiquetés et les arcs correspondent aux hélices. L'étiquette R correspond à la racine, M aux multiboucles, B aux renflements, I aux boucles internes et H aux boucles terminales. Un exemple est donné Fig. 17(c). Dans la représentation Vienna [HFS+94], à partir de la modélisation de Zuker et Sankoff [ZS84], les fratries (Déf. 1.33) de feuilles sont « compactées » en une seule ainsi que les suites de nœuds internes en un unique nœud interne. Enfin, la modélisation RNAforester [HTGK03] est une version étendue de la modélisation de Zuker et Sankoff [ZS84] dans laquelle

les paires de bases sont représentées par trois nœuds connectés, un nœud interne étiqueté P et deux enfants qui correspondent aux bases appariées, situés à gauche et à droite.

Algorithme 3 *ConstrutionArborescence*(S) : Construction d'une arborescence à partir d'une structure secondaire d'ARN.

Entrées : S une structure secondaire d'ARN.
Sorties : A une arborescence.

1: Créer un arborescence vide A.
2: Créer un nœud P dans A {création de la racine}
3: **Pour tout** base E de la structure du 5' vers le 3' **Faire**
4: **Si** E est une base appariée telle que la base qui s'apparie avec E a été traitée **Alors**
5: Soit P' le parent de P
6: On ajoute E à l'étiquette de P.
7: $P \leftarrow P'$
8: **Sinon Si** E est une base non appariée de S **Alors**
9: On ajoute au nœud P un enfant à droite de la fratrie des enfants.
10: Cet enfant P' est une feuille et son étiquette est E.
11: **Sinon Si** E est une base appariée telle que la base qui s'apparie avec E n'a pas encore été traitée **Alors**
12: On ajoute au nœud P un enfant, à droite de la fratrie des enfants.
13: Cet enfant est un nœud interne et son étiquette est provisoirement E.
14: $P \leftarrow P'$
15: **Fin si**
16: **Fin pour**
17: **Retourner** A

Séquences arc-annotées

Un ARN peut être modélisé par une *séquence arc-annotée* [Eva99] (Déf. 1.45), qui est constituée d'une séquence et d'arcs représentant les liaisons entre les bases. Un exemple de séquence arc-annotée est donné Fig. 18(a) et la même structure sous forme circulaire est présentée Fig. 18(b).

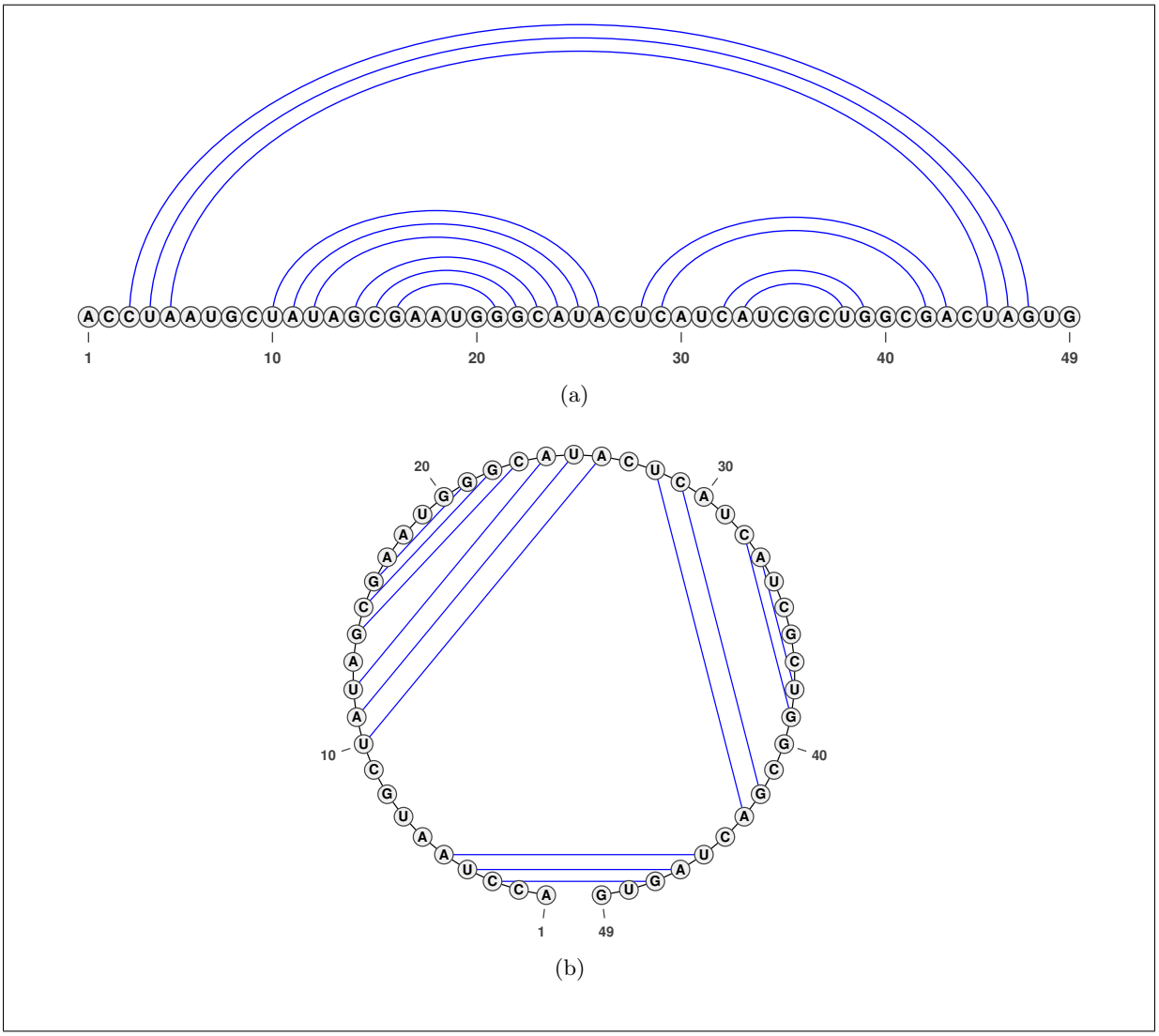


Figure 18 : Deux représentations différentes d'un même ARN. Illustrations réalisées avec le logiciel VARNA [DDP09]. (a) Une séquence arc-annotée. (b) Une vision circulaire.

2.2 Modélisation de l'architecture des plantes

L'*architecture des plantes* est une notion fondamentale utilisée en botanique, décrivant l'organisation de la structure de ces végétaux. Cette notion regroupe l'information sur la forme de la plante dite géométrique mais également sur les liens entre les composants dite information topologique et ceux à différentes échelles.

2.2.1 Notion d'architecture des plantes

Anatomie de la plante L'anatomie d'une plante joue un grand rôle dans la compréhension de la biologie de la plante. Une plante comporte différents organes tels que les organes végétatifs que sont les feuilles, la tige et la racine et les organes reproductifs [RUC+10]. Les feuilles réalisent

la photosynthèse, synthèse de matière organique à partir de la lumière. La tige porte les feuilles et les bourgeons, elle peut se ramifier en branches et rameaux. Enfin, la racine, généralement souterraine et ramifiée fixe la plante et contribue à sa nutrition.

Croissance de la plante Les plantes peuvent être vues comme un ensemble d'entités engendrées par des *méristèmes*. Ce terme a été introduit en 1858 par Karl Wilhelm von Nägeli dans un livre intitulé « Beiträge zur Wissenschaftlichen Botanik »(en français : Les contributions scientifiques à la botanique) [Gal07]. Il désigne un tissu biologique présent dans les plantes. Le méristème est constitué de cellules indifférenciées qui forment une zone de croissance dans laquelle se produit les mitoses, les divisions cellulaires. Ce tissu végétal est essentiel dans le développement de la plante, il crée à lui seul les différents organes de la plante tels que les racines, les feuilles ou encore les tiges, la plante est alors constituée d'un ensemble d'entités élémentaires.

Architecture des plantes

La notion d'architecture des plantes a été introduite dans les travaux d'Hallé *et al.* [HOT78], elle décrit pour une espèce donnée, la croissance idéale d'un individu et fut reprise dans de nombreux travaux [BEH91, Rob96, CB97]. Cependant l'architecture des plantes définit également le processus de croissance d'un individu donné. D'après Ross [Ros81], l'architecture d'une plante désigne « un ensemble d'attributs définissant la forme, la taille, la géométrie et la structure externe de la plante ».

Dans ce manuscrit nous utiliserons une définition générale de l'architecture des plantes présentée dans [God00]. L'architecture d'une plante est la description d'un individu contenant au moins l'une des informations suivantes :

- l'*information de décomposition*, c'est-à-dire l'ensemble des entités qui composent la plante,
- l'*information topologique* décrivant les connexions entre ces entités,
- l'*information géométrique* décrivant sa forme.

Ces informations peuvent être combinées afin de former une représentation plus ou moins complexe de l'architecture d'une plante.

Entités structurelles Ces entités fondamentales pour la description de l'architecture des plantes sont définies dans [CB97, Bel93]. Tout d'abord le *nœud* est l'endroit de la tige où s'insèrent les feuilles. L'*entre-nœud* est la portion de tige comprise entre deux nœuds. Le *métamère* est constitué d'un nœud et de l'entre nœud qui le précède [Whi79]. La structure mise en place par la tige au cours d'une phase d'allongement ininterrompue est appelée *unité de croissance* [CB97]. Enfin *un axe feuillé* est un organe engendré par un même métamère. Un exemple est donné Fig. 19.

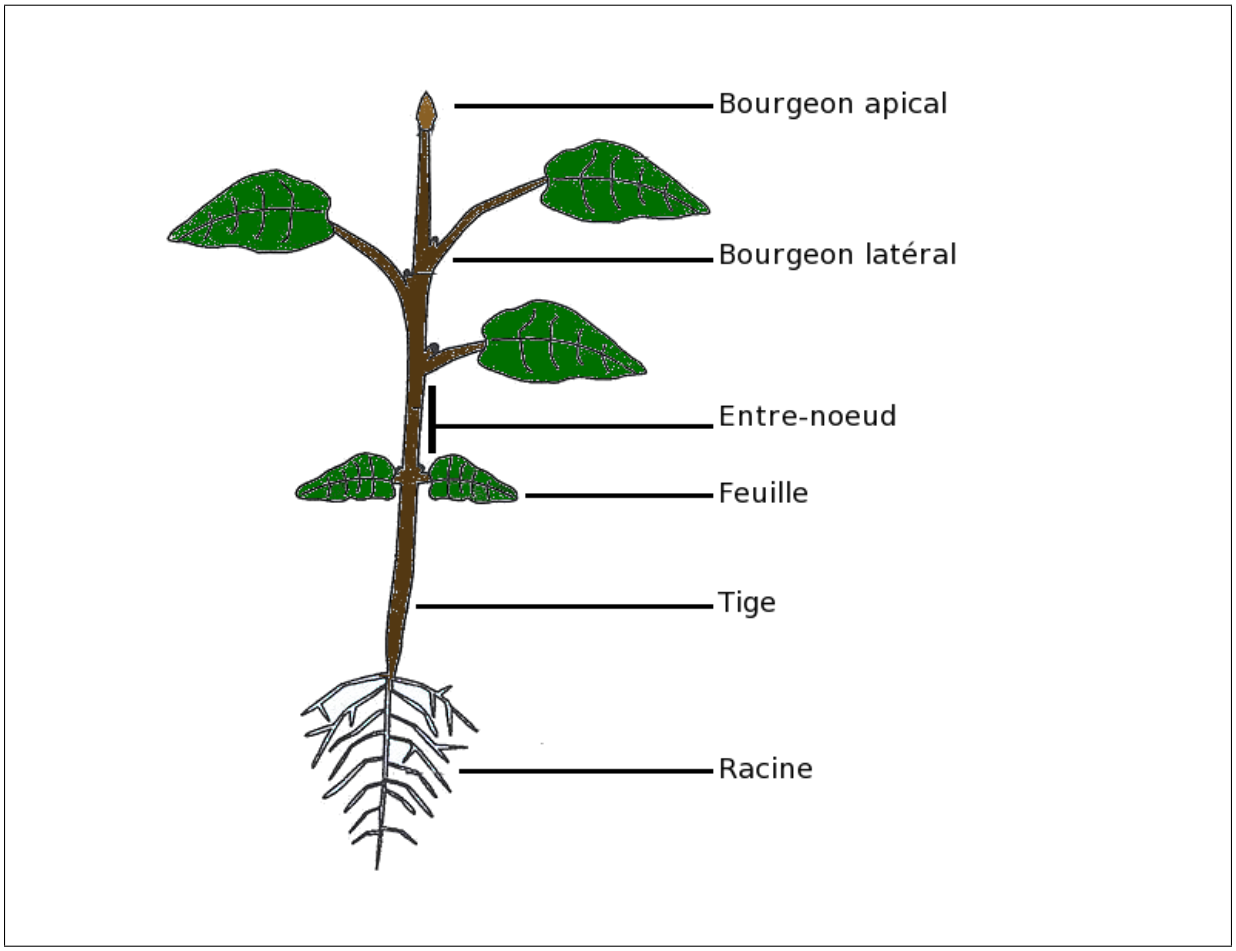


Figure 19 : Unités fondamentales de la plante, illustration sous licence GPL modifiée.

Information topologique La topologie décrit l'organisation des unités de la plante. Généralement utilisée pour modéliser les transferts de substances ou la croissance, l'information topologique peut également servir à simplifier la mesure sur le terrain des plantes. Par exemple, la théorie du « pipe model »(en français : Modèle de tuyau) [SYHK64] considère que certaines unités de la plante peuvent être vues comme un ensemble de tuyaux unitaires. Dans les travaux de Pettunen *et al.* [PSN+96], les ramifications sont modélisées en connectant des tuyaux. Depuis les feuilles jusqu'à la racine, à un croisement, le nombre de tuyaux est la somme du nombre de tuyaux arrivant sur le croisement. Cela permet, par exemple, de donner une approximation du diamètre de la branche en fonction du nombre de tuyaux. Dès les années 70, la topologie d'une plante a été décrite comme fondamentale pour l'étude du développement des plantes [HO70].

Information géométrique La géométrie de la plante caractérise la forme et l'organisation spatiale de ses composants de la plante ou de la plante dans sa globalité. Cette information permet, entre autres, d'étudier les interactions entre la plante et son environnement. L'interception de la lumière par les feuilles peut, par exemple, être étudiée à partir de l'organisation spatiale des feuilles de la plante [STMK98].

2.2.2 Représentations des architectures des plantes

De nombreux modèles de représentation de l'architecture des plantes ont été proposés et dépendent de l'application souhaitée. Sont souvent opposées, les représentations dites globales dans lesquelles la plante n'est pas décomposée et les représentations modulaires basées sur une décomposition spécifique. Nous présentons les représentations modulaires. Pour une présentation plus complète des différentes représentations de l'architecture des plantes, le lecteur pourra se référer à la revue [God00] ainsi qu'à [Bou04].

Représentation de la topologie

Les représentations sous forme de structures arborescentes de la structure topologique des plantes sont généralement basées sur la décomposition de la plante en ensembles de composants élémentaires tels que ceux donnés précédemment [GC98]. Une plante est alors modélisée par une structure arborescente telle que l'ensemble des nœuds représente les composants de la plante et l'ensemble des arcs décrit l'agencement de ces composants, comme cela est illustré Fig. 20.

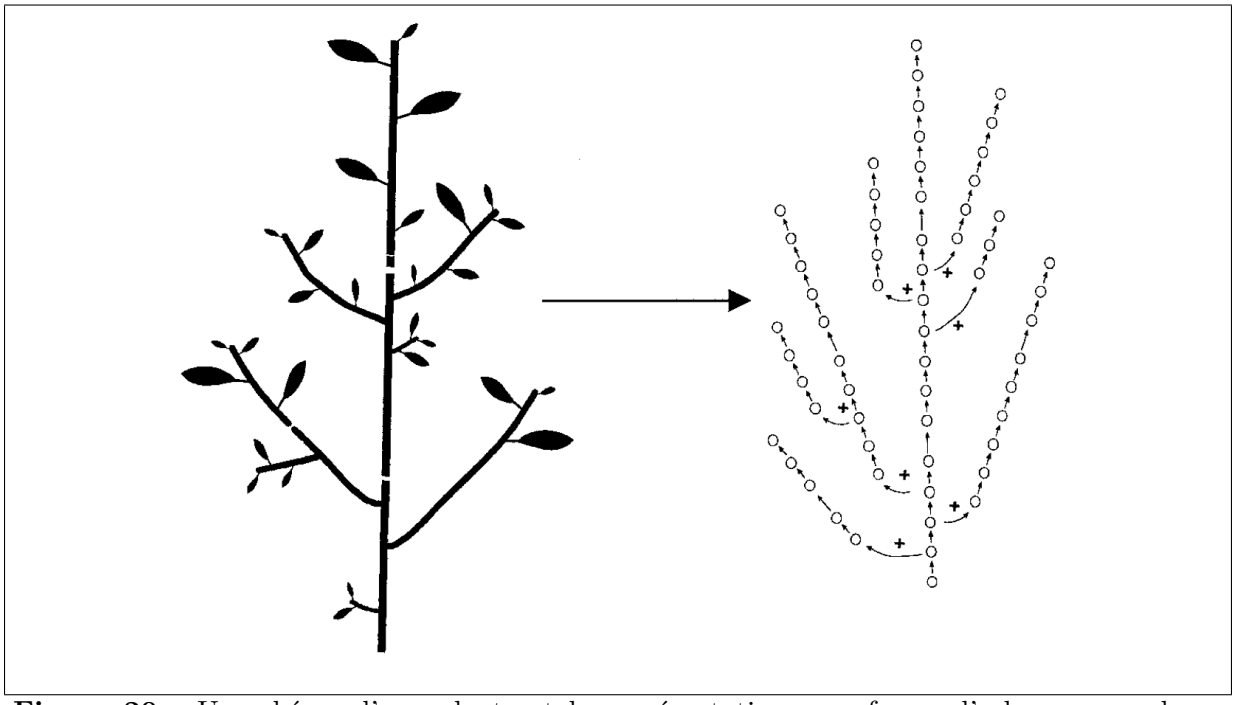


Figure 20 : Un schéma d'une plante et la représentation sous forme d'arborescence de sa topologie, illustration extraite de [FG00].

À une échelle donnée, l'architecture des plantes est alors représentée par un graphe orienté (Déf. 1.3) $T = (V, E)$ dans lequel V est un ensemble de nœuds représentants les entités de la plante à cette échelle et E est l'ensemble des arcs décrivant les connexions entre ces entités, chaque arc est représenté par une paire ordonnée de nœuds tels qu'il existe un arc (v_i, v_j) dans E si et seulement s'il existe une entité correspondante à v_j qui est liée à l'entité parente v_i.

Dans une plante, chaque entité est physiquement attachée à au plus, une entité descendante (à l'exception de la racine qui n'a pas d'entité parente et qui est liée à la racine de la plante).

La structure topologique est alors représentée par une arborescence. Dans le but de prendre en compte les différentes natures des connexions entre les entités botaniques, une définition complémentaire peut être ajoutée à la représentation arborescente de l'architecture de la plante.

Dans une plante, une entité parente peut être connectée à plusieurs entités enfants. Si aucune différence n'est prise en compte dans la nature des connexions entre les parents et les enfants alors aucun ordre n'est considéré entre l'ensemble des entités enfants d'un parent donné. Dans ce cas, l'architecture de la plante est modélisée par une arborescence non-ordonnée [FG00]. En fait, dans certains cas particuliers dans lesquels chaque entité a au plus un enfant, l'ensemble des entités de la plante peut être complètement ordonné pour chacune des échelles de la décomposition et l'architecture de la plante peut être représentée par une arborescence ordonnée avec plus de précision. De même dans le cas des architectures de plantes plus générales, à certaines échelles, le graphe peut être ordonné. Enfin, il est possible de modéliser l'architecture de la plante par des arborescences semi-ordonnées [Oua07].

Représentation de la géométrie

Les primitives géométriques tels que des voxels [Gre89, SB92], des cylindres, des cônes, des polygones ou des modèles géométriques plus complexes constituent une solution pour représenter la géométrie des plantes [Bou04]. Ainsi pour tout composant topologique particulier de la plante est donné une primitive, possiblement redimensionnée.

Une information sur l'orientation est également donnée. Le premier type d'orientation est l'orientation absolue, par exemple la plante est placée dans un repère et les coordonnées cartésiennes des primitives sont données. Les primitives géométriques avec une échelle et une orientation dans l'espace permettent alors une représentation en 3D de la plante, sans que les connexions entre les unités fondamentales ne soient prises en compte. L'orientation absolue peut être remplacée par une orientation relative. Dans cette alternative la géométrie de chaque composant est récursivement calculée relativement à celle de son parent. Cette définition relative est à la base de la géométrie de la tortue, une interprétation des L-systèmes [PL90]. Dans ce cas, la structure topologique des composants de la plante est indispensable afin de définir la relation de parenté. Les orientations relatives et absolues sont présentées Fig. 21.

Notons $p(x)$ le parent d'une unité fondamentale x, g_x sa géométrie et $M_{x/p(x)}$ la transformation définie par le redimensionnement, la position et l'orientation de la primitive représentant la géométrie de x en fonction de son parent. Pour positionner la primitive x dans l'espace, il faut appliquer une suite de transformations. Avec cette définition relative, la géométrie des plantes est définie par un triplet $\{p(x), g_x, M_{x/p(x)}\}$ fabriqué pour chaque composant x à partir de son parent, de la géométrie de sa primitive et de la transformation relative par rapport à son parent. Si la topologie de la plante est donnée par une structure arborescente, nous pouvons étiqueter les nœuds avec la primitive et la transformation.

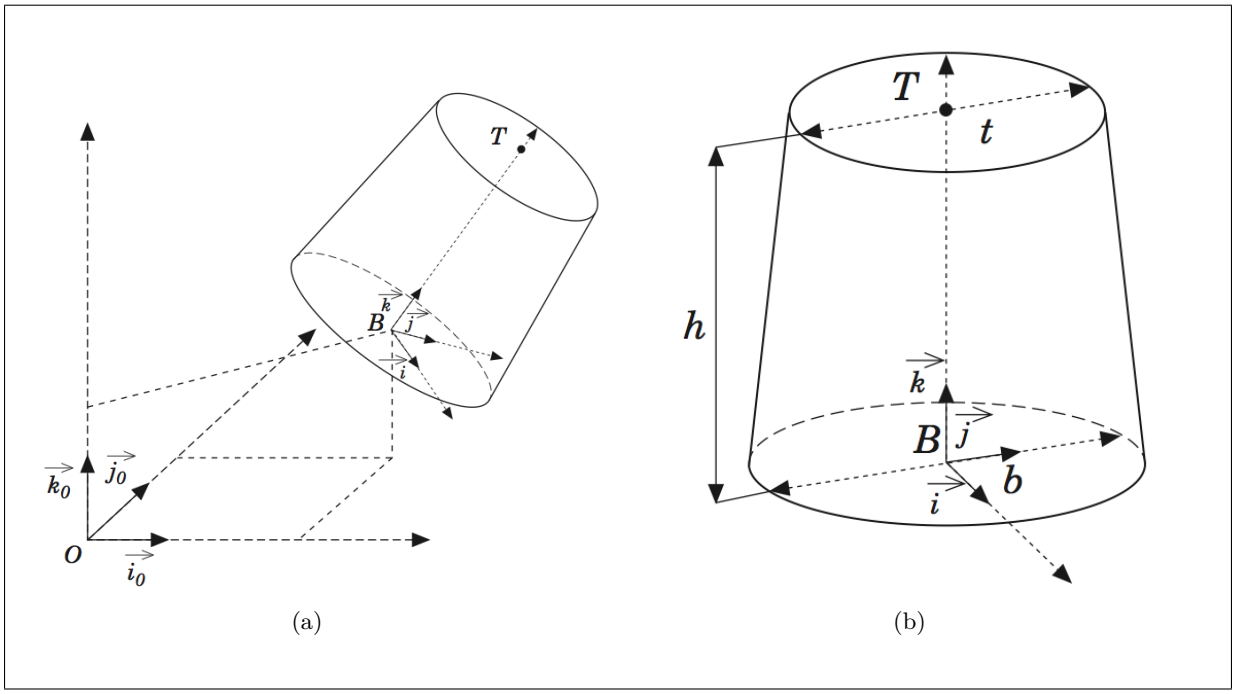


Figure 21 : Modélisation de l'orientation [God00]. (a) Orientation absolue. (b) Orientation relative.

2.3 Autres modélisations

Nous utilisons dans ce manuscrit des représentations sous forme d'arborescences mais celles-ci ne sont pas les seules, comme nous l'avons vu dans le chapitre précédent. Pour les structures secondaires d'ARN, d'autres modélisations peuvent être utilisées, par exemple sous forme multi-échelles. Les représentations multi-échelles proposent une description simultanée de la structure secondaire d'ARN sur plusieurs échelles [AS05, SZ90]. De façon similaire aux structures secondaires d'ARN, une modélisation multi-échelles de l'architecture des plantes est possible. Différents niveaux d'organisation sont alors pris en compte [GGCC97, RP97]. Deux exemples de modélisation multi-échelle sont présentés Fig. 22, les structures secondaires d'ARN sur la Fig. 22(a) et l'architecture des plantes sur la Fig. 22(b). Nous ne traitons pas ces modèles dans la suite de ce document.

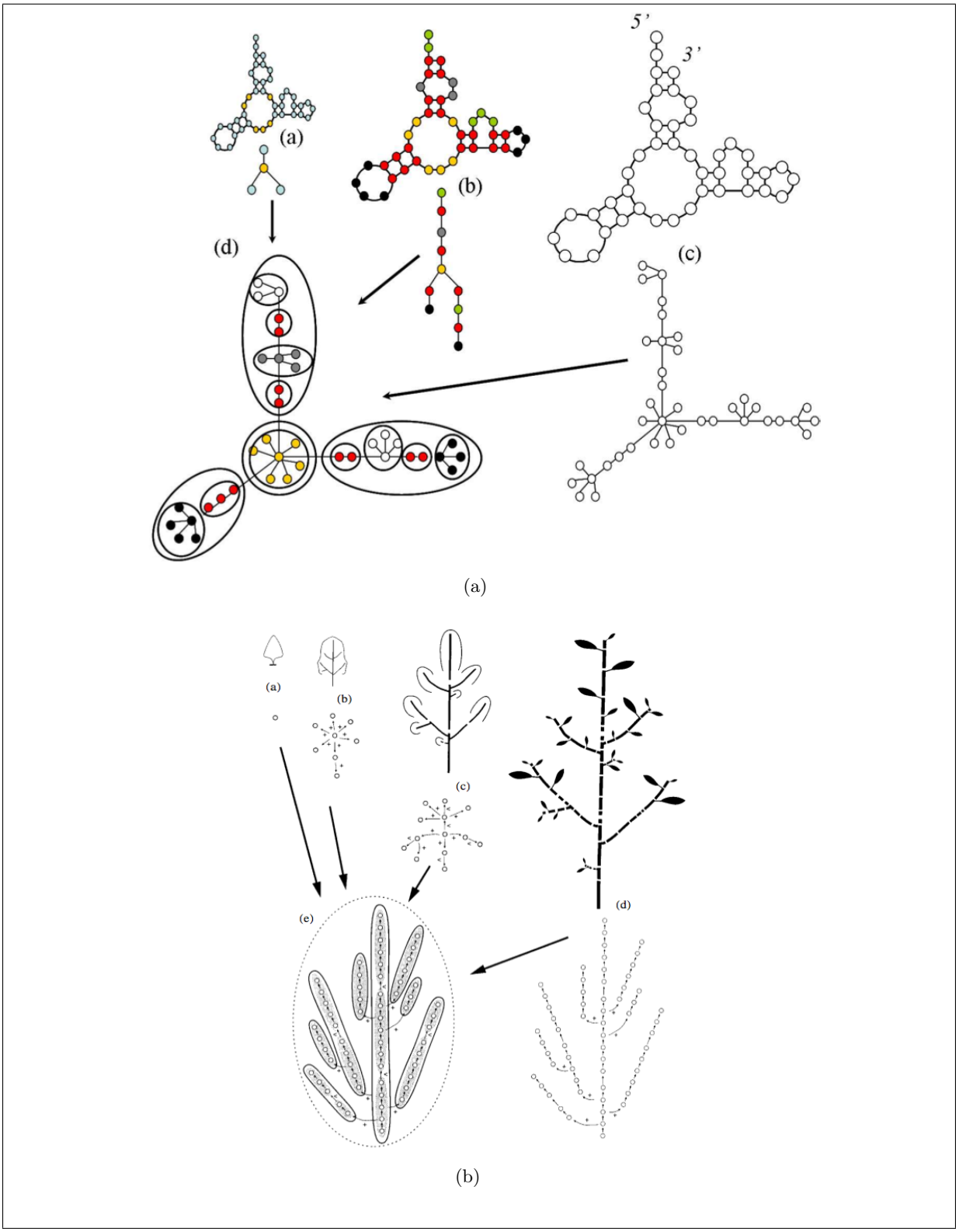


Figure 22 : Représentation multi-échelles d'objets biologiques. (a) Une représentation multi-échelle d'une structure secondaire d'ARN, extraite de [Oua07]. (b) Représentation multi-échelle de l'architecture des plantes, extraite de [GC98].

Chapitre 3

Algorithmes de comparaison de structures arborescentes

La comparaison de deux objets correspond à une mesure de la ressemblance entre ces objets, permettant de prendre en compte les points communs ou les différences. Deux familles de mesures peuvent donc être mises en place [BMRB96], les mesures de similarité entre objets et les mesures de dissimilarités. Elle peut avoir lieu en considérant l'objet dans sa globalité ou en le décomposant en modules. Les comparaisons globales des arborescences considèrent par exemple le nombre de nœuds ou d'arcs. La structure modulaire permet de prendre en compte l'organisation des composants les uns par rapport aux autres, nous nous focalisons sur celle-ci.

L'une des méthodes de comparaison des arborescences est l'utilisation de la distance d'édition, introduite par Selkow [Sel77]. Un ensemble d'opérations élémentaires est utilisé afin de transformer une arborescence en une autre. Par exemple, les opérations d'édition définies par Tai [Tai79] permettent de changer l'étiquette d'un nœud, de supprimer ou d'ajouter un nœud. Depuis, pour certaines applications de nouvelles opérations ont été ajoutées [JLMZ02, All04, Her07]. Le problème d'édition d'arborescences est une extension du problème d'édition de séquences introduit par Levenshtein [Lev66]. Nous présenterons également les problèmes d'alignement [NW70] sur les séquences et les arborescences. De nombreuses méthodes ont été développées pour répondre aux problèmes d'édition et d'alignement d'arborescences, généralement basées sur de la programmation dynamique. Les complexités sont très variables, allant de polynomiale pour le cas des arborescences ordonnées [Tai79, ZS89, Kle98, DT03, DMRW09] à MAX-SNP-difficile pour les cas des arborescences semi-ordonnées et non-ordonnées [ZSS92, Zha96, OF09].

Les mesures de comparaison utilisées sont détaillées Section 3.1. Nous décrivons dans la Section 3.2 les méthodes de comparaison de séquences. La comparaison d'arborescences s'inspirant des principes des séquences est présentée dans la Section 3.3.

3.1 Mesure de comparaison

3.1.1 Mesure de ressemblance

Les mesures de ressemblance sont utilisées pour déterminer les caractéristiques communes entre les individus. La définition de *similarité* (Déf. 3.1) est issue de [DD06].

Définition 3.1 (Similarité) Une similarité s sur un ensemble X est une fonction $s : X \times X \to \mathbb{R}$ telle que $\forall x, y \in X$:

1. $s(x,y) = s(y,x)$ (symétrie),
2. $s(x,y) \leq s(x,x)$,
3. $s(x,y) = s(x,x) \Leftrightarrow x = y$.

Ainsi x est d'autant plus proche de y que la similarité entre x et y, $s(x,y)$, est grande.

3.1.2 Mesure de dissimilarité

À l'inverse les mesures de dissimilarité évaluent les différences entre les objets.

Définition 3.2 (Dissimilarité) Une dissimilarité d sur un ensemble X est une fonction $d : X \times X \to \mathbb{R}$ telle que $\forall x, y \in X$:

1. $d(x,y) = d(y,x)$ (symétrie),
2. $d(x,y) \geq 0$ (non-négativité),
3. $d(x,x) = 0$.

La distance est une dissimilarité particulière qui vérifie l'inégalité triangulaire.

Définition 3.3 (Distance) Une distance d sur un ensemble X est une fonction $d : X \times X \to \mathbb{R}$ telle que $\forall x, y, z \in X$:

1. d est une dissimilarité,
2. $d(x,y) = 0 \Leftrightarrow x = y$ (séparation),
3. $d(x,y) \leq d(x,z) + d(z,y)$ (inégalité triangulaire).

Ainsi x est d'autant plus proche de y que la distance $d(x,y)$ entre x et y est petite. La notion d'édition que nous verrons par la suite est une mesure de dissimilarité.

Les mesures de similarité, de dissimilarité et de distance sont donc définies sur un ensemble X ainsi elles peuvent être calculées entre deux séquences, arborescences ou encore deux objets quelconques.

3.2 Comparaison de séquences

Les solutions apportées pour comparer les séquences sont antérieures à celles sur les arborescences. Les principes et notions sont cependant proches, dans un souci pédagogique nous allons donc commencer par présenter celles relatives aux séquences. Nous considérons deux séquences S_1 et S_2 sur un alphabet Σ et λ la séquence vide (Déf. 1.44).

3.2.1 Distance de Hamming

Une des distances les plus simples est la *distance de Hamming* [Ham50] (Déf. 3.4). Elle correspond au nombre de remplacements de caractères, appelés *substitutions*, à effectuer afin de passer de la première séquence à la seconde.

Définition 3.4 (Distance de Hamming) Soient S_1 et S_2 deux séquences sur l'alphabet Σ tel que $|S_1| = |S_2| = n$. La distance de Hamming entre S_1 et S_2 vaut :

$$d_h(S_1, S_2) = \sum_{i=0}^{n-1} f(S_1[i], S_2[i]) \text{ avec}$$

$$f(x, y) = \begin{cases} 0 & \text{si } x = y \\ 1 & \text{sinon.} \end{cases}$$

Un exemple de distance de Hamming entre les mots COMPARAISON et COMPRESSION est donné Tab. 1.

S_1	C	O	M	P	A	R	A	I	S	O	N
S_2	C	O	M	P	R	E	S	S	I	O	N
i	0	1	2	3	4	5	6	7	8	9	10
$f(S_1[i], S_2[i])$	0	0	0	0	1	1	1	1	1	0	0

La distance de Hamming entre S_1 et S_2 vaut 5.

Tableau 1 – Calcul de la distance de Hamming entre les séquences COMPARAISON et COMPRESSION.

Le calcul de la distance de Hamming entre deux séquences de taille n est réalisé avec une complexité en temps et en espace en $O(n)$, conséquence du parcours et du stockage des caractères des séquences [Pet06].

Cette méthode peut pénaliser fortement la distance entre deux séquences proches. Par exemple, considérons $S_1 = S[0], S[1], \ldots, S[n-2], S[n-1]$ et $S_2 = S[1], S[2], \ldots, S[n-1], S[0]$ avec $\forall i, j \in S_1$ tels que $i \neq j$, $S[i] \neq S[j]$ alors la distance de Hamming entre S_1 et S_2 est de n. En pratique seul le premier caractère de S_1 a été *déplacé* à la fin de S_2. D'autres méthodes permettent de traiter de tels cas, ont vu le jour tels que l'édition et l'alignement.

3.2.2 Édition de séquences

L'édition de séquences consiste à transformer une séquence en une autre à l'aide d'*opérations d'édition*. Elle a été introduite par Levenshtein [Lev66].

Les opérations d'éditions sont l'*insertion*, la *délétion* ou *suppression* et la *substitution* :

- l'insertion consiste à ajouter un caractère $a \in \Sigma$ dans S, elle est notée $\mathbf{ins}(a)$,
- l'opération symétrique de l'insertion, la délétion, un caractère $S[i] \in S$ est supprimé de S, notée $\mathbf{del}(S[i])$,
- la substitution permet le remplacement d'un caractère $a \in \Sigma$ par un autre $b \in \Sigma$ au sein de S, notée $\mathbf{sub}(a, b)$.

Afin d'évaluer une édition, des coûts, notés γ sont attribués à chaque opération. Les coûts doivent vérifier les propriétés d'une distance. Ainsi les fonctions de coûts pour l'édition de séquences doivent respecter les propriétés suivantes :

1. $\mathbf{sub}(a, b)$ est une distance,
2. $\mathbf{del}(a)$ et $\mathbf{ins}(a)$ ont le même score strictement positif,

3. $\mathbf{sub}(a,b) \leq \mathbf{del}(a) + \mathbf{ins}(b)$

Le coût d'une édition E_i est la somme des coûts des opérations de transformation utilisées pour passer d'une séquence à une autre, nous le noterons $\gamma(E_i)$. Le *coût d'édition* (Déf. 3.5) est l'édition de l'ensemble des éditions possibles qui a le coût minimum.

Définition 3.5 (Distance d'édition entre deux séquences) Soient S_1 et S_2 deux séquences de l'alphabet Σ, $\mathcal{E}$ l'ensemble des éditions possibles de S_1 en S_2 et $\gamma(E_i)$ la somme des coûts des opérations utilisées dans l'édition $E_i \in \mathcal{E}$. Si le coût vérifie les propriétés d'une distance, la distance d'édition de S_1 et S_2 vaut :

$$d_e(S_1, S_2) = \min\{\gamma(E_i) | E_i \in \mathcal{E}\}.$$

Un exemple d'édition entre les séquences COMPARAISON et COMPRESSION est donné Tab. 2. En donnant pour coût de 1 pour l'insertion, la délétion et la substitution le coût de cette édition entre COMPARAISON et COMPRESSION est de 6. La distance d'édition est cependant de 5 en substituant les mêmes caractères que pour la distance de Hamming.

S_1	C	O	M	P	A	**R**	A	I	S	O	N
sub(R,E)											
	C	O	M	P	**A**	**E**	A	I	S	O	N
sub(A,R)											
	C	O	M	P	**R**	E	A	**I**	S	O	N
del(I)											
	C	O	M	P	R	E	**A**		S	O	N
del(A)											
	C	O	M	P	R	E			S	O	N
ins(I)											
	C	O	M	P	R	E		S	**I**	O	N
ins(S)											
S_2	C	O	M	P	R	E	**S**	S	I	O	N

Tableau 2 – Une édition entre les séquences COMPARAISON et COMPRESSION.

3.2.3 Alignement de séquences

L'alignement de séquences consiste en la création d'une séquence commune aux séquences [NW70] qui sont comparées comme cela est présenté entre les séquences COMPARAISON et COMPRESSION sur le Tab. 3. Cette séquence commune est appelée super-séquence.

S_1	C	O	M	P	A	R	A	I	S		O	N
	(C,C)	(O,O)	(M,M)	(P,P)	(A,$\emptyset$)	(R,R)	(A,E)	(I,S)	(S,S)	($\emptyset$,I)	(O,O)	(N,N)
S_2	C	O	M	P		R	E	S	S	I	O	N

Tableau 3 – Une super-séquence entre les séquences COMPARAISON et COMPRESSION.

L'alignement de séquences (Déf. 3.6) est une alternative à l'édition.

Définition 3.6 (Alignement de séquences) Soient S_1 et S_2 deux séquences, Σ un alphabet et $\Sigma^- = \Sigma \cup \{-\}$. Un alignement de S_1 et S_2 est une super-séquence de S_1 et S_2, c'est-à-dire, $A = (x_0, y_0), \ldots, (x_p, y_p)$ telle que :

1. $\forall 0 \leq i \leq p$, $(x_i, y_i) \in (\Sigma^-)^2 \setminus \{(-,-)\}$,
2. la séquence $x_0 \ldots x_p$ (*resp.* $y_0 \ldots y_p$) privée des x_i (*resp.* y_i) tels que $x_i = -$ (*resp.* $y_i = -$) est exactement la séquence S_1 (*resp.* S_2).

Dans le cas des séquences, l'édition et l'alignement sont des problèmes équivalents, ainsi l'*insertion*, la *délétion* ou *suppression* et la *substitution* correspondent respectivement à la mise en correspondance d'un caractère de la séquence S_1 avec un caractère $-$, à la mise en correspondance d'un caractère $-$ avec un caractère de la séquence S_2 et à la mise en correspondance d'un caractère de S_1 avec un caractère dans S_2.

Définition 3.7 (Score d'alignement de séquences) Soient S_1 et S_2 deux séquences de l'alphabet Σ, $\mathcal{A}$ l'ensemble des alignements possibles de S_1 en S_2 et $\sigma(A_i)$ la somme des scores des mises en correspondances utilisées dans l'alignement A_i. Le score d'alignement entre S_1 et S_2 vaut :

$$s_a(S_1, S_2) = \max\{\sigma(A_i) | A_i \in \mathcal{A}\}.$$

Un exemple d'alignement entre COMPARAISON et COMPRESSION est donné Tab. 3.2.3.

S_1	C	O	M	P	A	R	-	A	I	S	O	N
S_2	C	O	M	P	-	R	E	S	S	I	O	N

Tableau 4 – Un alignement entre les séquences COMPARAISON et COMPRESSION.

Le score d'alignement peut être calculé grâce à des algorithmes de programmation dynamique tels que l'algorithme de Needleman et Wunsch [NW70], présenté Algorithme 4, nous ne donnons pas le détail des équations de récurrence permettant le calcul. Une alternative à cette alignement dit global est l'alignement local tel que celui de Smith et Waterman [SW81], dans le calcul du maximum lignes 10-13, $\sigma(S_1[0] \ldots S_1[i], \lambda)$ est réinitialisé à 0 si les autres scores sont négatifs.

La complexité de cet algorithme pour le calcul du score d'alignement entre S_1 et S_2 est en $O(n^2)$ en temps et en $O(n)$ en espace avec $n = \max(|S_1|, |S_2|)$. Crochemore *et al.* [CLZu02] ont proposé un algorithme avec une complexité en temps en $O(\frac{n^2}{log(n)})$ dans le cas général et en $O(\frac{hn^2}{log(n)})$ avec $0 < h \leq 1$, h étant une variable représentant l'entropie dans le texte, en utilisant l'algorithme de compression LZ78 [ZL78] [1], et une complexité en espace en $O(\frac{h^2 n^2}{(log(n))^2})$.

1. Les méthodes de compression de séquences, dont l'algorithme LZ78, sont présentées Section 5.1.1.

Algorithme 4 *ScoreAlignement(S_1, S_2)* : Calcul de la distance d'alignement entre deux séquences S_1 et S_2.

Entrées : S_1 et S_2 deux séquences telles que $|S_1| = n$ et $|S_2| = m$.
Sorties : Le score d'alignement entre les deux séquences.

1: $score(\lambda, \lambda) = 0$
2: **Pour** i de 0 à n-1 **Faire**
3: $\quad \sigma(S_1[0] \dots S_1[i], \lambda) = \sigma(S_1[0] \dots S_1[i-1], \lambda) + \mathrm{S}(\mathbf{del}(S_1[i]))$
4: **Fin pour**
5: **Pour** j de 0 à m-1 **Faire**
6: $\quad \sigma(\lambda, S_2[0] \dots S_2[j]) = \sigma(S_2[0] \dots S_2[j-1], \lambda) + \mathrm{S}(\mathbf{ins}(S_2[j]))$
7: **Fin pour**
8: **Pour** i de 0 à n-1 **Faire**
9: $\quad$ **Pour** j de 0 à m-1 **Faire**
10: $\quad\quad \sigma(S_1[0] \dots S_1[i], \lambda) = max\{$
11: $\quad\quad\quad \sigma(S_1[0] \dots S_1[i-1], S_2[0] \dots S_2[j-1]) + \mathrm{S}(\mathbf{sub}(S_1[i], S_2[j]))$;
12: $\quad\quad\quad \sigma(S_1[0] \dots S_1[i], S_2[0] \dots S_2[j-1]) + \mathrm{S}(\mathbf{del}(S_1[i]))$;
13: $\quad\quad\quad \sigma(S_1[0] \dots S_1[i-1], S_2[0] \dots S_2[j]) + \mathrm{S}(\mathbf{ins}(S_2[j]))$
14: $\quad\quad \}$
15: $\quad$ **Fin pour**
16: **Fin pour**
17: **Retourner** $\sigma(S_1, S_2)$

3.3 Comparaison d'arborescences

De nombreuses méthodes de comparaison d'arborescences existent, nous ne les détaillerons pas toutes dans ce manuscrit. Le lecteur pourra alors se référer à [Bil05] pour une revue plus exhaustive sur la comparaison d'arborescences et les problèmes qui lui sont liés.

3.3.1 Édition d'arborescences

Sur les arborescences, les trois opérations d'édition sont définies et introduites par Tai [Tai79].

Les opérations d'éditions sont la *délétion (ou suppression) de nœuds*, l'*insertion de nœud* et la *substitution de nœuds* :

- l'insertion du nœud v consiste à définir un ensemble de nœuds consécutifs d'une fratrie comme enfant de v. L'ancien parent de ces nœuds consécutifs devient le parent de v.
- l'opération symétrique de l'insertion. Lorsque l'on supprime le nœud v les nœuds enfants de v deviennent les nœuds enfants du parent de v.
- la substitution consiste à modifier l'étiquette d'un nœud.

Définition 3.8 (Mise en correspondance) Soient deux arborescences T_1 et T_2, V_1 et V_2 leurs ensembles respectifs de nœuds, une mise en correspondance (« mapping » en anglais) $\mathcal{M}$ est un ensemble de couples $(v_i, v_j) \in (V_1, V_2)$ tels que pour tout $(v_i, v_j) \in \mathcal{M}$ et $(v'_i, v'_j) \in \mathcal{M}$:

1. $v_i = v'_i$ si et seulement si $v_j = v'_j$,
2. v_i est un ancêtre de v'_i si et seulement si v_j est un ancêtre de v'_j,

3. dans le cas d'arborescences ordonnées la relation d'ordre total (Déf. 1.38) doit être conservée, dans le cas semi-ordonné la relation de semi-ordre total (Déf. 1.37) doit être conservée, dans le cas des arborescences non-ordonnées seules les deux premières closes doivent être vérifiées.

Définition 3.9 (Coût d'une édition d'arborescences) Soit $\mathcal{M}$ une mise en correspondance définissant une édition E entre deux arborescences T_1 et T_2, le coût de l'édition E est donné par : $d_E(T_1,T_2) = \sum_{(v_1,v_2)\in\mathcal{M}} \gamma(v_1 \to v_2) + \sum_{v\in T_1} \gamma(v \to \lambda) + \sum_{v\in T_2} \gamma(\lambda \to v)$. Le coût de l'édition correspond au coût de l'édition minimum entre les deux arborescences. Nous notons $\gamma(\mathcal{M})$ le coût de la mise en correspondance.

En ce qui concerne la résolution de l'édition d'arborescences, nous allons séparer trois cas que sont les arborescences ordonnées, semi-ordonnées et non-ordonnées.

Arborescences ordonnées

Pour les arborescences ordonnées la relation d'ordre entre les nœuds doit être préservée, les algorithmes utilisent une méthode de programmation dynamique basée sur une décomposition des arborescences. L'algorithme de Tai [Tai79] possède une complexité en temps et en espace en $O(|T_1||T_2||L(T_1)|^2|L(T_2)|^2)$ avec $L(T_i)$ le nombre de feuilles dans l'arborescence T_i. Quelques années plus tard, l'algorithme de Zhang-Shasha [ZS89] a proposé une solution en $O(|T_1||T_2|min(D(T_1),L(T_1))min(D(T_1),L(T_1)))$ en temps et en $O(|T_1||T_2|)$ en espace avec $D(T_i)$ la profondeur (Déf. 1.30) de T_i. Le pire cas est alors en $O(n^4)$ avec n la taille maximale des deux arborescences (Déf. 1.2). L'algorithme dit de Zhang-Shasha est basé sur une décomposition à gauche. Enfin, Demaine *et al.* [DMRW09] ont amélioré la complexité en pire cas en temps en $O(|T_1||T_2|^2(1+\log(\frac{|T_1|}{|T_2|}))$ ce qui revient à une complexité en $O(n^3)$.

Arborescences non-ordonnées

Nous allons présenter les résultats du problème d'édition sur les arborescences non-ordonnées dans le cas général et dans un cas contraint.

Cas général Dans le cas des arborescences non-ordonnées le problème de l'édition est *Max-SNP-difficile*. Les problèmes de la classe de complexité *Max-SNP* [PY88] étant des problèmes d'optimisation, en l'occurrence de maximisation, correspondant à des problèmes de décision de la classe de complexité *strict NP*[2]. Les problèmes de la classe de complexité *Max-SNP-difficile* sont donc au moins aussi difficiles que l'ensemble de l'ensemble des problèmes *Max-SNP*. Les algorithmes connus pour résoudre les problèmes de la classe de complexité *Max-SNP-difficile* ont une complexité en temps exponentielle en la taille de l'entrée et n'ont pas d'approximation pouvant être calculé en temps polynomial.

L'édition d'arborescences non-ordonnées reste *Max-SNP-difficile* dans le cas d'arborescences binaires étiquetées sur un alphabet à deux caractères [ZSS92]. Un algorithme a été proposé par Zhang *et al.* [ZSS92] pour résoudre l'édition générale entre deux arborescences non-ordonnées, la complexité en temps est en $O(|T_1||T_2| + k!2^k(k^3 + deg^+(T_2)^2)|T_2|)$ avec k le nombre de feuilles de $|T_1|$ et $deg^+(T_2) = \max_{v\in V(T_2)} deg^+(v)$ avec $deg^+(v)$ le degré sortant du nœud v (Déf. 1.6).

2. La classe de complexité *strict NP* étant incluse dans *NP* (*cf.* Section 1.4).

Cas contraint Dans un certain nombre d'applications, il est possible de prendre en compte des contraintes supplémentaires pour la mise en correspondance. L'idée étant que lors de l'édition contrainte entre deux arborescences T_1 et T_2, deux sous-arborescences (Déf. 1.34) distinctes dans l'arborescence T_1 doivent être mises en correspondances avec deux sous-arborescences distinctes dans T_2 (Déf. 3.10). L'insertion et la suppression d'un nœud v est seulement possible entre un nœud et l'ensemble de ces enfants, entre un nœud et un seul de ces enfants ou lorsque le nœud est une feuille. Ce qui signifie que la mise en correspondance illustrée Fig. 23 n'est pas possible. Le problème d'édition contraint n'est alors plus dans la classe de complexité *Max-SNP-difficile*.

Définition 3.10 (Mise en correspondance contrainte) Une mise en correspondance contrainte $\mathcal{M}_c$ est une mise en correspondance telle que pour toutes paires (v_i, v_j), (v'_i, v'_j) et $(v''_i, v''_j) \in \mathcal{M}_c$: $(v_i \wedge v'_i) \leq v''_i \Leftrightarrow (v'_i \wedge v'_j) \leq v''_j$ avec $x \wedge y$ un ancêtre commun de x et y tel que pour tout ancêtre commun z de x et y, $z \leq (x \wedge y)$.

L'algorithme de Zhang [Zha96] propose une résolution du problème d'*édition contrainte*, les équations sont données Déf. 3.11 et Déf. 3.12. Notons que le problème se ramène à un problème d'appariement qui est résolu comme un problème de flot par l'algorithme de Edmons et Karp [EK72] puis amélioré par Tarjan [Tar83]. La méthode de Zhang possède alors une complexité en $O(|T_1||T_2|(\mathrm{Deg}(T_1)+\mathrm{Deg}(T_2))log((\mathrm{Deg}(T_1)+\mathrm{Deg}(T_2)))$ avec $\mathrm{Deg}(T_i)$ le degré maximum des nœuds de T_i et est décrite dans l'Algorithme 5.

Définition 3.11 (Distance d'édition contrainte entre sous-forêts) Soient T_1 et T_2 deux arborescences, $F_1[v_i]$ et $F_2[v_j]$ deux forêts induites par v_i et v_j, la distance d'édition contrainte d_C entre les deux sous-forêts est définie comme suit :

$$d_C(F_1[v_i], F_2[v_j]) = \min \begin{cases} \min\{\gamma(\mathcal{M}_c) | \mathcal{M}_c \in (F_1[v_i], F_2[v_j])\} \\ \min\limits_{v_k \in enf(v_j)} \{d_C(F_1[v_i], F_2[v_k]) - d_C(\emptyset, F_2[v_k]) + d_C(\emptyset, F_2[v_j])\} \\ \min\limits_{v_k \in enf(v_i)} \{d_C(F_1[v_k], F_2[v_j]) - d_C(F_1[v_k], \emptyset) + d_C(F_1[v_i], \emptyset)\}. \end{cases}$$

Définition 3.12 (Distance d'édition contrainte entre sous-arborescences) Soient T_1 et T_2 deux arborescences, $T_1[v_i]$ et $T_2[v_j]$ deux sous-arborescences complètes enracinées en v_i et v_j, la distance d'édition contrainte d_C entre les deux sous-arborescences est définie comme suit :

$$d_C(T_1[v_i], T_2[v_j]) = \min \begin{cases} d_C(F_1[v_i], F_2[v_j]) + \gamma(v_i \to v_j) \\ \min\limits_{v_k \in enf(v_j)} \{d_C(T_1[v_i], T_2[v_k]) - d_C(\emptyset, T_2[v_k]) + d_C(\emptyset, T_2[v_j])\} \\ \min\limits_{v_k \in enf(v_i)} \{d_C(T_1[v_k], T_2[v_j]) - d_C(T_1[v_k], \emptyset) + d_C(T_2[v_i], \emptyset)\}. \end{cases}$$

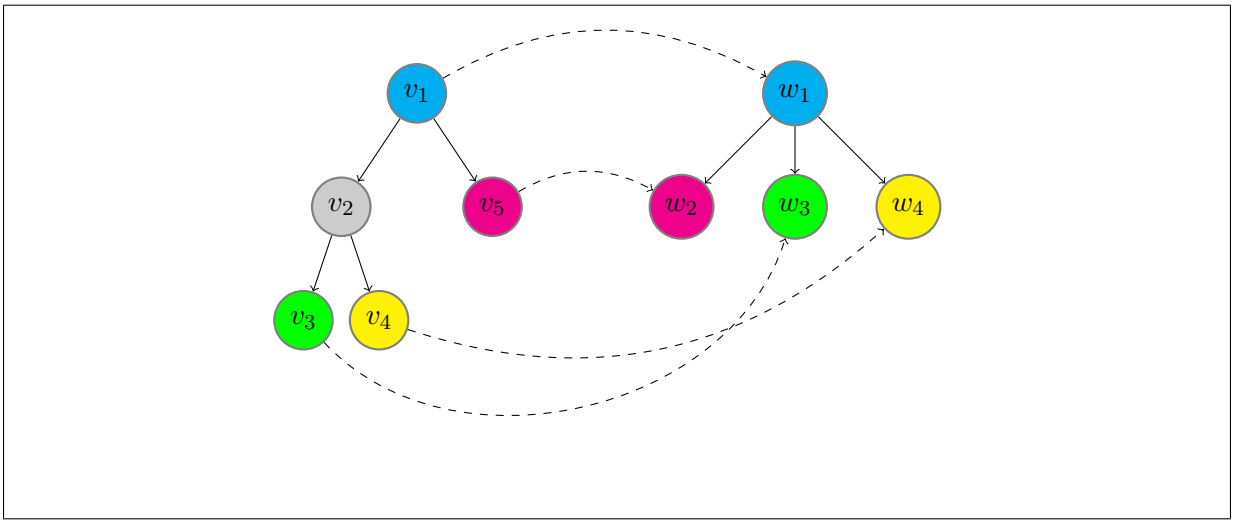


Figure 23 : Mise en correspondance non valide dans le cas de la distance d'édition contrainte.

Algorithme 5 *CoûtÉdition(T_1, T_2)* : Calcul du coût d'édition entre deux arborescences T_1 et T_2.

Entrées : $T_1 = (V_1, E_1)$ et $T_2 = (V_2, E_2)$ deux arborescences, de taille n et m.

Sorties : Le coût d'édition entre les deux sous-arborescences $T_1[v_i]$ et $T_2[v_j]$ avec $v_i \in V_1$ et $v_j \in V_2$.

1: $d(\emptyset, \emptyset) = 0$
2: **Pour tout** $v_i \in V_1$ **Faire**
3: $\quad d(F_1[v_i], \emptyset) = \sum_{v_k \in V_1} d(T_1[v_k], \emptyset])$
4: $\quad d(T_1[v_i], \emptyset) = d(F_1[v_i], \emptyset) + \gamma(T_1[v_i] \to \lambda)$
5: **Fin pour**
6: **Pour tout** $v_i \in V_2$ **Faire**
7: $\quad d(\emptyset, F_2[v_i]) = \sum_{v_k \in V_2} d(\emptyset, T_2[v_k]])$
8: $\quad d(\emptyset, T_2[v_i]) = d(\emptyset, F_2[v_i]) + \gamma(\lambda \to T_2[v_i])$
9: **Fin pour**
10: **Pour** i de 0 à $n-1$ **Faire**
11: $\quad$ **Pour** j de 0 à $m-1$ **Faire**

12: $$d_C(F_1[v_i], F_2[v_j]) = \min \begin{cases} \min\{\gamma(\mathcal{M}) | \mathcal{M} \in (F_1[v_i], F_2[v_j])\} \\ \min\limits_{v_k \in enf(v_j)} \{d_C(F_1[v_i], F_2[v_k]) - d_C(\emptyset, F_2[v_k]) + d_C(\emptyset, F_2[v_j])\} \\ \min\limits_{v_k \in enf(v_i)} \{d_C(F_1[v_k], F_2[v_j]) - d_C(F_1[v_k], \emptyset) + d_C(F_1[v_i], \emptyset)\} \end{cases}$$

13: $$d_C(T_1[v_i], T_2[v_j]) = \min \begin{cases} d_C(F_1[v_i], F_2[v_j]) + \gamma(v_i \to v_j) \\ \min\limits_{v_k \in enf(v_j)} \{d_C(T_1[v_i], T_2[v_k]) - d_C(\emptyset, T_2[v_k]) + d_C(\emptyset, T_2[v_j])\} \\ \min\limits_{v_k \in enf(v_i)} \{d_C(T_1[v_k], T_2[v_j]) - d_C(T_1[v_k], \emptyset) + d_C(T_2[v_i], \emptyset)\} \end{cases}$$

14: $\quad$ **Fin pour**
15: **Fin pour**
16: **Retourner** $d(T_1[v_i], T_2[v_j])$

Arborescences semi-ordonnées

Le cas des arborescences semi-ordonnées a été traité par Ouangraoua et Ferraro [OF09]. Le problème général reste *Max-SNP-difficile* mais ils ont proposé une solution à l'édition contrainte dans laquelle les sous-arborescences distinctes doivent être mises en correspondances avec des sous-arborescences distinctes mais le semi-ordre doit également être conservé. La méthode proposée est en $O(|T1||T2|(deg^+(T1)+deg^+(T2))\log_2(deg^+(T1)+deg^+(T2)))$ en temps.

3.3.2 Alignement d'arborescences

L'alignement de deux arborescences T_1 et T_2 consiste à construire une super-arborescence commune aux deux arborescences.

En ce qui concerne les arborescences, le calcul de l'édition et l'alignement ne sont pas équivalents, comme l'illustre la Fig. 24. La Fig. 24(a) représente les opérations d'édition utilisées pour passer d'une arborescence à une autre tandis que la Fig. 24(b) représente la construction d'une arborescence commune aux deux arborescences.

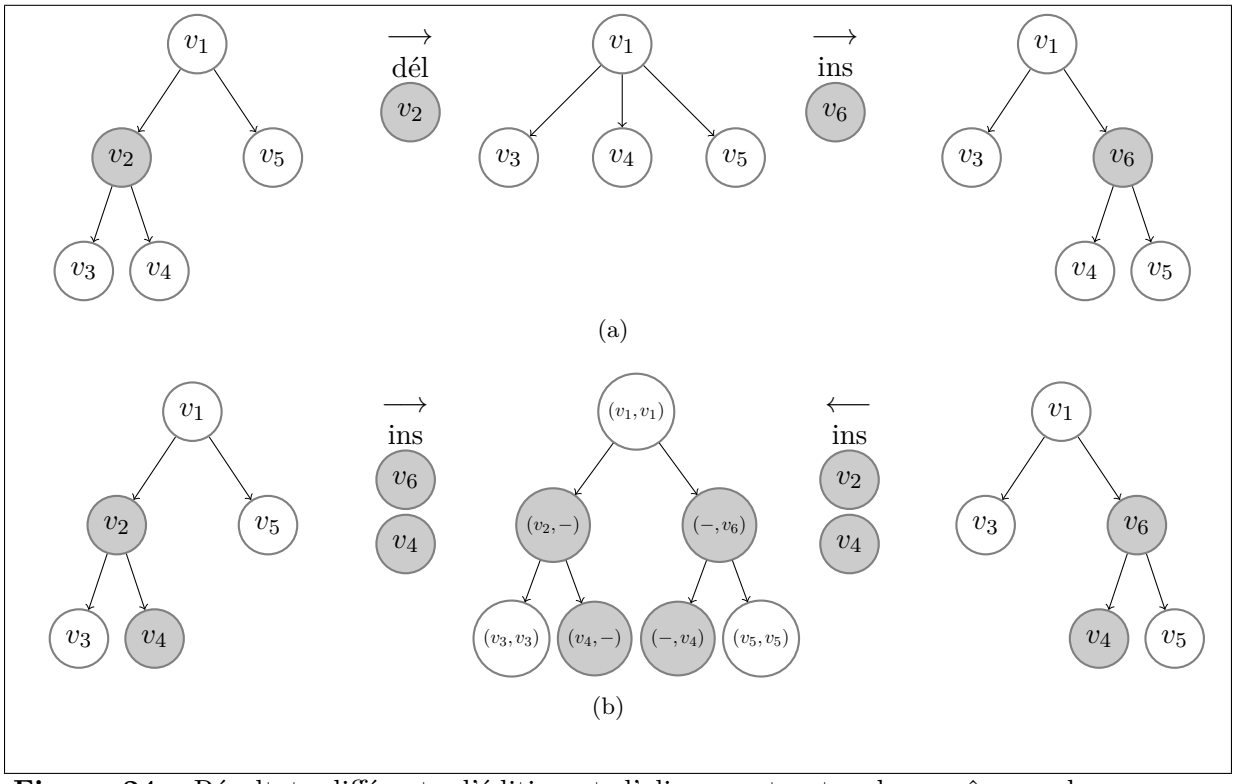


Figure 24 : Résultats différents d'édition et d'alignement entre deux mêmes arborescences. (a) Résultat d'une édition entre deux arborescences. (b) Résultat d'un alignement entre deux arborescences.

L'alignement est une édition contrainte dans laquelle toutes les insertions doivent précéder les délétions. Ce problème dans le cas d'arborescences ordonnées, fut introduit par [JWZ95] qui proposèrent un algorithme en $O(|T1||T2|(I1+I2)^2)$ en temps et en $O(|T1||T2|(I1+I2))$ en espace. Dans le cas des arborescences non-ordonnées, la complexité est en $O(|T_1|,|T_2|)$ si les deux arborescences sont bornées [JWZ95].

Diverses méthodes existent pour comparer des structures que cela soit des séquences ou des arborescences. Nous pouvons tout de même remarquer deux points importants : l'alignement et l'édition d'arborescences sont basés sur la comparaison de sous-arborescences et ses méthodes sont au moins quadratiques en pire cas. Afin de comparer efficacement une arborescence contre un ensemble d'arborescences, d'autres méthodes peuvent être proposées telle que celle présentée Chapitre 4.

Conclusion et perspectives de la partie 1

Les structures secondaires d'ARN et les architectures des plantes peuvent se modéliser sous forme d'arborescences, qu'elles soient ordonnées, semi-ordonnées ou non-ordonnées. Nous avons défini l'édition et l'alignement, des méthodes de comparaison deux à deux d'arborescences basées sur des opérations d'édition. Nous nous sommes focalisés sur les opérations d'édition définies par Tai [Tai79], mais dans le cadre d'une application aux structures secondaires d'ARN, de nouvelles opérations ont été introduites.

Opérations d'édition pour les applications aux ARN

En 2002, Jiang *et al.* [JLMZ02] ont défini dans le cas de la comparaison de séquences arcs-annotées, de nouvelles opérations plus réalistes pour les ARN. Dans cette section, nous supposerons que les arborescences ont été construites selon le modèle présenté Section 2.1.2. Ils ont remarqué que les trois opérations d'édition définies par Tai [Tai79], c'est-à-dire l'insertion, la suppression et la substitution étaient applicables sur les bases mais également sur les paires de bases, ce qui correspond dans le cas des arborescences aux feuilles mais aussi aux nœuds internes. De plus, ils ont défini d'autres opérations qui correspondent à la perte, l'altération ou au gain d'un arc.

Les opérations sont illustrées Fig. 25. Elles peuvent être traduites sous forme d'arborescence. L'insertion, la suppression et la substitution d'une base correspondent aux opérations présentées Section 3.2.2. La délétion, correspondance et substitution d'arc sont équivalentes aux opérations de la Section 3.3.1. Les deux autres opérations n'ont pas été définies précédemment.

La *perte d'un arc*, notée *scission d'une paire de bases* dans [Her07], a lieu sur un nœud interne correspondant à une paire de bases. Notons ce nœud v, v modélise une paire de bases, son étiquette représente deux bases appariées que nous noterons v_1 et v_2. Deux nouvelles feuilles sont créées correspondant aux bases v_1 et v_2. La première est placée à gauche dans la fratrie enfant de v tandis que la seconde est placée à la fin, à droite. Le nœud v est supprimé. Le père de v devient le père des enfants de v. L'opération inverse est appelée *fusion de deux bases* dans [Her07].

L'*altération d'un arc*, dite *altération gauche d'une paire de bases* et *altération droite d'une paire* dans [Her07] a également lieu sur un nœud interne correspondant à une paire de base. Notons ce nœud v, v modélise une paire de base, son étiquette représente deux bases appariées

que nous noterons v_1 et v_2. Une nouvelle feuille est créée, dans le cas de l'altération gauche il correspond à la base v_1, dans le cas de la complétion droite à v_2. La feuille est placée en premier (à gauche) dans la fratrie enfant de v dans le cas gauche, tandis qu'elle est placée à la fin (à droite) dans le cas droit. Le nœud v est supprimé. Le père de v devient le père des enfants de v. L'opération inverse est appelée *complétion gauche d'une paire de bases* et *complétion droite d'une paire* dans [Her07].

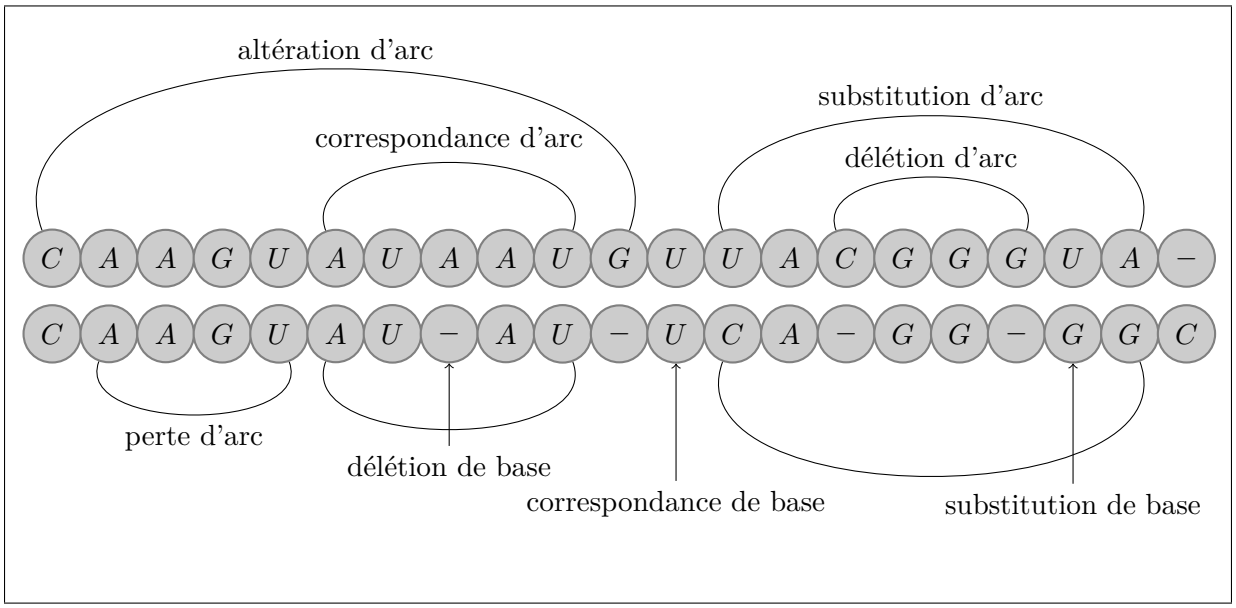


Figure 25 : Les opérations d'édition réalistes pour l'ARN définies sur les bases et les paires de bases par [JLMZ02].

Dans le cas des arborescences, de nouvelles opérations ont été définies dans les travaux [All04, Her07]. Ces opérations permettent une meilleure prise en compte de l'objet biologique étudié.

Nouvelle opération réaliste pour l'architecture des plantes

Sur le même principe, nous proposons une opération réaliste pour l'architecture des plantes. Nous avons vu que l'architecture des plantes pouvait être décomposée selon des unités fondamentales. Dans le cas d'une erreur, de mesure par exemple, des unités peuvent être dupliquées. Une opération intéressante que nous notons fusion, consiste en un regroupement d'un ensemble de nœuds le long d'un chemin au sein d'un même nœud. Ainsi, les enfants des nœuds fusionnés sont regroupés au sein d'une même fratrie. L'opération inverse, appelée scission, consiste à scinder un nœud en un ensemble de nœuds le long d'un chemin. Ainsi les enfants du nœud scindé sont répartis comme enfant d'un nœud de l'ensemble de nœud. La fusion d'un ensemble N de n nœuds en un nœud v et la scission d'un nœud v en l'ensemble N de n nœuds valent le même coût strictement positif.

Remarque 12 La fusion de n nœuds est équivalente à une délétion de $n-1$ nœuds et à une substitution. La fonction de coût de l'édition doit alors être inférieure à celui de la somme de $n-1$ délétion et d'une substitution.

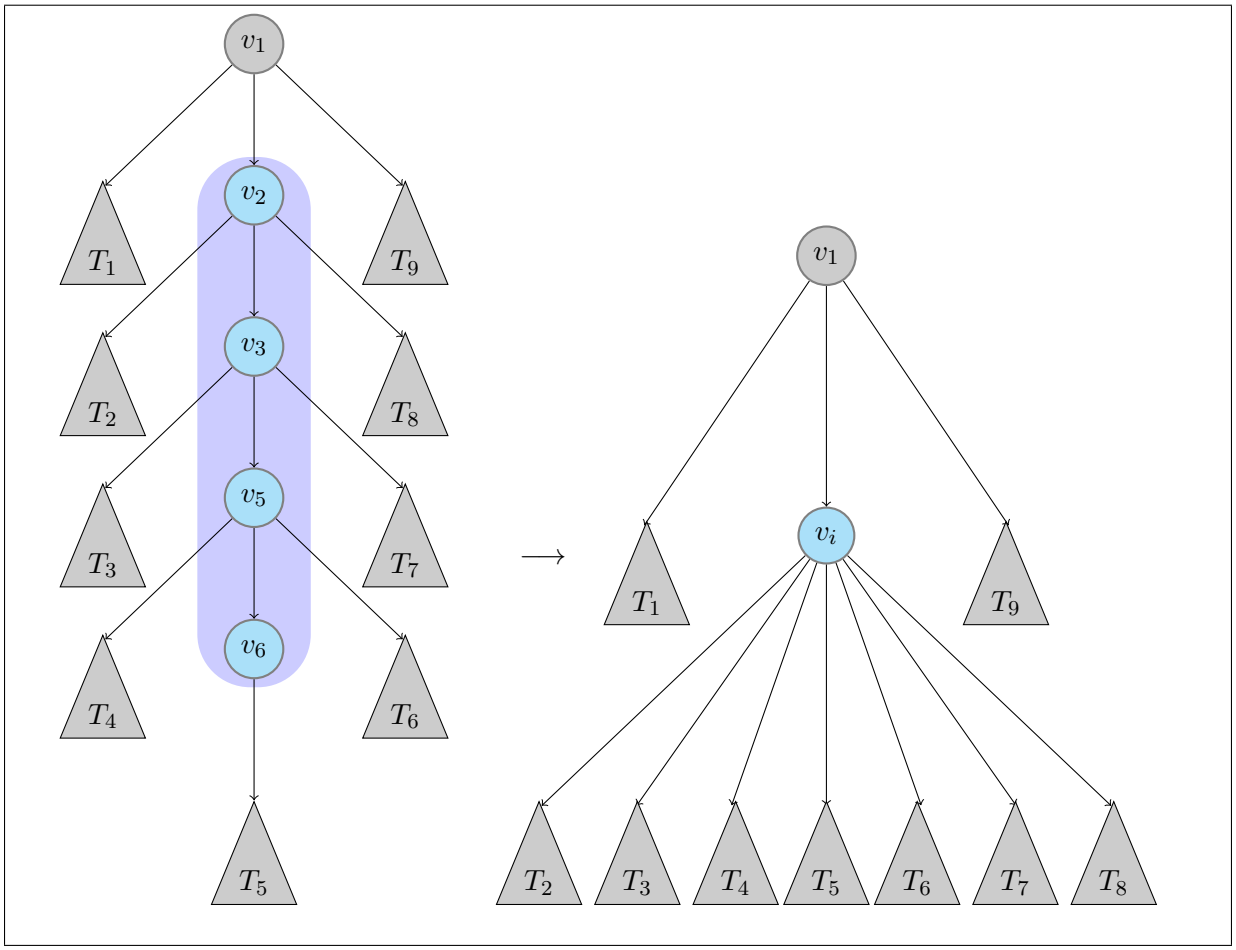


Figure 26 : Opération de fusion sur les arborescences.

Il convient de vérifier que la distance d'édition contenant ces opérations reste une distance.

Propriété 1 Soit d une distance d'édition entre deux arborescences T_1 et T_2 non-ordonnées permettant les opérations de délétion, insertion, substitution, la fusion et la scission. d est une distance si et seulement si :

- la suppression et l'insertion d'un nœud valent le même coût qui est strictement positif,
- la substitution est une distance,
- la fusion et la scission avec le coût définit précédemment.

Preuve Si d est une distance alors :

- soit v_i un nœud, $\mathbf{dél}(v_i) = d(v_i, \emptyset) = d(\emptyset, v_i) = \mathbf{ins}(v_i)$ donc $\mathbf{dél}(v_i) = \mathbf{ins}(v_i) > 0$,
- soient v_i et v_j deux nœuds, $\mathbf{sub}(v_i, v_j) = d(v_i, v_j)$ donc $\mathbf{sub}(v_i, v_j) \geq 0$,
- soient T_1 et T_2 deux arborescences, v_i un nœud de T_1 et N un ensemble de nœuds de T_2 tels que la fusion du nœud v_i en N transforme T_1 en T_2, $\mathbf{fus}(v_i, N) = d(T_1, T_2) = d(T_2, T_1) = \mathbf{sci}(N, v_i) \geq 0$

Pour que d soit une distance, il faut vérifier les propriétés présentées Déf 3.3 :

1. Symétrie, $\mathbf{sub}(v_i, v_j) = \mathbf{sub}(v_j, v_i)$, $\mathbf{dél}(v_i) = \mathbf{ins}(v_i)$, $\mathbf{fus}(v_i, N) = \mathbf{sci}(N, v_i)$, les opérations d'édition symétriques ont des coûts égaux alors la suite d'opérations pour l'édition de T_1 à T_2 a le même score que celle pour passer de T_2 à T_1,

2. Non-négativité, d a pour valeur la somme de coûts, tous positif, d est donc positive,
3. Séparation, si $d(v_i, v_j)$ est nulle alors $v_i = v_j$, le seul coût nul étant la substitution d'un nœud par un même nœud,
4. Inégalité triangulaire, supposons qu'il existe une arborescence T_3 telle que $d(T_1, T_2) > d(T_1, T_3) + d(T_3, T_2)$ alors les suites d'opérations définies par $d(T_1, T_3)$ et $d(T_3, T_2)$ peuvent être contactées pour réaliser une édition entre T_1 et T_2 qui aurait un coût inférieur à $d(T_1, T_2)$, or d'après la définition de la distance d'édition cela n'est pas possible. □

Il faudrait, pour l'application de la comparaison à l'architecture des plantes, déterminer une méthode de comparaison avec de telles opérations et les appliquer afin de déterminer leurs pertinences.

Deuxième partie

Traitement des redondances au sein des structures biologiques arborescentes

Chapitre 4

Filtrage de structures arborescentes

Nous avons présenté dans la partie précédente des méthodes de comparaison entre arborescences. Une application fondamentale de la comparaison est la recherche dans des bases de données d'un ensemble d'objets qui sont similaires à un objet requête. Avec le récent développement des technologies de séquençage haut débit et des méthodes d'annotation, trouver de nouvelles méthodes de comparaison efficaces est devenue indispensable pour l'analyse d'objets combinatoires complexes, utilisés pour des modèles de structures biologiques avec une plus grande complexité [PBS+06], notamment pour les structures arborescentes.

Sur les séquences, la comparaison deux à deux basée sur les distances d'éditions entre une requête et toutes les séquences de la base de données n'est pas applicable sur des bases de données de grande taille compte tenu de la complexité du calcul de la distance d'édition. Une approche classique pour améliorer les complexités en temps d'exécution lors de la comparaison de séquences est de dépendre de petites sous-séquences appelées *graines*, présentent dans la requête. Les graines peuvent être détectées rapidement en utilisant des techniques d'indexation [Bro07] ; ensuite un ensemble optimal de graines appelé *chaînes*, qui doivent être colinéaires dans la requête mais également dans la séquence de la base de données sont considérées. Cette technique est largement utilisée, notamment dans les programmes tels que BLAST [AGM+90] et FASTA [LP85, PJ88]. En contrepartie de la vitesse d'exécution, les méthodes sont des heuristiques qui ne garantissent pas l'obtention du meilleur appariement. Le lecteur pourra se référer aux revues [Gus97, Alu05] traitant de la comparaison de séquences en bioinformatique.

Ce travail a été réalisé en partie à Vancouver en collaboration avec Julien Allali (LaBRI), Cédric Chauve (SFU) et Pascal Ferraro (LaBRI). Il a été présenté lors de l'International Workshop on Combinatorial Algorithms (IWOCA) 2010 à Londres [ACFG11] et a été accepté dans Journal of Discrete Algorithms (JDA) [ACFG].

Nous proposons dans ce chapitre une méthode de comparaison d'une requête contre une base de données appliquée aux structures arborescentes. Dans la Section 4.1, nous présenterons plus finement les résultats de filtrage sur les séquences. Dans la Section 4.3, nous donnons une définition de graines pour les arborescences. Enfin dans la Section 4.2, nous proposons une méthode pour calculer le chaînage maximal de graines entre arborescences et permettre d'envisager la définition d'heuristiques de type BLAST pour les arborescences.

4.1 Filtrage sur les séquences

Dans le domaine de l'analyse de séquences, l'approche standard pour explorer de larges bases de données de séquences moléculaires dans le but de trouver des homologues à une séquence requête est de dépendre de petites sous-séquences, appelées *graines*, présentes dans la requête. Les graines peuvent être détectées très rapidement dans la base de données en utilisant des techniques d'indexation telles que les tables de hachage. En pratique, la base de données peut être filtrée rapidement pour éliminer toutes les séquences qui n'ont pas assez de graines en commun avec la requête. Il en résulte un nombre réduit de candidats potentiellement proches de la requête.

Ensuite, chaque candidat conservé est comparé à la requête. Pour cela une détection d'un ensemble optimal de graines a lieu. Cet ensemble qui respecte le même ordre relatif sur les graines dans les deux séquences est appelé *chaîne*. On suppose qu'un score est associé à chaque graine ainsi qu'à chaque chaîne, correspondant généralement à la somme des scores des graines dans la chaîne. Finalement les candidats tels que les chaînes ont un score élevé sont proposés comme potentiellement homologues à la séquence requête et sont comparés à celle-ci en utilisant un algorithme d'édition de distance plus coûteux en temps.

À la vue de la taille des bases de données de séquences génomiques, ces approches requièrent des algorithmes efficaces à la fois pour la phase de filtrage mais également pour la phase de chaînage. De plus, la définition des graines est cruciale pour permettre la pertinence du résultat. Idéalement, dans un premier temps, il faudrait qu'aucune séquence avec une distance d'édition faible avec la requête ne soit éliminée durant la phase de filtrage. Par la suite, il est souhaitable de limiter les séquences avec une distance d'édition élevée qui passent cette phase.

Nous présenterons dans la Section 4.1.1 le principe général et les méthodes utilisant un tel principe. Dans la Section 4.1.2, nous présenterons des exemples de graines définies sur les séquences. Enfin, Section 4.1.3, nous donnerons les méthodes de chaînages sur les séquences.

4.1.1 Filtrage sur les séquences

Plusieurs méthodes réalisent un filtrage basé sur une recherche de segments de texte [LP85, PJ88, AGM+90, Gus97, Alu05].

Principe d'index

Dans un dictionnaire, les données sont organisées selon un ordre lexicographique, les mots sont une suite finie de lettres de l'alphabet qui est un ensemble ordonné. La recherche d'un mot dans cet ouvrage est accélérée grâce à ce principe. Sur un principe similaire, lorsqu'un livre possède un index, chaque mot indexé possède une unique entrée pointant vers un ensemble de références du mot.

Cette idée a été adaptée pour l'étude des séquences en bioinformatique. Nous avons une séquence, par exemple d'ADN, et cherchons au sein de cette séquence des motifs particuliers, comme les séquences cis, séquence d'ADN capable de moduler l'expression d'un gène, telles que la boîte TATA (« Goldberg-Hogness box » ou « TATA box » en anglais). Le problème formalisé revient à traiter une séquence S de grande taille, notée n et à chercher des parties de S qui sont égales à certains motifs, qui ne sont pas encore connus.

Une première solution, sans prétraitement sur S obligera lors de chaque recherche à parcourir S entièrement. La complexité sera alors en $O(n)$ pour chaque recherche. Lorsque le nombre de requêtes est élevé, une telle solution n'est plus envisageable. Ainsi des méthodes alternatives basées sur un prétraitement de S ont vu le jour. Une structure de données appelée index de S est alors créée. De la séquence S sont extraites toutes les sous-parties possibles. Ces éléments sont ordonnés de façon à ce que chaque requête prenne un temps proportionnel à la taille de celle-ci. L'index est alors construit puis compressé afin d'en éliminer les redondances. Chaque requête est donc réalisée plus rapidement en $O(m)$ avec m la taille de la requête.

Principe sur deux séquences

Dans le cas de deux séquences, nous ne cherchons plus des motifs au sein d'une même structure mais des éléments communs entre deux structures.

Une approche basée sur des points d'attaches peut être utilisée. Naïvement, l'idée est de construire un alignement global à partir d'un ensemble d'alignements locaux. Les petites parties mises en correspondance sont appelées graines ou fragments. Ces pièces sont alors chainées les unes aux autres en respectant des contraintes afin d'assurer leurs compatibilités.

L'approche générale est la suivante [OA05] :

1. calcul des fragments, c'est-à-dire des motifs, et association d'un score à chaque fragment,
2. calcul d'une chaîne optimale de fragments non chevauchants,
3. compléter les trous en alignant les régions entre les points d'attaches.

Ces méthodes donnent une solution approximative au problème de recherche dans une base de données, elles sont donc des heuristiques. Deux méthodes très célèbres utilisent une telle technique, FASTA et BLAST. Dans les deux cas une séquence requête est comparée à toutes les séquences d'une base de données.

FASTA [LP85, PJ88] est un algorithme de comparaison de séquences pouvant se découper en différentes tâches. Premièrement, des sous-séquences communes sont recherchées, généralement via une table de correspondance ou de hachage. La longueur standard étant de 4 à 6 nucléotides pour l'ADN et de 1 à 2 acides aminés pour les protéines. Ensuite, les meilleures correspondances proches de la *diagonale* sont sélectionnées. Le terme « diagonale » faisant référence à la vision des alignements via un dot-plot. Le programme évalue alors les possibles regroupements entre les similarités trouvées.

Dans la méthode BLAST, les séquences sont préalablement traitées de façon à ce que les régions de faible complexité ou trop répétés ne soient pas prises en compte. Le logiciel BLAST [AGM+90] ne considère qu'un mot de taille bien supérieur à ceux de FASTA, généralement de l'ordre d'une dizaine de caractères pour l'ADN. Ce mot est ensuite étendu le long des séquences. De nombreuses variantes ont été proposées tel que BLAT [Ken02], plus rapide mais moins précise.

4.1.2 Graines sur les séquences

D'un point de vue général, soient deux objets combinatoires X et Y, une graine peut être définie comme une correspondance exacte entre une partie de X et une partie de Y. De nombreux modèles de graines pour les séquences ont été proposés [NK05].

Par exemple, les graines entre deux séquences peuvent être des sous-séquences de taille fixe contigües [LP85, PJ88, AGM+90, Mor96]. Ces occurrences exactes d'une taille commune fixée sont appelées q-gram. Une propriété importante des q-gram est que le nombre de q-gram pour une séquence est linéaire en la taille de la séquence. Cependant des modèles plus complexes ont été proposés, relâchant la notion de correspondance exacte. Les méthodes par hachage ont introduit la notion de fragments espacés [CR93], repris par les graines optimisées [MTL02, KLMT02, NK05, KNR05], leurs performances sont généralement meilleures [Bro07].

4.1.3 Chaînage dans des séquences

Nous distinguerons deux sortes de chaînage, celui au sein d'une seule séquence appelé chaînage 1D [GMP96] et le chaînage entre deux séquences, le chaînage 2D [JMT92].

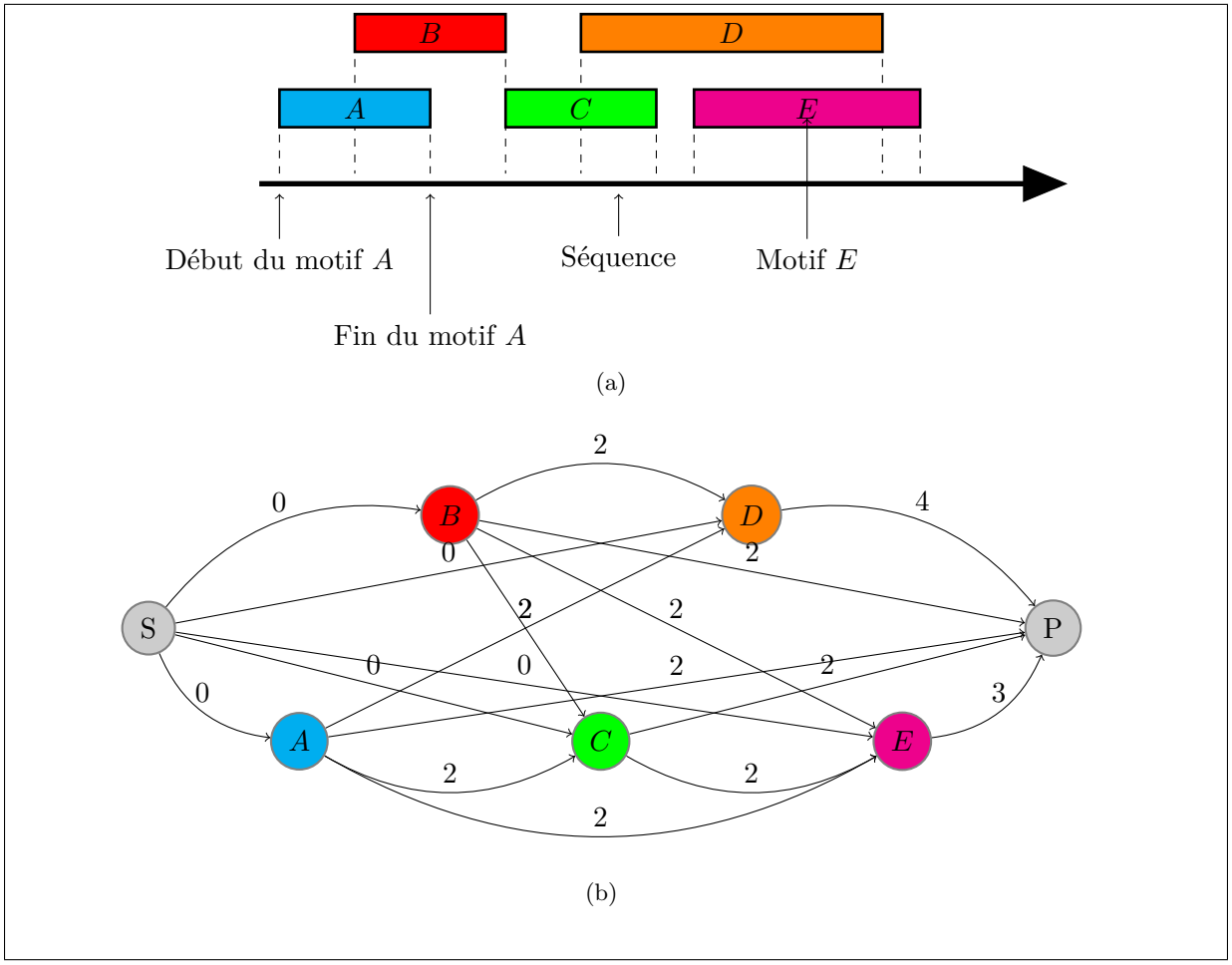


Figure 27 : Modélisation du problème de chaînage avec une séquence et un ensemble de motifs. (a) Modélisation du problème de chaînage 1D sur les séquences. On suppose que les scores des motifs A, B, C, D et E sont respectivement de 2, 2, 2, 4 et 3. Les chaînages possibles sont $\{A\}$, $\{B\}$, $\{C\}$, $\{D\}$, $\{E\}$, $\{A,C\}$, $\{A,D\}$, $\{A,E\}$, $\{A,C,E\}$, $\{B,C\}$, $\{B,D\}$, $\{B,E\}$ et $\{B,C,E\}$. (b) Une modélisation du problème de chaînage sous forme de graphe. Chaque motif est représenté par un nœud, les nœuds compatibles sont reliés par un arc qui est valué par le score du motif.

Chaînage 1D

Le chaînage que nous appelons 1D [GMP96] possède des applications, par exemple dans la recherche d'exons dans une séquence d'ADN [GMP96]. Nous disposons d'une séquence d'ADN S d'un gène eucaryote et d'un ensemble de m exons $E = \{E_1, E_2, \ldots, E_m\}$ candidats pour ce gène. Pour chaque exon, un score est associé. Ce score peut, par exemple, être égal au nombre de caractères contenus dans le motif. Le problème consiste à trouver la suite d'exons dont les occurrences sur S ne se chevauchent pas et dont la somme des scores est maximale. Sur Fig. 27(a) la séquence est représentée par une flèche, les motifs par des rectangles, la partie gauche (*resp.* droite) du rectangle correspond à la position de début (*resp.* fin) du motif sur la séquence. Les meilleurs chaînages possibles sur cet exemple sont au nombre de 2, les chaînages $\{A, C, E\}$ et $\{B, C, E\}$, ils ont un score de 7, les autres possédant un score inférieur.

Il est possible de calculer le chaînage en utilisant un graphe acyclique orienté (DAG) tel que chaque exon soit représenté par un nœud, deux nœuds sont reliés s'ils sont compatibles, c'est-à-dire s'ils respectent le même ordre et ne se chevauchent pas. Deux nœuds sont ajoutés, un nœud source avec un arc allant de ce nœud vers chaque autre nœud précédemment créé et un nœud puits avec un arc allant de chaque nœud vers ce nœud puits. Les arcs sont alors valués par le score du nœud d'origine. Sur Fig. 27(b) une modélisation du problème sous forme de graphe suivant ce principe. La recherche du plus long chemin (Déf. 1.11) du nœud source vers le nœud puits est alors en $O(n^2)$.

Une autre solution consiste pour chaque exon, à calculer le meilleur chaînage terminant par cet exon. L'algorithme 6 [GMP96] présente le chaînage 1D entre une séquence S de taille n et un ensemble de m motifs. Les positions de début et de fin de chaque exon (graine) sont triées selon leur ordre dans la séquence et stockées dans un tableau (lignes 1-2). Pour chaque exon le meilleur chaînage terminant par l'exon est stocké dans un tableau initialisé à 0 (ligne 3), de même le score du meilleur chaînage est initialisé à 0 (ligne 4). Le tableau est ensuite parcouru (ligne 5) :

- à chaque fois que l'on rencontre la position de début d'un exon, cet exon est chaîné avec le meilleur chaînage déjà calculé (lignes 6-7),
- si l'on atteint la position de fin d'un exon alors on regarde si son chaînage est le meilleur chaînage, s'il est meilleur alors la valeur du meilleur chaînage est mise à jour (lignes 8-9).

Le chaînage 1D réalisé selon cette technique possède une complexité en $O(m)$ en temps.

Algorithme 6 *Chaînage1DSéquence*(S, E) : Calcul du chaînage sur la séquence S.

Entrées : S : une séquence de taille n, $E = \{E_0, \ldots, E_{m-1}\}$: un ensemble de m motifs avec un score.

Sorties : Le score de la chaîne maximal dans S

1: $t \leftarrow$ un tableau contenant les positions de début et de fin de chaque graine
2: Trier t selon leur ordre d'apparition dans la séquence
3: Créer un tableau V de taille m, tel que $V[k]$ contiendra le meilleur score de chaînage terminant au motif E_k, initialisé à 0
4: maximum$\leftarrow 0$
5: **Pour tout** $i \in t$ **Faire**
6: **Si** i est la position gauche du motif E_k **Alors**
7: $V[k] \leftarrow score(E_k)$+maximum
8: **Sinon** $\{i$ est la position droite du motif $E_k\}$
9: maximum$\leftarrow max\{V[k]$,maximum$\}$
10: **Fin si**
11: **Fin pour**
12: **Retourner** maximum

Chaînage 2D

Le problème du chaînage 2D suppose que nous avons deux séquences S_1 et S_2 ainsi qu'un ensemble E de couples de sous-chaînes de S_1 et S_2 de fortes similarités [JMT92, OA05]. On suppose qu'un score est associé à chaque couple de E, par exemple, le nombre de caractères identiques dans les motifs. Pour modéliser le problème, les séquences S_1 et S_2 sont représentées dans le plan. Chaque graine est représentée par un rectangle mettant en correspondance deux sous-séquences de S_1 et S_2. Le problème consiste à trouver un sous-ensemble de E tel que les zones ne soient pas chevauchantes dans S_1, ni dans S_2, que l'ordre des sous-chaînes soit le même dans les deux séquences et que la somme des scores retenus soit maximale. La Fig. 28 cette modélisation du problème de chaînage 2D. Le meilleur chaînage est $\{B, E, D\}$, son score de 11.

L'Algorithme 7 résout le chaînage 2D sur les séquences de [OA05]. Comme dans le cas 1D les positions de début et de fin des graines sont mises dans un tableau qui est trié, étant donné que l'ordre peut être différent dans les deux séquences, le tri à lieu selon l'ordre sur une séquence (ligne 1). Sans perte de généralité et afin de rester consistant avec la Fig. 28 nous pouvons considérer qu'ils sont triés selon S_1. Une liste contenant un triplet avec la position de fin sur S_2 des graines, la valeur du meilleur chaînage se terminant sur la graine et la graine est triée selon les positions de fin croissante sur S_2 (lignes 2-3). Le tableau est ensuite parcouru (ligne 5) :

- à chaque fois que l'on rencontre la position de début d'une graine, cette graine est chaîné avec le meilleur chaînage déjà calculé (lignes 5-7),
- si l'on atteint la position de fin d'une graine alors on regarde si son chaînage est le meilleur chaînage, s'il est meilleur alors la valeur du meilleur chaînage est mise à jour et les anciens chaînages moins bons précédemment calculés sont supprimés (lignes 8-16).

D'un point de vue algorithmique, soient m graines, la chaîne optimale entre deux séquences peut être calculée en $O(m \log(m))$ en temps et $O(m)$ en espace [JMT92] (voir [OA05] pour une revue).

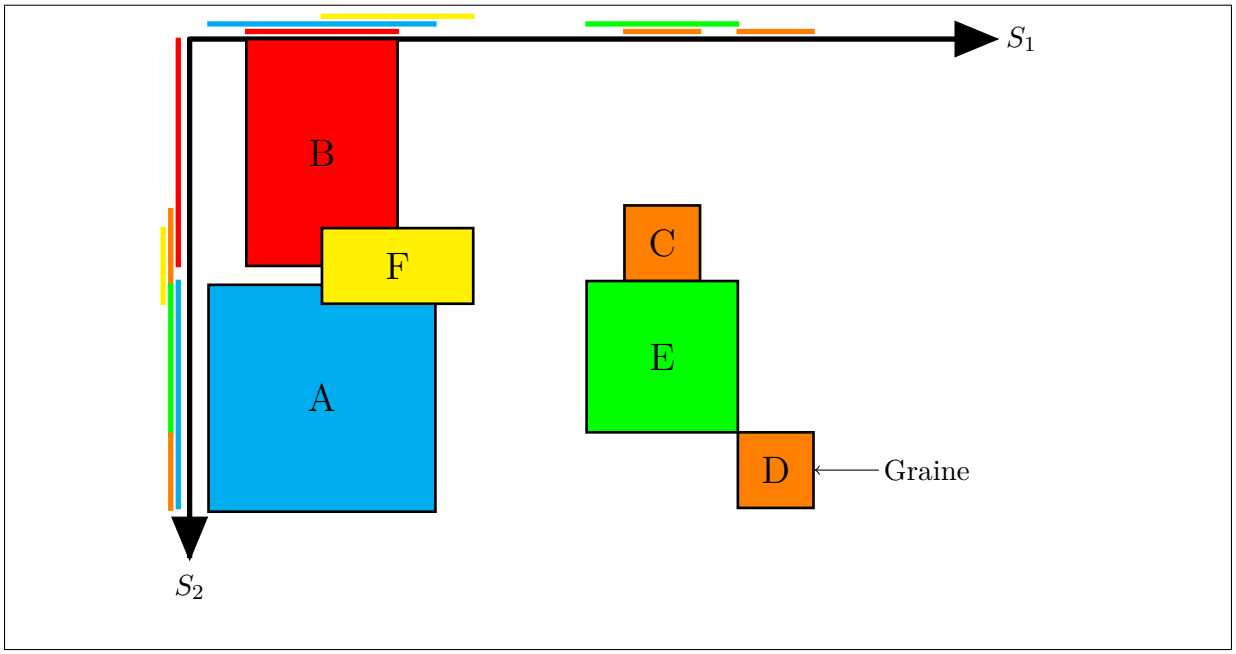


Figure 28 : Modélisation du problème de chainage 2D. Les scores associés aux graines $\{A\}$, $\{B\}$, $\{C\}$, $\{D\}$ et $\{E\}$ sont respectivement de 9, 6, 1, 1, 4 et 2. Les chaînages possibles sont $\{A\}$, $\{B\}$, $\{C\}$, $\{D\}$, $\{E\}$, $\{F\}$, $\{F,D\}$, $\{B,E\}$, $\{B,D\}$ et $\{B,E,D\}$.

Algorithme 7 $\hat{Chaînage2DSéquence}(S_1, S_2, E)$: Calcul du chaînage entre les séquences S_1 et S_2.

Entrées : S_1, S_2 : deux séquences, E : un ensemble de m motifs avec un score.
Sorties : Le score de la chaîne maximal entre S_1 et S_2.

1: $t \leftarrow$ un tableau avec les positions triées de début et de fin des graines sur la séquence S_1
2: $L \leftarrow$ une liste de triplets $(j_b, V[j], j)$ avec j_b la coordonnée de la position de fin de la graine j sur la séquence S_2 et $V[j]$ la valeur du meilleur chaînage se terminant par j
3: trier L selon les positions décroissantes
4: **Pour tout** $i \in t$ **Faire**
5: **Si** i est la position de début de la graine k **Alors**
6: Trouver dans L le dernier triplet $(j_b, V[j], j)$ tel que $j_b < i$ {j est au-dessous de k}
7: Positionner $V[k]$ à $v_k + V[j]$
8: **Sinon** {i est la position de fin de k}
9: Rechercher le dernier triplet $(j_b, V[j], j)$ de t tel que $j_b \leq i$
10: **Si** $V[k] > V[j]$ **Alors**
11: Ajouter $(k_b, V(k), k)$ dans L
12: **Fin si**
13: Retirer de L les triplets j' tels que $j'_b \leq k_b$ et $V[k] > V[j']$ {rectangles plus hauts avec un score plus faible}
14: **Fin si**
15: **Fin pour**
16: **Retourner** La valeur maximal de V dans L

Le principe de filtrage a été très étudié pour les séquences, par exemple les méthodes BLAST et FASTA sont abondamment utilisées. Afin de mettre en place des méthodes de recherche rapides pour les structures arborescentes, nous proposons dans les sections suivantes une méthode de chaînage des graines sur les arborescences et une définition des graines.

4.2 Chaînage de graines dans des arborescences ordonnées

Récemment, plusieurs approches ont été proposées pour filtrer de larges bases de données de structures secondaires d'ARN. Janssen *et al.* [JRG08] proposent d'indexer les ARN en utilisant leur formes, une vue simplifiée de leurs structures secondaires [SVR+06], cependant cette approche ne dépend pas d'un calcul de distance d'édition. Basée sur un principe de programmation dynamique, Heyne *et al.* [HWBB09] ont introduit le problème de chaînage avec une représentation alternative des arborescences ordonnées appelée séquences arc-annotées, motivés par les comparaisons deux à deux de structures secondaires d'ARN : une fois qu'une chaîne optimale de graines entre deux structures secondaires d'ARN a été détectée, les mises en correspondance des régions entre deux graines successives sont réalisées indépendamment de l'algorithme de distance d'édition, ce qui améliore significativement le processus de comparaison. Heyne *et al.* [HWBB09] considèrent que les graines définies sont comme des motifs exacts communs et désignés par un algorithme de programmation dynamique pour résoudre le problème de chaînage de graines. À notre connaissance [HWBB09] est le premier papier traitant du problème du calcul de chaînage dans des arborescences (voir également [LPR+08]). Cependant lorsqu'il est appliqué pour chaîner des graines dans les séquences, leur algorithme a une complexité plus élevée que le meilleur algorithme connu pour les séquences, ce qui pose la question d'un algorithme plus efficace, sur un intérêt théorique et pratique.

Après quelques préliminaires (sections 4.2.1, 4.2.2, 4.2.3), décrivant les propriétés combinatoires des chaînes et le prétraitement de l'algorithme, nous décrivons dans la Section 4.2.4 les algorithmes pour trouver une chaîne avec un score optimal entre deux arborescences ordonnées (Maximal Chaining Problem), nous continuerons dans la Section 4.2.5 par une analyse de la complexité notamment une comparaison avec l'algorithme de Heyne *et al.* [HWBB09]. Nous conclurons, Section 4.2.6, par une description de l'adaptation de notre algorithme principal pour la résolution d'un problème plus simple, celui du chaînage de graine au sein d'une seule arborescence ordonnée.

4.2.1 Formalisation du problème

Dans ce chapitre, nous considérons T une arborescence ordonnée de taille n, les nœuds de T sont identifiés par leur index dans ordre postfixe de 0 à $n-1$, l'index $n-1$ représente la racine de T et T_i est la sous-arborescence de T enracinée en i. Nous notons par $T[i,j]$ la forêt induite par les nœuds qui appartiennent à l'intervalle $[i,j]$; si $i > j$, alors $T[i,j]$ est vide. La relation partielle « i est ancêtre de j » est notée $i \prec j$. Pour une arborescence T et un nœud i de T, la première (*resp.* dernière) feuille visitée lors d'un parcours postfixe de T_i est notée $l(i)$ (*resp.* $r(i)$) et est appelée la *feuille la plus à gauche* (*resp. feuille la plus à droite*) du nœud i. La forêt ordonnée induite par les propres descendants de i est notée $\widehat{T}_i = T[l(i), i-1]$.

Graines, chaînes et le problème de chaînage maximum

La notion d'*arborescence interne* (Déf. 4.1) permet de définir les graines sur les arborescences.

Définition 4.1 (Arborescence interne) Soit T une arborescence ordonnée :

1. Soit $G = \{g_0, \ldots, g_{k-1}\}$ un ensemble ordonné de k nœuds de T, avec $0 \leq g_j < n$.
   - Si le sous-graphe de T induit par G est connexe, alors G est appelé un *arborescence interne* enracinée en g_{k-1} également noté r_G.
   - S'il existe une partition de G dans p arborescences internes $G_1, \ldots G_p$ respectivement enracinés en r_{G_i} et tel que pour tout $i > 1$, r_{G_i} est le voisinage (Déf. 1.8) droit de $r_{G_{i-1}}$ dans T, alors G est appelé une *forêt interne* (ainsi, toute arborescence interne est aussi une forêt interne).
   - r_{G_p} (la racine de l'arborescence la plus à droite de G) est appelée la racine de G, notée par r_G.
2. La feuille la plus à gauche de la forêt interne G est définie par $l(G) = min(l(x)/x \in G)$.
3. L'ensemble de feuilles de la forêt interne G est noté $L(G)$.
4. Un nœud g_j d'une forêt interne G est *complètement inclus dans* G si g_j n'est pas une feuille de T et si tous ses enfants appartiennent à G. L'ensemble des nœuds de G qui ne sont pas complètement inclus dans G est appelé le *bord* de G et est noté $B(G)$.
5. Deux forêts internes G^1 et G^2 sont *chevauchantes* si $G^1 \cap G^2 \neq \emptyset$.

Une illustration d'arborescence interne est présentée Fig. 29.

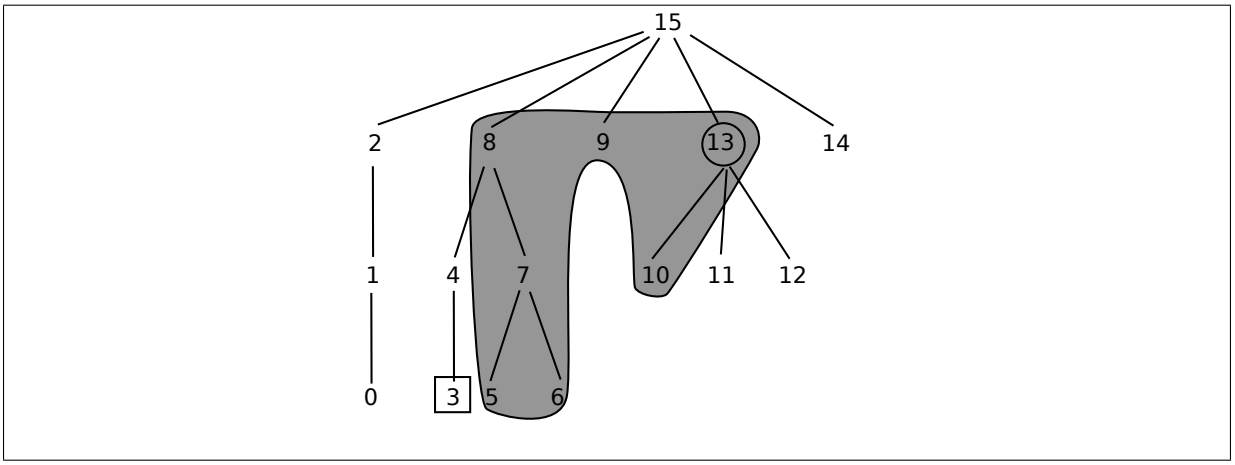


Figure 29 : Exemple d'une forêt interne $G = \{5,6,7,8,9,10,13\}$ contenant trois arborescences internes $G_1 = \{5,6,7,8\}$, $G_2 = \{9\}$ et $G_3 = \{10,13\}$. $r_G = 13$. $r_{G_1} = 8$, $r_{G_2} = 9$, $r_{G_3} = 13$. $L(G) = \{5,6,10\}$. Le nœud 7 est complètement dans G. $B(G) = \{5,6,8,9,10,13\}$. $l(G) = 3$.

Soit une mise en correspondance P entre Q et T, la plus petite forêt interne de Q (*resp.* T) qui contient tous les nœuds de Q (*resp.* T) appartenant à la paire de P est notée avec Q_P (*resp.* T_P). Q_P et T_P sont respectivement appelées les forêts internes de Q et T induites par P.

La définition suivante introduit la notion centrale de *graines* entre deux arborescences ordonnées (Déf. 4.2). Cette définition est plus large que celle présentée dans la section suivante. Autrement dit, les graines sont des forêts internes, une dans chaque arborescence, avec une structure similaire en terme de nombre dans l'arborescence et dans les bords.

Définition 4.2 (Graine) Soient Q et T deux arborescences ordonnées.

1. Une *graine* P entre Q et T est une mise en correspondance entre Q et T telle que :
   – Les deux forêts internes Q_P et T_P contiennent le même nombre t d'arborescences internes, de racines respectives (dans l'ordre postfixe croissant) $r_{Q_{P_1}},\ldots,r_{Q_{P_t}}$ dans Q et $r_{T_{P_1}},\ldots,r_{T_{P_t}}$ dans T.
   – Pour tous $0 \leq i < t$, $(r_{Q_{P_i}}, r_{T_{P_i}}) \in P$.
   – Chaque nœud du bord de Q_P (*resp.* T_P) appartient à une paire de P.
2. Le *bord* $B(P)$ (*resp.* les feuilles $L(P)$) de la graine P est un ensemble de paires $(x,y) \in P$ telles que $x \in B(Q_P)$ et $y \in B(T_P)$ (*resp.* $x \in L(Q_P)$ et $y \in L(T_P)$).
3. La *taille* $|P|$ de la graine P est le nombre de paires que ses mises en correspondance contiennent.
4. Pour un ensemble S de graines, $\|S\|$ est la somme des tailles des $|S|$ graines dans S.

Notons que, théoriquement, le nombre de graines entre Q et T peut-être exponentiel en la taille de Q et T, bien que dans les applications telles que la comparaison de structure secondaire d'ARN cette borne supérieure exponentielle est hors de portée (voir par exemple [HWBB09]).

Voyons maintenant la définition de graines *chaînables* (Déf. 4.3).

Définition 4.3 (Chaînable) Soient Q et T deux arborescences ordonnées :

1. Une paire (P^1,P^2) de graines entre Q et T est *chaînable* si Q_{P^1} n'est pas chevauchante avec Q_{P^2}, T_{P^1} n'est pas chevauchante avec T_{P^2}, et $P^1 \cup P^2$ est une mise en correspondance.
2. Une *chaîne* est un ensemble $C = \{P^0,P^1,\ldots,P^{\ell-1}\}$ de graines entre Q et T tel que toute paire (P^i,P^j) de graines distinctes dans C est chaînable.
3. Étant donné une fonction de score v pour les graines P^i, le score de la chaîne C est la somme des scores de ses graines : $v(C) = \sum_i v(P^i)$.
4. Étant donné un ensemble S de graines possible chevauchantes entre Q et T, $\mathcal{C}_S(Q,T)$ représente l'ensemble de toutes les chaînes possibles entre Q et T incluses dans S.

Nous pouvons maintenant définir le problème que nous traitons dans ce travail.

Considérons un ensemble $S = \{P^0,\ldots,P^{m-1}\}$ de m chevauchements possibles de graines entre Q et T, $\mathcal{C}_S(Q,T)$ représente l'ensemble de toutes les chaînes possibles entre Q et T incluses dans S.

Dans un souci de clarté, Q et T sont des arborescences ordonnées mais le présent travail est également valide sur des forêts ordonnées en ajoutant, par exemple, une racine fictive.

Problème : Problème de chaînage maximum (MCP) :
Entrée : Une paire (Q,T) d'arborescences enracinées ordonnées, un ensemble $S = \{P^0,\ldots,P^{m-1}\}$ de m graines possiblement chevauchantes entre Q et T, une fonction de score v sur les graines P^i.
Sortie : La chaîne de score maximal C incluse dans S :

$$MCP(Q,T,S) = \max\{v(C); C \in \mathcal{C}_S(Q,T)\}.$$

La Fig. 30 illustre le MCP avec un exemple de 6 graines.

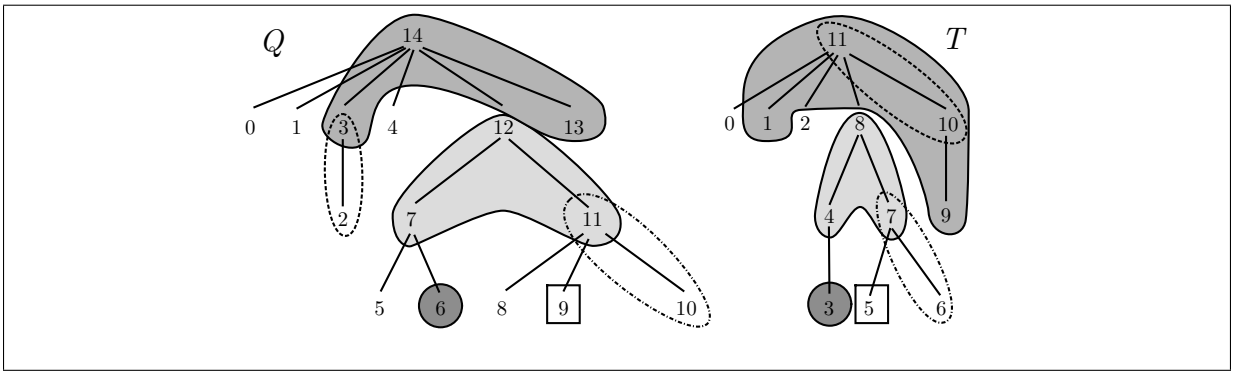


Figure 30 : Une instance du MCP avec 6 graines : $P^0 = \{(2,10),(3,11)\}$, $P^1 = \{(6,3)\}$, $P^2 = \{(9,5)\}$, $P^3 = \{(10,6),(11,7)\}$, $P^4 = \{(7,4),(11,7),(12,8)\}$, $P^5 = \{(3,1),(13,9),(14,11)\}$. Si pour chaque graine $v(P^i) = |P^i|$. La chaîne optimale est composée des graines $\{P^1, P^2, P^4, P^5\}$ et a un score de 8.

Remarque 13 Sans perte de généralité, nous supposerons que les graines P^i sont triées selon l'ordre postfixe de leurs racines dans Q, c'est-à-dire : $r_{Q_{P^0}} \leq \cdots \leq r_{Q_{P^i}} \leq \cdots \leq r_{Q_{P^{m-1}}}$. De plus, pour chaque (x,y) appartenant à la graine P^j, nous notons x^j l'unique nœud y de T associé avec x dans P^j. Nous notons également $r_{Q_{P^j}}$ par r_j, et après $r_{T_{P^j}}$ par r_j^j. Pour une chaîne donnée C, la *d*ernière graine de C est alors la graine avec un plus grand index postfixe dans Q.

Remarque 14 La notion de mise en correspondance s'étend naturellement pour les forêts ordonnées. Donc si S est un ensemble de graines telles que chaque graine est une graine entre une arborescence de F_1 et une arborescence de F_2, alors le MCP peut naturellement être étendu aux forêts ordonnées.

Remarque 15 Notons que nous considérons ici seulement le problème de calcul de score d'une chaîne pour des raisons de présentation ; les techniques de backtracking standard permettent de calculer une chaîne de score maximum depuis notre algorithme de programmation dynamique.

Pour une chaîne donnée C, la *dernière graine* est une graine de C qui a une racine avec le plus large numéro postfixe (à la fois dans Q mais également dans T par définition d'une chaîne). Nous étendons naturellement la notion de chaîne à deux forêts, définie par ajout d'une racine à chacune des deux forêts.

Motivation et résultats À notre connaissance, [HWBB09] est le seul travail attaquant le MCP bien que les auteurs le décrivent en terme de séquences arc-annotatées. Ils proposent un algorithme de programmation dynamique pour résoudre le problème de chaînage maximum avec quelques restrictions (précisément les graines sont des motifs maximums exacts communs pour considérer les séquences). Cette technique de programmation dynamique est différente de l'approche utilisée pour le meilleur algorithme connu pour le problème équivalent sur les séquences [JMT92, OA05]. De plus, quand elle est appliquée aux séquences arc-annotées sans arc (c'est-à-dire aux séquences) et m graines, il peut être montré que l'algorithme a une complexité en pire cas en $O(m^2)$ (voir Section 4.2.5). Notre résultat est le suivant :

Théorème 1 Soit S un ensemble de m graines entre deux arborescences ordonnées Q et T. Après un prétraitement en $O(\|S\|)$ des m graines de S, le problème de chaînage maximum peut être réalisé en $O(\|S\|\log(\|S\|) + m\|S\|\log(m))$ en temps et en $O(m\|S\|)$ en espace.

Preuve La preuve est donnée Section 4.2.4.□

Remarque 16 Afin de comparer la méthode de chaînage sur les séquences et sur les arborescences, nous représentons une séquence $u = (u_0, \ldots, u_{n-1})$ par une arborescence unaire, enracinée dans le nœud u_{n-1}, dans laquelle chaque nœud ne possède qu'un unique enfant et u_0 est l'unique feuille. Le parcours postfixe de l'arborescence est alors exactement u.

4.2.2 Propriétés des graines et des chaînes

Nous décrirons les propriétés combinatoires des graines et des chaînes, qui amènent naturellement à un schéma récursif qui calcule une chaîne maximum. Plus précisément, nous montrons que pour une chaîne C et sa dernière graine P, les racines et bords de P définissent une partition de $Q - Q_P$ et de $T - T_P$ en un ensemble de paires de forêts, définie en terme de notion clé de *zones chaînables* (Définitions 4.4 et 4.5), qui contiennent les graines $C - \{P\}$ et forment des sous-chaînes de C.

Définition 4.4 (Zone maximale chaînable) Soit P une graine entre deux arborescences Q et T et $(a,b;c,d)$ un quadruplet tel que $l(Q_P) \leq a < b < r_{Q_P}$, $l(T_P) \leq c < d < r_{T_P}$, $Q_P \cap Q[a,b] = \emptyset$ et $T_P \cap T[c,d] = \emptyset$. $(a,b;c,d)$ est une *zone chaînable sur* P si, pour tout $i \in [a,b]$ et tout $j \in [c,d]$, $P \cup (i,j)$ est une mise en correspondance valide, c'est une *zone maximale chaînable de* P si ni $(a-1,b;c,d)$, ni $(a,b+1;c,d)$, ni $(a,b;c-1,d)$, ni $(a,b;c,d+1)$ sont des zones chaînables pour P.

Par exemple, dans la Fig. 30, si $P = P^5$, alors $(4,12;2,8)$ est l'aire maximum de chaînage. Voir également la Figure 31 ci-dessous.

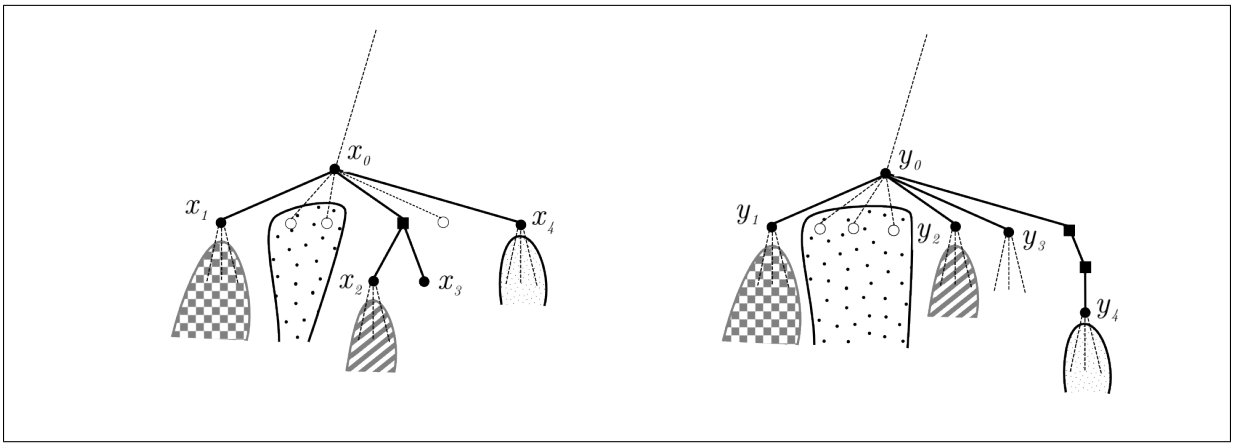


Figure 31 : Illustration de la notion de zones chaînables d'une graine $P = \{(x_0,y_0), \ldots, (x_4,y_4)\}$ de taille 5. La graine P possède 4 zones de chaînage chacune indiquée par un motif différent.

Nous allons maintenant affiner la définition d'une zone chaînable pour définir précisément les zones associées à la frontière des nœuds de la graine. De telles zones sont fondamentales pour mettre en correspondance des nœuds de P et de Q qui appartiennent à la chaîne et qui doivent donc appartenir à ces zones chaînables.

Définition 4.5 (Zones chaînables) 1. Soient $(x,y) \in B(P)$ d'une graine P entre Q et T. Nous définissons $F(x,y) = \{(a_i,b_i;c_i,d_i)\}$ comme l'ensemble de toutes les zones chaînes de

P incluses dans $(Q_x;T_y)$ telles qu'il n'y a pas de nœud du bord des nœuds de P dans Q (*resp.* T) sur le chemin de b à x (*resp.* d à y). Nous appelons cet ensemble *zone chaînable de* (x,y).

2. Les *zones chaînables* de la graine P, notées $CA(P)$, sont l'union des ensembles de quadruplets $F(x,y)$ pour toutes les paires $(x,y) \in B(P)$.

Par exemple, considérons une paire $(x,y) \in L(P)$ telle que x et y ne soient pas des feuilles respectivement dans Q et T, alors $F(x,y)$ représente le couple de forêts $\widehat{Q_x}$ et $\widehat{T_y}$, $F(x,y) = \{(l(x),x-1;l(y),y-1)\}$. Dans la Fig. 30, avec $P = P^4$ et $(x,y) = (11,7)$, $F(x,y) = \{(8,10;5,6)\}$; avec $P = P^5$, si $(x,y) = (14,11) \in B(P^5) - L(P^5)$, $F(x,y) = \{(0,1;0,0),(4,12;2,8)\}$.

La propriété suivante est une conséquence directe de la définition de graines et de zones chaînables.

Propriété 2 Soit une graine P entre deux arborescences Q et T alors $|CA(P)| \leq 2 \times |B(P)| - 1$.

Preuve Les zones chaînables sont avant le bord gauche, entre chaque bord et après le dernier bord.□

Notation. Pour une graine P et une zone chaînable $(a,b;c,d)$, nous disons que $P \subset (a,b;c,d)$ si $a \leq l(Q_P) \leq r_{Q_P} \leq b$ et $c \leq l(T_P) \leq r_{T_P} \leq d$; pour une chaîne C, $C \subset (a,b;c,d)$ si toutes ces graines sont incluses dans $(a,b;c,d)$. Nous notons finalement $F_j(x)$ l'ensemble des quadruplets $F(x,x^j)$ pour une paire de nœuds $(x,x^j) \in B(P^j)$.

La propriété suivante décrit la structure de toutes les chaînes, entre deux forêts $Q[a,b]$ et $T[c,d]$, incluses dans un ensemble de m graines $S = \{P^0,\ldots,P^{m-1}\}$.

Propriété 3 Soit P^j la dernière graine de la chaîne C entre deux forêts $Q[a,b]$ et $T[c,d]$.

1. C peut-être décomposé en $|CA(P^j)|+2$ (possiblement vides) sous-chaînes distinctes :
   - P^j lui-même,
   - pour chaque $(e,f;g,h) \in CA(P^j)$ une chaîne (possiblement vide) entre $Q[e,f]$ et $T[g,h]$,
   - et une chaîne entre $Q[a,l(Q_{P^j})-1]$ et $T[c,l(T_{P^j})-1]$.
2. C est une chaîne de score maximum sur toutes les chaînes dans $Q[a,b]$ et $T[c,d]$ qui contient P^j si et seulement si chacune de ses sous-chaînes décrites ci-dessus ont un score maximum dans la forêt correspondante définie par P^j.

Preuve La propriété est une conséquence directe de la contrainte qui définit une mise en correspondance valide et le fait que les graines sont non-chevauchantes dans une chaîne.□

Le point 2 de la Propriété 3 conduit naturellement à un schéma récursif pour calculer la chaîne optimale entre deux forêts $Q[a,b]$ et $T[c,d]$ qui terminent par la dernière graine de l'ensemble. Si on note $MCP'(Q[a,b],T[c,d],\{P^0 \ldots P^j\})$ le score de la chaîne maximum entre $Q[a,b]$ et $T[c,d]$ qui contient P^j comme dernière graine alors :

$$MCP'(Q[a,b],T[c,d],\{P^0 \ldots P^j\}) = \tag{4.1}$$

$$\begin{cases} 0 & \text{si } P^j \not\subset (a,b;c,d), \\ v(P^j) + \sum\limits_{(e,f;g,h)\in CA(P^j)} MCP(Q[e,f],T[g,h],\{P^0 \ldots P^{j-1}\}) & \\ \quad + MCP(Q[a,l(Q_{P^j})-1],T[c,l(T_{P^j})-1],\{P^0 \ldots P^{j-1}\}) & \text{sinon.} \end{cases}$$

et, assure que les graines sont triées de façon incrémentale (voir remarque 13), $MCP(Q,T,S)$ peut être calculé en utilisant MCP' comme suit :

$$MCP(Q[a,b],T[c,d],\{P^0 \dots P^j\}) = \max_{i=0\dots j} MCP'(Q[a,b],T[c,d],\{P^0 \dots P^i\}) \quad (4.2)$$

$$MCP(Q,T,S) = MCP(Q[0,r_Q],T[0,r_T],S) \quad (4.3)$$

Le principal point pour réaliser un algorithme pour le MCP est d'implémenter efficacement cette formule de récurrence qui était déjà centrale dans l'algorithme de programmation dynamique de [HWBB09].

4.2.3 Un algorithme de calcul des zones chaînables des graines

Le coût du calcul des zones chaînables des nœuds du bord d'une graine dépend de la nature de cette graine. Nous décrivons ici un algorithme efficace calculant $F(x,y)$.

Soit P une graine entre deux arborescences Q et T et soit $B(P)$ l'ensemble des paires des nœuds de son bord. Pour chaque nœud i de Q et T (en fait, sont seulement requis les nœuds des bords de P), nous supposons que nous avons accès aux informations suivantes en $O(1)$:

- $l(i)$: la feuille la plus à gauche de i.
- $u(i)$: le nœud avec le plus haut index tel que $r(u(i)) = r(i)$ où $r(i)$ est la feuille le plus à droite de i.

Les nœuds $u(i)$ sont parfois référencés comme la racine la plus à droite de l'arborescence.

Dans l'Algorithme 8, nous utilisons B à la place de $B(P)$ et nous supposons que B est un tableau de k paires de nœuds dans $Q \times T$. Pour que $0 \leq i < k$, $B[i]$ représente la $(i+1)^{\text{ième}}$ paire de B et que $B[i]_Q$ soit le nœud Q de cette paire et $B[i]_T$ est le nœud de T de cette paire.

L'Algorithme 8 utilise une pile de paires de nœuds appelée *Pile*. $haut(Pile)$ fait référence au dernier élément inséré dans *Pile* et de la même façon que pour B, $haut(Pile)_Q$ et $haut(Pile)_T$ sont le nœud de Q et le nœud de T dans $haut(Pile)$. Nous notons $empiler(Pile,(x,y))$ pour ajouter (x,y) en haut de la pile et $dépiler(Pile)$ afin de supprimer le dernier élément de la pile.

L'algorithme qui calcule $F(x,y)$ pour toutes les paires des nœuds du bord (x,y) de P est présenté dans l'Algorithme 8.

Description de l'algorithme Dans la suite, une paire de nœud du bord de P est appelée une *paire*. Les paires sont parcourues progressivement selon l'ordre postfixe de leurs index. Donc, les descendants sont visités avant leurs parents. Rappelons qu'une graine est une mise en correspondance valide, alors les ancêtres et les relations sur l'ordre entre les nœuds du bord sont respectés. Ainsi, si un nœud du bord est une feuille dans P_Q, c'est également une feuille dans P_T.

Avant chaque insertion d'une nouvelle zone de chaînage dans $F(x,y)$, nous testons si la zone chaînable est vide ou non (*cf.* lignes 7, 13, 19, 13 de l'Algorithme 8).

Appelons le descendant direct de la paire (x,y), la paire $(x',y') \in B(P)$ telle que x' est un descendant de x (*resp.* y' est un descendant de y) et qui n'est pas un nœud du bord de P dans Q (*resp.* T) entre x' et x (*resp.* y' et y).

Sauf pour la dernière paire, chaque fois qu'une paire est visitée, elle est ajoutée à la *Pile* car il est nécessaire de traiter le descendant direct d'une paire non visitée. Notons que *Pile* contient les paires triées incrémentalement par leur index postfixe.

Deux cas doivent être considérés :

1. la paire est une paire de feuilles dans Q_P, T_P ($(x,y) \in L(P)$) et
2. la paire n'est pas une paire de feuilles dans Q_P,T_P ($(x,y) \notin L(P)$).

Les lignes 6–8 correspondent au premier cas et ne nécessitent pas d'explications supplémentaires.

Pour le second cas, la paire courante (x,y) a nécessairement un descendant direct dans $B(P)$. Ces descendants ont été visités (index postfixe inférieur) et sont donc dans *Pile*. La zone chaînable (a,b,c,d) (possiblement vide) sur la droite de son descendant direct le plus à droite (x',y') ($a > x'$ et $c > y'$) et la zone chaînable (possiblement vide) (a,b,c,d) à gauche de son descendant le plus à gauche (x'',y'') ($b < x''$ et $d < y''$) requièrent un traitement particulier. Les possibles zones chaînables entre deux descendants directs sont considérées dans la boucle des lignes 16–17. Pour calculer ces zones chaînables, les propriétés suivantes sont utilisées dans l'algorithme : soit $(x,y) \in B(P)$

1. Toutes les zones chaînables (a,b,c,d) de (x,y) sont telles que b et d sont des enfants de x et y et a et c sont les feuilles les plus à gauche des enfants de x et y.
2. Par définition, pour les zones chaînables (a,b,c,d) de (x,y) sauf celle sur la droite du descendant de la plus à droite, b et d sont tels que $b+1$ et $d+1$ sont les feuilles les plus à gauche du descendant le plus à gauche de x et y.
3. Pour les zones chaînables (a,b,c,d) de (x,y) sauf celle à gauche du descendant le plus à gauche, a et c sont tels que $a-1$ et $c-1$ sont des enfants de x et y et sont soit des nœuds du bord, soit des ancêtres des nœuds du bord.

Comme *Pile* est trié incrémentalement, le haut de la pile contient le descendant direct le plus à droite de la paire courante. Les lignes 13–14 calculent les zones chaînables *à droite* de ce descendant. Après, la boucle aux lignes 16–16 calcule la zone entre les descendants directs en utilisant les propriétés ci- dessus. Finalement, les lignes 24– 25 calculent la zone chaînable sur *la gauche* le descendant direct le plus à gauche (qui est la dernière paire (x',y') dans la *Pile* tel que $x' \geq l(x)$ et $y' \geq l(y)$). Remarquons que tous les descendants directs sont maintenant sortis de la *Pile* et sont replacés par la paire courante.

La complexité en temps de cet algorithme est $O(|B(P)|)$ comme nous itérons sur chaque paire et chaque paire est ajoutée seulement une fois à la *Pile*.

Notons que notre algorithme est général et peut être appliqué à chaque ensemble de graines suivant la Déf. 4.2. Quand nous considérons les familles de graines restreintes, il est possible de le décrire plus simplement, toujours efficacement, les algorithmes pour calculer les $F(x,y)$ et les zones chaînables. Par exemple, si on considère seulement les *graines compactes*, c'est-à-dire les graines telles que $B(P) = L(P)$ pour chaque graine P, alors pour chaque bord (x,y), $|F(x,y)| = 1$ et le calcul requiert un temps linéaire en le nombre de graines.

Algorithme 8 $F(x,y)$: Calcul $F(x,y)$ pour une graine P.

Entrées : TODO Q,T : deux arborescences ordonnées, S : un ensemble de m graines et v : une fonction de score sur S.

Sorties : TODO Le score de la chaîne maximal dans S

1: Trier B de façon incrémentale
2: **Pour tout** paire $B[i]$ de B **Faire**
3: $F(B[i]) \leftarrow \emptyset$
4: **Fin pour**
5: **Pour** i de 0 à $k-1$ **Faire**
6: **Si** $i=0$ ou $B[i-1]_Q < l(B[i]_Q)$ **Alors** $\{B[i] \in L(P)\}$
7: **Si** $l(B[i]_Q) \leq B[i]_Q - 1$ et $l(B[i]_T) \leq B[i]_T - 1$ **Alors** $\{B[i]_Q$ n'est pas une feuille dans Q et $B[i]_T$ n'est pas une feuille dans $T\}$
8: $F(B[i]) \leftarrow (l(B[i]_Q), B[i]_Q - 1; l(B[i]_T), B[i]_T - 1$
9: **Fin si**
10: **Sinon** {c'est-à-dire $F(B[i])$ est vide}
11: $(x,y) \leftarrow haut(Pile)$
12: *dépiler* $(Pile)\{x$ est un descendant de $B[i]_Q\}$
13: **Si** $u(x)+1 \leq B[i]_Q - 1$ et $u(y)+1 \leq B[i]_T - 1$ **Alors** {Calcul de la zone chaînable la plus à droite de $B[i]\}$
14: $F(B[i]) \leftarrow F(B[i]) + (u(x)+1, B[i]_Q - 1; u(y)+1, B[i]_T - 1)$
15: **Fin si**
16: **Tant que** *Pile* n'est pas vide et $haut(Pile)_Q \geq l(B[i]_Q)$ **Faire** {Calcul des zones intermédiaires de chaînage}
17: $(s,t) \leftarrow haut(Pile)$
18: $dépiler(Pile)$
19: **Si** $u(s)+1 \leq l(x)-1$ and $u(t)+1 \leq l(y)-1$ **Alors**
20: $F(B[i]) \leftarrow F(B[i]) + (u(s)+1, l(x)-1; u(t)+1, l(y)-1)$
21: $(x,y) \leftarrow (s,t)$
22: **Fin si**
23: **Fin tant que**
24: **Si** $l(x)-1 \geq l(B[i]_Q)$ and $l(y)-1 \geq l(B[i]_T)$ **Alors** {Calcul de la zone chaînable la plus à gauche de $B[i]\}$
25: $F(B[i]) \leftarrow F(B[i]) + (l(B[i]_Q), l(x)-1; l(B[i]_T), l(y)-1)$
26: **Fin si**
27: **Fin si**
28: $empiler(Pile, B[i])$
29: **Fin pour**

4.2.4 Un algorithme de résolution du problème de chaînage maximum

Nous décrivons une première mais non optimale implémentation de la résolution de ce problème (Section 4.2.4), basée sur les propriétés décrites dans la section précédente, nous suivrons avec une implémentation plus efficace (Section 4.2.4).

Nous supposons que nous avons deux arborescences ordonnées Q et T, un ensemble $S = \{P^0, \ldots, P^{m-1}\}$ de graines fermées, avec $P^0 = (r_Q, r_T)$ (qui contient une unique paire composée

des racines de Q et T), et que le score $v(j)$ d'une graine P^j peut être accédé en temps constant. Nous supposons également que les graines P^j sont données dans la liste I de paires (i,r,j) où i est le nombre postfixe du nœud de Q qui est soit une racine, soit un bord de la graine P^j et r indique si i est un bord, une raciné ou les deux dans P^j ; comme un prétraitement, nous trions I en ordre croissant dans l'arborescence de référence (le nombre postfixe dans Q). Nous considérons également que la liste $F_{P^j}(b)$ pour tous les nœuds du bord des graines est disponible et que pour les nœuds Q appartenant à la graine P^j, le nœud correspondant dans T, i^j (ou plus précisément sont nombre postfixe dans T) peut être accédé en temps constant. Finalement, nous supposons que chaque nœud i dans Q et T, sa feuille la plus à gauche $l(i)$ peut être accédé en temps constant. Dans les deux algorithmes, nous visitons les éléments de I, et une graine est dite *visitée* après que sa racine ait été visitée.

Une implémentation simple

La première implémentation, a pour principal intérêt d'être très simple car elle ne nécessite pas de structure de données spéciale.

Le point clé est de traiter la requête pour trouver les sous-chaînes maximales en $O(1)$ en temps. Afin d'obtenir cette complexité désirée, nous utilisons deux structures de données : un tableau V de m entiers et un tableau M, indexé par le quadruple d'entiers (a,b,c,d). Nous utilisons la fonction `Mise à jour` qui prend une entrée $M[a,b,c,d]$ et une valeur w et remplace la valeur de cette entrée par w si et seulement si $w > M[a,b,c,d]$. La correction de l'algorithme est liée aux invariants suivants :

M1. $M[a,b,c,d]$ est la valeur de la chaîne maximale incluse dans $(Q[a,b],T[c,d])$ composée uniquement des graines visitées.

V1. $V[j]$ est le score de la meilleure chaîne formée de P^j et des graines visitées telles que $Q_{r_j} \in Q$ et $T_{r_j^j} \in T$.

Afin de prouver la correction de l'algorithme, nous devons prouver que la propriété V1 est satisfaite, ce qui implique que $\max_j V[j]$ contient le score de la chaîne maximale. D'après les propriétés 3 et le fait que V est mis à jour seulement ligne 10, on en déduit que V1 est satisfait si M1 est satisfait pour toutes les entrées $M[a,b,c,d]$ telle que $(a,b,c,d) \in F_{P^j}(i)$. Alors nous avons juste besoin de montrer que M1 est satisfait pour les quadruplets (a,b,c,d) qui appartiennent à $F_{P^j}(i)$ pour les nœuds i de Q bords de la graine P^j. De plus, une entrée M nécessite seulement d'être modifiée quand une nouvelle graine qui est contenue dans les forêts définies dans Q et T est visitée, ce qui est fait sur les lignes 6-11.

La complexité en espace est en $O(m||S||)$, compte tenu de l'espace requis pour encoder M, comme chaque entrée est indexée soit par un quadruplet $(l[v],r_g,l[v^j],r_g^g)$ pour un nœud du bord v d'une graine P^j et racine r_g d'une graine P^g, soit par un quadruplet $(l[i],i-1,l[i^j],i^j-1)$ dans lequel i est un nœud du bord. La complexité en temps est en $O(m^3+||S||\log(||S||))$. Le terme m^3 est conséquence de la double boucle des lignes 7-11 qui peut au maximum faire $O(m^2)$ itérations (due au fait que les graines sont compactes) et est réalisé une fois pour chaque racine des m graines. Le $||S||\log(||S||)$ et vient du tri du I.

Algorithme 9 $MCP_1(Q,T,S,v)$: Calcul le score de la chaîne maximal.

Entrées : Q,T : deux arborescences ordonnées, S : un ensemble de m graines et v : une fonction de score sur S.

Sorties : Le score de la chaîne maximal dans S

Pour j de 0 à $m-1$ **Faire**
  $V[j] \leftarrow v(j)$
Fin pour
Pour tout $(a,b;c,d) \in Y$ **Faire**
  $M[a,b,c,d] \leftarrow 0$
Fin pour
Pour tout $(i,f,j) \in I$ **Faire**
  Si f=0 **Alors** {c-à-d $(i,j^i) \in B(p^j)$}
    Pour tout $(a,b,c,d) \in F_{P^j}(i)$ **Faire**
      $V[j] \leftarrow V[j] + M[a,b,c,d]$
    Fin pour
  Sinon {c-à-d $f=1$ et i est la racine de Q^{p^j}, $i = r_j$}
    Pour tout $(a,b;c,d) \in Y_1 \cup Y_2$ tel que $P_j \subset (a,b;c,d)$ **Faire**
      Mettre à jour $M[a,b,c,d]$ avec $w = V[j] + M[a,l(Q_{p_j})-1,c,l(T_{[}P_j])-1]$
      Pour tout $P_g \subset (r_j+1,b;r_j^j+1,d)$ **Faire**
        Mettre à jour $M[a,l(Q_{P^g})-1,c,l(T_{P^g})-1]$ avec w
      Fin pour
    Fin pour
  Fin si
  $V[j] \leftarrow V[j] + M[0,l(Q_{p^j})-1,0,l(T_{P^j})-1]$
Fin pour
Retourner $\max_j V[j]$

Algorithme plus efficace

L'idée clé pour améliorer la complexité en temps est d'utiliser moins d'accès à M (tout en maintenant la propriété M_1 sur les entrées) et de compléter M avec une structure de donnée R qui peut être interrogée en $O(\log(m))$ en temps au lieu de $O(1)$, mais qui peut être maintenant sans qu'une boucle avec $O(m^2)$ itérations soit requise.

Formellement, soit $X = \{(a,c)$ c'est-à-dire $\exists (a,b;c,d) \in Y_1 \cup Y_2\}$ et R une structure de donnée indexée par X tel que, pour un index donné $(a,c) \in X$, $R[a,c]$ est un ensemble de paires (j,s) dans lequel j est l'index de la graine P^j et s est le score maximal des chaînes dans $(Q[a,r_j],T[c,r_j^j])$ qui termine avec P^j. Intuitivement,

- M est utilisé pour l'accès, toujours en $O(1)$ en temps, les valeurs $MCP(a,l(Q_{P^j})-1,c,l(T_{P^j})-1,\{P^0 \dots P^{j-1}\})$ requirent de calculer MCP' dans l'équation (4.1),
- $R[a,c]$ est utilisé pour accéder, en temps $O(\log(m))$, aux scores des meilleurs chaînes incluses dans $(Q[a,r_Q],T[c,r_T])$ (les valeurs $MCP(Q[e,f],T[g,h],\{P^0 \dots P^{j-1}\})$ dans l'équation (4.1)) et remplace les entrées $M[a,b,c,d]$ avec $(a,b;c,d) \in Y_1 \cup Y_2$, qui étaient utilisées dans l'algorithme précédent.

L'algorithme réalise une itération sur une liste de triplets $J = I \bigcup \left(\cup_{j=0}^{m-1} (l(Q_{P^j}), -1, j) \right)$, triés en utilisant un ordre lexicographique utilisé dans la précédente section, avec la modification suivante : si nous avons deux graines P^j et P^g avec $g > j$ tels que $(l(Q_{P^j}), l(T_{P^j})) = (l(Q_{P^g}), l(T_{P^g}))$ alors seulement $(l(Q_{P^j}), -1, j)$ apparait dans J. Le calcul J peut être réalisé en $O(||S|| \log(||S||))$ en temps.

Algorithme 10 $MCP_2(Q, T, S, v)$: Calcul une chaîne maximum dans S.

Entrées : Q, T : deux arborescences ordonnées, S : un ensemble de m graines et v : une fonction de score sur S.

Sorties : Le score de la chaîne maximal dans S

Pour j de 0 à $m-1$ **Faire**
  $V[j] \leftarrow v(j)$
Fin pour
Pour tout $(a, b; c, d) \in Y_3$ **Faire**
  $M[a, b, c, d] \leftarrow 0$
Fin pour
Pour tout $(a, c) \in X$ **Faire**
  $R[a, c] \leftarrow \emptyset$
Fin pour
Pour tout (i, f, j) dans J **Faire**
  Si $f = -1$ **Alors** $\{i = l(Q_{P^j})\}$
    Pour tout $(a, c) \in X$ tels que $a, c < l(Q_{P^j}), l(T_{P^j})$ **Faire**
      $M[a, l(Q_{P^j}) - 1, c, l(T_{P^j}) - 1] \leftarrow$ valeur s du dernier (y, s) de $R[a, c]$ s.t. $r_y^y < l(T_{P^j})$
    Fin pour
  Sinon Si $f = 0$ **Alors** $\{(i, i^j) \in B(P^j)\}$
    Pour tout $(a, b; c, d) \in F_j(i)$ **Faire**
      Ajouter à $V[j]$ la valeur s de la dernière entrée (y, s) de $R[a, c]$ telle que $r_y^y \leq d$
    Fin pour
  Sinon $\{f = 1$ et i est la racine de Q_{P^j}, $i = r_j\}$
    Pour tout $(a, c) \in X$ tels que $a, c \leq l(Q_{P^j}), l(T_{P^j})$ **Faire**
      $w \leftarrow V[j] + M[a, l(Q_{P^j}) - 1, c, l(T_{P^j}) - 1]$
      Insérer l'entrée (j, w) dans $R[a, c]$ et mettre à jour $R[a, c]$ comme suit :
      Trouver la dernière entrée (y, s) telle que $r_y^y < r_j^j$
      Si $s < w$ **Alors**
        Insérer (j, w) juste après (y, s) dans $R[a, c]$
        Supprimer de $R[a, c]$ toutes les entrées (z, t) telles que $r_j^j \leq r_z^z$ et $t < w$
      Fin si
    Fin pour
  Fin si
  $V[j] \leftarrow V[j] + M[0, l(Q_{P^j}) - 1, 0, l(T_{P^j}) - 1]$
Fin pour
Retourner $\max_j V[j]$

Validation de l'algorithme Nous considérons les invariants suivants.

M2. Après que la racine de P^j ait été traitée, alors $M[a,b,c,d] = MCP(Q[a,b],T[c,d],\{P^0,\ldots,P^j\})$ pour chaque $(a,b;c,d) \in Y_3$.

V1. Après que la racine de P^j ait été calculée, alors $V[j] = MCP'(Q,T,\{P^0,\ldots,P^j\})$.

R1. Après que la racine de P^j ait été calculée, alors pour tous $(a,c) \in X$, $R[a,c]$ contient tout les (y,s) satisfaisant :

a. $y \leq j$ et $s = MCP'(Q[a,r_y],T[c,r^y_y],\{P^0,\ldots,P^y\})$.

b. $\forall (z,t) \in R[a,c],\ r^z_z < r^y_y \Rightarrow t < s$.

R2. $\forall (a,c) \in X$, $R[a,c]$ est totalement ordonné tel que : $(y,s) < (z,t)$ si et seulement si $r^y_y < r^z_z$.

Nous considérons premièrement que $R1$ et $R2$ sont satisfaits. Comme précédemment, si $V1$ est satisfait alors l'algorithme calcul le $MCP(Q,T,S)$. La ligne d'initialisation 1 assure que $V[j]$ contient $v(j)$. Après, pour prouver V1, nous avons besoin de montrer que, quand nous calculons le bord i de la graine P^j (ligne 17), nous ajoutons $V[j]$ la meilleure chaîne de chaque zone chaînable $(a,b;c,d)$ du bord ; cela continue avec trois faits :

1. chaque graine P^{j+e} avec $e > 0$ n'appartient pas à la forêt $Q[a,b]$ (car $b < i \leq r_{j+e}$) et ainsi n'appartient pas à la chaîne dans la zone $(a,b;c,d)$,
2. le score de cette chaîne est présent dans $R[a,c]$ (depuis R1) et,
3. c'est la dernière entrée (y,s) telle que $r^y_y \leq d$ (depuis R2).

M2 peut-être prouvé d'une façon similaire à M1, mais restreint les entrées $M[a,b,c,d]$ telles que $(a,b;c,d) \in Y_3$. Pour vérifier que M2 est satisfait, nous avons juste besoin de regarder la ligne 13, qui est la seule ligne mettant à jour M. Pour les entrées $M[a,b,c,d]$ telles que $a \geq l(Q_{P^j})$ ou $c \geq l(T_{P^j})$, $M[a,b,c,d] = 0$ compte tenu de l'initialisation à la ligne 1. Pour toutes les entrées, M2 suivies immédiatement de R1 et R2, en utilisant des arguments similaires aux précédents.

Finalement, nous avons besoin de vérifier que R1 et R2 sont satisfaits. Premièrement, comme précédemment, dans le cas où $a \geq l(Q_{P^j})$ ou $c \geq l(T_{P^j})$, $R[a,c] = \emptyset$ qui est garanti par l'initialisation de la ligne 7. Donc nous avons seulement à considérer le cas dans lequel $a,c < l(Q_{P^j}), l(T_{P^j})$, qui est assuré dans les lignes 19 et 26. Chaque graine P^y tel que $y < j$ a déjà été calculé et $s = MCP'(Q[a,r_y],T[c,r^y_y],\{P^0,\ldots,P^y\})$ ne peut pas être modifié après que P^y ait été calculé, donc les lignes 20 et 21, ensemble avec M2, assurent que (y,s) ont été insérées dans $R[a,c]$ précédemment, et le même argument est appliqué si $y = j$. Les entrées (z,t) enlevées à la ligne 26 n'appartiennent à aucun des (y,s), ce qui implique R1.a et R1.b, et alors R1, sont satisfaits. R2 est manifestement satisfait de la position où (j,w) est inséré dans $R[a,c]$ à la ligne 25.

4.2.5 Complexité

Analyse de la complexité : preuve du théorème 1

La complexité en espace est donnée par l'espace requis pour les structures M et R. M nécessite un espace en $O(m^2)$ comme il est indexé par Y_3. R requiert un espace en $O(m\|S\|)$, comme $|Y_1 \cup Y_2| \in O(\|S\|)$ et pour chaque graine P^j, une entrée (j,s) est insérée au plus une fois pour chaque $R[a,c]$. Tout compris, la complexité en espace est en $O(m^2 + m\|S\|) = O(m\|S\|)$.

Nous allons maintenant décrire la complexité en temps. Premièrement, nous pouvons noter que la technique suivante utilisée pour le calcul de la chaîne maximum dans les séquences [Gus97, JMT92, OA05], les structures $R[l_Q,l_T]$ peuvent être implémentées en utilisant des structures de données classiques telles que des AVL ou des files que l'on peut concaténer qui supportent les

requêtes de données, insertions et délétions, successeur et prédécesseur, dans un ensemble de n éléments totalement ordonnés en $O(\log(n))$ dans le pire cas en temps (également appelé Range Minimum Query).

Maintenant analysons la complexité des lignes 10 à 13. La boucle de la ligne 12 est réalisée en au plus $O(m\|S\|)$ fois pour chaque itération requise $O(\log(m))$ en temps (ligne 13), ce qui amorti la complexité en temps donnée en $O(m\|S\|\log(m))$.

La ligne 17 est appliquée au plus une fois pour chaque $O(\|S\|)$ zone chaînable $F_j(i)$ (propriété 2), et chaque itération requiert $O(\log(m))$ en temps, qui donné une complexité amortie en temps en $O(\|S\|\log(m))$.

Finalement, nous analysons la complexité des lignes 19 à 30. Premièrement nous ne considérons pas l'opération de la ligne 26. La boucle commençant à la ligne 20 est réalisée en $O(m)$ en temps, et la complexité de chaque boucle est en $O(\|S\|)$ en temps. Le coût des opérations réalisées durant chaque itération est $O(\log(\|S\|))$ (lignes 21 et 24 sont tous les deux réalisés en $O(1)$ et lignes 22 et 23 en temps $O(\log(\|S\|))$). La complexité totale en temps de cette partie, sans considérer la ligne 26, est alors en $O(m\|S\|\log(\|S\|))$. Afin de compléter cette analyse de complexité en temps, nous montrons que la complexité amortie de la ligne 26 est en $O(m\|S\|)$. En fait, d'après R2, toutes les entrées supprimées dans un pas sont consécutives en suivant un ordre total sur $R[a,c]$ défini dans R2. Alors, si un appel ligne 26 supprime k éléments de $R[a,c]$, cela peut-être réalisé en $O((k+2)\log(m))$ en temps, tel que le successeur d'un élément donné peut être trouvé en $O(\log(m))$ en temps. Comme tous les éléments de R sont supprimés en au plus une fois pendant tout l'algorithme, cela implique une complexité amortie en $O(m\|S\|\log(m))$ pour la ligne 26. Pour conclure, notre algorithme calcule le $MCP(Q,T,S)$ en un temps en $O(m\|S\|\log(m))$, en utilisant cette structure de données et après un pré-traitement en $O(\|S\|\log(\|S\|))$ en temps pour calculer les zones chaînables et pour trier J. Cela prouve le Théorème 1.

Remarque 17 Si on considère que Q et T sont des séquences, ou, comme défini dans la remarque 16, arborescences unaires, alors chacune des deux arborescences ont une seule feuille et chaque graine est non ambigüe définie par sa racine et son bord, ce qui implique que $\|S\| = m$. Il y a seulement un $R[a,c]$, tel que $a = c = 0$, qui contient $O(m)$ entrées. Alors, toutes les boucles qui sont calculées sur R ont maintenant une seule itération, ce qui réduit la complexité en temps d'un facteur m à $O(\|S\|\log(m)) = O(m\log(m))$.

Analyse détaillée de la complexité

Afin d'établir la complexité en pire cas de l'Algorithme 10, nous avons à étudier le coût de l'algorithme pour chaque valeur de f. Afin de simplifier la lecture, nous notons n_1 la taille de Q et n_2 la taille de T. Sans perte de généralité, nous supposons de plus que $n_2 \leq n_1$.

Suivant les invariants $R1$ et $R2$, chaque liste de R contient au plus $min(m,n_2)$ éléments, comme il n'y a pas $(y,s),(y',s') \in R[a,c]$ tel que $r_y^y = r_{y'}^{y'}$, et $|X| \leq min(\|S\|,n_1n_2)$. Alors, dans le pire cas, nous avons au plus $O(n_1^2n_2^2)$ différentes zones chaînables, $|R| = O(n_1n_2)$, Pour tout (a,c) : $|R[a,c]| = O(n_2)$ et $|X| = O(n_1n_2)$.

$f = -1$ ligne 10 : Pendant l'exécution de l'algorithme chaque $M[a,l(r_j)-1,c,l(r_j^j)-1]$ est calculé seulement une fois pour chaque quadruplet possible tel qu'il n'y ait pas de (i,f,j), $(i',f',j') \in J$ tel que $(l(r_j),l(r_j^j)) = (l(r_{j'}),l(r_{j'}^{j'}))$. Chaque calcul nécessite une recherche dans $R[a,c]$ qui peut être faite en $O(log(n_2))$. Aors, le temps total de la complexité pour ce cas est en $O(n_1^2n_2^2log(n_2))$.

$f=0$ ligne 15 : Le calcul ligne 17 peut être stocké dans un tableau dédié M' tel que la meilleure chaîne de la zone (a,b,c,d) est calculé seulement une fois. Ainsi, durant toute l'exécution de l'algorithme, chaque zone chaînable différente requiert une recherche dans $R[a,c]$ et la complexité en temps totale de l'algorithme pour ce cas est en $O(\|S\|+n_1^2n_2^2\log(n_2))$.

$f=1$ ligne 19 : Ce cas est traité une fois par graine, donc $O(m)$ fois. Chaque traitement coûte $O(n_1n_2\log(n_2))$ et la complexité en temps est en $O(mn_1n_2\log(n_2))$.

À partir de cela, nous pouvons conclure que la complexité en temps du pire cas de notre algorithme est en

$$\begin{aligned}&O(\|S\|\log(\|S\|)+n_1^2n_2^2log(n_2)+\|S\|+n_1^2n_2^2\log(n_2)+mn_1n_2\log(n_2))\\&=O(\|S\|\log(\|S\|)+n_1n_2\log(n_2)(n_1n_2+m)+\|S\|)\\&=O(\|S\|\log(\|S\|)+n_1n_2\log(n_2)(n_1n_2+m))\end{aligned}$$

ce qui représente une amélioration de la complexité en pire cas de l'algorithme de Heyne *et al.* [HWBB09].

Pour conclure, nous pouvons fusionner l'analyse de la complexité en pire cas de la Section 4.2.4 avec la complexité en temps de l'Algorithme 10 :

$$\begin{aligned}O(\quad&\|S\| && \text{calcule des zones chaînables}\\&+\|S\|\log(\|S\|) && \text{tri des zones}\\&+\min(m,n_1n_2)\times\min(\|S\|,n_1n_2)\times log(\min(m,n_2)) && \text{cas } f=-1\\&+\|S\|+\min(\|S\|,n_1^2n_2^2)\times\log(\min(m,n_2)) && \text{cas } f=0\\&+m\times\min(\|S\|,n_1n_2)\log(\min(m,n_2)) && \text{cas } f=1\end{aligned}$$

tel que $|X|\leq\min(\|S\|,n_1n_2)$ et $|R[a,c]|\leq\min(m,n_2)$ pour tout a,c.

Comparaison avec l'algorithme de Heyne *et al.* [HWBB09]

Heyne *et* al. [HWBB09] ont aussi introduit un problème de chaînage sur une représentation alternative des arborescences ordonnées. Dans cette section, nous allons premièrement décrire cette méthode avec une analyse détaillée de sa complexité, suivie par une comparaison de notre algorithme.

Algorithme de programmation dynamique de Heyne *et al.* [HWBB09] L'algorithme de programmation dynamique décrit dans [HWBB09] est basé sur une traversée récursive des zones chaînables dans Q et T. Soient deux nœuds a et c, une matrice de programmation dynamique $D^{a,c}[b,d]$ contenant le score de la meilleure chaîne dans la zone.

De plus, un tableau W est utilisé pour stocker dans $W[j]$ les valeurs :

$$MCP'(Q[l(Q_{P^j}),r_j],T[l(T_{P^j}),r_j^j],\{P^0\dots P^j\}).$$

La récurrence de programmation dynamique est définie comme suit :

$$D^{a,c}[b,d]=\max\begin{cases}0 & \text{si } b<a \text{ ou } d<b\\ D^{a,c}[b-1,d]\\ D^{a,c}[b,d-1]\\ \forall P^j\in S \text{ tel que } r_j=b,\ r_j^j=d,\ l(Q_{P^j})\geq a \text{ et } l(T_{P^j})\geq c:\\ \quad D^{a,c}[l(Q_{P^j})-1,l(T_{p^j})-1]+W[j]\end{cases}\tag{4.4}$$

dans laquelle

$$W[j] = v(P^j) + \sum_{(a',b';c',d') \in CA(P^j)} D^{a',c'}[b',d'].$$

Afin de prendre en compte la dépendance entre le calcul des matrices $D^{a,c}$ et le tableau W, les graines sont traitées l'ordre postfixe de leurs racines (dans Q). Finalement le score du MCP est stocké dans $D^{0,0}[r_Q, r_T]$ [HWBB09].

L'algorithme peut être implémenté avec une complexité en espace de l'ordre de $O(n_1 \times n_2)$ provenant de la mémoire utilisée par la matrice $D^(a,c)$ après le calcul de toutes les zones chaînables. Les détails sont donnés dans [HWBB09], nous allons maintenant considérer la complexité en temps dans le pire cas. Lors d'une étape préliminaire, l'algorithme requiert un tri de toutes les graines et de toutes les zones chaînables, qui peut-être réalisé en $O(\|S\| \log \|S\|)$ pour le temps dans le pire cas qui correspond au nombre de zones chaînables qui est au plus linéaire en la somme des tailles de bords, *i.e.* $O(\|S\|)$. Ensuite chaque zone $(a,b;c,d)$ (*i.e.* le calcul de l'entrée $D^{a,c}[b,d]$) requiert un temps de calcul en $O((b-a)(d-c)+m)$ (en supposant que les valeurs nécessaires de W ont déjà été calculées. Cette complexité en temps est en fait bornée par $O(n_1 \times n_2 + m)$. Mettre à jour $W[j]$ peut-être fait par pallier quand les entrées $D^{a',c'}[b',d']$ pour $(a',b';c',d') \in CA(P^j)$ sont calculées, sans coût asymptotique supplémentaire. Comme une matrice de programmation dynamique est calculée pour chaque zone chaînable et qu'il y a au plus $n_1^2 n_2^2$ zones comme cela, la complexité en temps est alors à la fin de :

$$O(\|S\| \log \|S\| + \min(\|S\|, n_1^2 n_2^2)(n_1 n_2 + m)). \tag{4.5}$$

Avec le modèle de graine considéré par [HWBB09], le nombre de zones chaînables (*i.e.* $\|S\|$), et alors le nombre de graines peut-être borné par $O(n_1 n_2)$ qui résulte du pire cas en complexité en temps de $O(n_1^2 n_2^2)$ et en espace en $O(n_1 n_2)$ décrit dans [HWBB09].

Chaînage des graines dans les séquences Le problème de chaînage d'un ensemble de m graines sur deux séquences Q et T de longueurs respectives n_1 et n_2 peut-être résolu en $O(m \log m)$ [Gus97].

Cette complexité est en général considérée comme la meilleure connue. L'algorithme consiste dans un tri des extrémités des graines qui sont ensuite considérées par palier. Pour toutes les extrémités, un arbre binaire de recherche ou ABR (Déf. 1.42) est utilisé pour trouver la meilleure chaîne compatible ou pour insérer la nouvelle chaîne calculée [OA05]. La taille de l'ABR étant bornée par n_1, la complexité en pire cas est en $O(m \log min(m, n_1))$ en temps et en $O(m + min(m, n_1))$ en espace.

Pour les séquences, l'algorithme de Heyne *et al.* [HWBB09] nécessite une matrice de programmation dynamique C telle que $C[a,b]$ contient la meilleure chaîne incluse dans $Q[0 \ldots a]$ et $T[0 \ldots b]$. De plus, il ne requiert pas que les chaînes soient triées et qu'elle calcule la meilleure chaîne en $O(n_1 n_2 + m)$ en temps et en $O(n_1 n_2)$ en espace (ou, encore mieux en temps linéaire s'il n'y a pas de retour sur les pas pour extraire la meilleure chaîne optimale). Une des principales raisons de la différence de complexité est que l'algorithme de Heyne [HWBB09] factorise quelques calculs répétés dans l'algorithme de chaînage classique des graines.

Donc, si le nombre de graines est plus élevé que $\frac{n_1 n_2}{\log n_1}$, l'algorithme de Heyne [HWBB09] a une meilleure complexité asymptotique que l'algorithme normalement considéré qui relie une recherche dans une requête dans un ABR, car le coût pour la recherche dans un ABR dépasse le coût de scanne de tout l'espace de recherche. Aussi loin que nous le savons, cette propriété

d'algorithme de programmation dynamique pour le chaînage de graines sur les séquences n'a jamais été considérée.

Dans la pratique, le chaînage de graines est souvent utilisé pour éviter une comparaison deux à deux, couteuse entre Q et T. En génomique, par exemple, FASTA [LP85] utilise des graines pour obtenir une complexité quadratique en temps entre deux séquences. La présente comparaison montre les limites sur le nombre de graines, il faut donc limiter le nombre de graines avec un filtre efficace. En pratique, si les graines sont bien sélectionnées (et ont en particulier une bonne spécificité), nous pouvons supposer que m est en $O(min(n_1,n_2))$ dans ce cas l'algorithme classique [OA05, Gus97] est meilleur, car il ne dépend pas de la taille de Q et de T.

Chaînage de graines dans des arborescences ordonnées L'analyse que nous venons juste de réaliser pour le chaînage de graines intervient avec une petite modification de notre problème. En fait, l'Algorithme 10 est une extension du classique algorithme pour le chaînage de graines dans les séquences et son plus mauvais cas en complexité en temps ne dépend pas de la taille de Q et T mais il nécessite un ABR pour éviter de traverser exhaustivement les zones chaînables. En fait, l'algorithme de Heyne peut avoir une meilleure complexité en pire cas en complexité en temps que l'Algorithme 10 dépendant de la structure des graines considérées.

Par exemple, il est évident à partir de notre analyse de complexité que si $\|S\| \leq n_1^2 n_2^2$ et $m\log(m) \leq n_1 n_2$, alors notre algorithme a une meilleure complexité asymptotique dans le plus mauvais cas de complexité. En fait, l'algorithme de Heyne *et al.* a une complexité en temps en $O(\|S\|(\log(\|S\|) + n_1 n_2 + m))$, alors que notre algorithme a une complexité en temps en $O(\|S\|(\log(\|S\|) + m\log(m))$. Si les graines ont une taille constante (c'est-à-dire que pour toutes les graines P, $|B(P)| < k$ pour un k borné) – ce qui est souvent le cas en biologie computationnelle dans laquelle k-mers sont très utilisés pour modéliser des graines – nous avons $\|S\| = O(m)$ et dans ce cas, notre algorithme est plus efficace si $m \leq \frac{n_1 n_2}{\log n_1}$, encore une supposition réaliste.

4.2.6 Chaînage 1D

Dans cette section, nous présentons un sous-problème, un algorithme résolvant le problème du chaînage maximum dans une seule arborescence. Ce problème peut être défini intuitivement depuis le MCP avec $Q = T$ et dans lequel $Q_P = T_P$ pour chaque graine P. Nous appellerons ce problème 1D-MCP.

Définition 4.6 1D-MCP :
Considérons la racine d'une arborescence ordonnée T, un ensemble S de forêts internes de T appelées graines et une fonction de score v sur les graines. Une chaîne valide est un sous-ensemble de S de graines non chevauchantes. Soit $C_S(T)$ l'ensemble de toutes les chaînes valides,

$$1D - MCP(T,S) = max\{v(C); C \in C_S(T)\}$$

où $v(C) = \sum_{P \in C} v(P)$.

D'après la définition du 1D-MCP l'algorithme Section 4.2.4 résout le 1D-MCP.

En fait, l'Algorithme 10 réalise un calcul imposé par le fait que les graines sont une mise en correspondance entre deux arborescences. La structure R maintient un ensemble de listes qui peuvent être simplifiées par un ensemble de valeur dans le cas 1D.

Dans cette section, nous appelons zones chaînables d'une graine P, tous les couples (a,b) avec $l(T_P) \leq a,b < r_{T_P}$ qui définit une S-forêt interne valide et sont libres de nœuds de P. (a,b) est maximal si ni $(a-1,b)$, ni $(a,b+1)$ ne sont des zones chaînables valides dans P. Pour tout $b \in B(T_P)$, nous notons $F(b)$ l'ensemble de toutes les zones chaînables maximales (a,b) pour P tel que $l(b) \leq a,b < b$ et pour tout $i \in [a,b]$, il n'y a pas de nœud de P sur le chemin de i à b.

Nous redéfinissons les ensembles Y, Y_1, Y_2 et Y_3 pour le cas 1D comme suit :

$$Y = Y_1 \cup Y_2 \cup Y_3$$

$$Y_1 = \bigcup_{j=0}^{m-1} CA(P^j),\ Y_2 = \{(0, r_T)\},$$

$$Y_3 = \{(a, l(r_{Q_{P^j}}) - 1) \mid \exists b; (a,b) \in Y_1 \cup Y_2 \text{ et } P^j \subset (a,b)\}$$

$$X = \{a \text{ tel que } \exists (a,b) \in Y_1 \cup Y_2\}$$

L'Algorithme 11 résout le 1D-MCP similairement à l'Algorithme 10. Pour simplifier, approximativement, l'algorithme n'a pas à régler les problèmes de croisement entre les deux graines et une mise en correspondance valide. La preuve de la validité suit la preuve de la validité de l'Algorithme 10.

Complexité La complexité en espace dépend de la taille des structures M et R. À partir de la propriété 2, $|Y_1|$ est en $O(min(\|S\|, n^2))$. $|Y_3|$ est en $O(min(m^2, n^2))$ et $|X|$ est en $O(min(n, \|S\|)$. Ainsi la structure M prend $O(min(m^2, n^2))$ en mémoire et la structure R utilise $O(min(n, \|S\|))$ en espace. La complexité totale en espace est $O(min(m^2, n^2) + min(n, \|S\|))$.

En supposant que les zones chaînables sont déjà calculées pour toutes les graines, la complexité en temps de l'algorithme est décomposée comme suit :

- $O(\|S\| \log \|S\|)$: temps requit pour trier J
- $O(min(m^2, n^2))$, ligne 4
- $O(min(n, \|S\|))$, ligne 7
- cas $f = -1$: ligne 11-13. Ces lignes sont réalisées en $O(min(m,n))$ en temps. Chaque itération se fait en $O(min(m,n)$ en temps alors le coût total de ce cas est en $O(min(m,n)^2)$.
- cas $f = 0$: ligne 15-17. Sur toute l'exécution, chaque zone chaînable est considérée seulement une fois. Il y a au plus $O(\|S\|)$ zones dans ce cas, alors le coût des lignes est en $O(\|S\|)$.
- cas $f = 1$: ligne 19-24. Il y a m racine et $|X|$ est en $O(min(n, \|S\|)$, alors ces lignes sont en $O(m \times min(n, \|S\|))$.

La complexité totale est alors en $O(\|S\| \log \|S\| + min(m^2, n^2) + m \times min(n, \|S\|))$.

Dans le cas des séquences, $X = 0$ et Y_3 est vide. Alors, pour chaque graine P^j, il n'y a pas de chaîne à gauche de P^j (dans $T[0, l(T_{P^j})[= \emptyset$) mais seulement sous l'unique nœud du bord : la structure M ne sera jamais utilisée (Y_3 est vide), la boucle de la ligne 12 n'est jamais exécutée (comme $l(T_{P^j}) = 0$), la ligne 15 est exécutée seulement une fois par graine et chaque bord comme une simple zone chaînable et finalement la boucle ligne 20 est exécutée une fois par racine. Toutes ses observations montrent que, dans le cas des séquences, notre algorithme est identique à l'algorithme décrit dans [Gus97] et que sa complexité est limitée par le temps requis pour trier J : $O(m \log m)$.

Algorithme 11 $1D-MCP(T,S,v)$: Calcul le chainage maximum sur S.

Entrées : T : une arborescence ordonnée, S : un ensemble de m graines et v : une fonction de score sur S.

Sorties : Le score de la chaîne maximal dans S

Pour j de 0 à $m-1$ **Faire**
  $V[j] \leftarrow v(j)$
Fin pour
Pour tout $a \in X$ **Faire**
  $M[a,b] \leftarrow 0$
Fin pour
Pour tout $a \in X$ **Faire**
  $R[a] \leftarrow 0$
Fin pour
Pour tout (i,f,j) in J **Faire**
  Si $f=-1$ **Alors** $\{i=l(T_{P^j})\}$
    Pour tout $a \in X$ tel que $a < l(T_{P^j})$ **Faire**
      $M[a,l(r_j)-1] \leftarrow R[a]$
    Fin pour
  Sinon Si $f=0$ **Alors** $\{(i) \in B(P^j)\}$
    Pour tout $(a,b) \in F_j(i)$ **Faire**
      $V[j] \leftarrow V[j]+R[a]$
    Fin pour
  Sinon $\{f=1$ et i est la racine de T_{P^j}, $i=r_j\}$
    Pour tout $a \in X$ tel que $a \leq l(T_{P^j})$ **Faire**
      $w \leftarrow V[j]+M[a,l(T_{P^j})-1]$
      Mettre à jour $R[a]$ avec w
    Fin pour
    $V[j] \leftarrow V[j]+M[0,l(T_{P^j})-1]$
  Fin si
Fin pour
Retourner $\max_j V[j]$

Comme présenté dans [LPR+08, HWBB09] cette méthode peut être utilisée pour améliorer la comparaison d'une requête contre une base de données de structures secondaires d'ARN. Mais auparavant il est nécessaire de définir la notion de graines sur les arborescences.

4.3 Définition de graines sur des arborescences ordonnées

Notre travail, bien que relativement général est motivé par des questions biologiques et particulièrement appliqué aux structures secondaires d'ARN. Le traitement de larges bases de données de structures secondaires d'ARN, telles que Rfam [GJBM+03, GJMM+05, DGT+08, GDT+08, GDT+10], est un problème important en bioinformatique. Une première approche

adaptant la notion de distance d'édition sur des arborescences ordonnées a été apportée par Zhang et Shasha [ZS89]. L'approche d'édition d'arborescence a ensuite été étendue de différentes manières apportant :

- des problèmes difficiles lorsque sont considérées des opérations d'éditions plus complètes, telles que présentées dans le chapitre précédent [JLMZ02],
- des algorithmes avec une complexité en pire cas au mieux cubique, même avec un ensemble minimum d'opérations d'édition [DMRW09, ZS89].

Pour la structure secondaire d'ARN, le calcul de la distance d'édition est plus coûteux que celui entre séquences pour lesquelles des méthodes heuristiques, basées sur le filtrage ont vu le jour. Le présent travail est une partie d'un plus large travail qui doit permettre d'étendre cette approche aux structures secondaires d'ARN. Cette notion a été abordée dans [HWBB09], à travers la notion de graines basées sur une mise en correspondance exacte maximale.

Dans la Section 4.2, nous décrivons un algorithme efficace pour le chaînage de graines dans des arborescences ordonnées représentant des structures secondaires d'ARN. Nous présentons dans cette section un modèle de graines pour les structures secondaires d'ARN, défini par des segments de séquences (structure primaire) comportant en plus des éléments de structures secondaires conservés. Ce modèle permet d'utiliser les informations structurelles afin d'identifier des homologues tout en gardant un modèle de graines qui reste proche du modèle classique de graines sur les séquences et pouvant être calculé avec des méthodes efficaces. Notre modèle de graines associé à une méthode de filtrage peut-être développé pour d'autres modèles d'arborescences ordonnées.

Dans la Section 4.3.1 nous présentons les graines (l,d) centrées. Leur lien avec la distance d'édition est donné Section 4.3.2. Enfin, Section 4.3.3 nous illustrons la couverture de ces graines sur des ARN.

4.3.1 Définition de graines pour des arborescences ordonnées

Définir des graines pour une arborescence ordonnée est plus compliqué que sur les séquences. Une première idée serait de considérer toutes les sous-structures connexes des arborescences de taille fixée q. Cela apporte de nombreux problèmes, notamment, le fait que le nombre de graines peut-être exponentiel en q et qu'il peut-être très coûteux d'énumérer de telles sous-structures. Un des buts principaux est de définir un modèle de graines d'arborescences qui assurent que le nombre maximum de graines dans une arborescence est linéaire en sa taille.

Définition de graines (l,d) centrées pour des arborescences ordonnées

Nous assurons cette propriété avec les *graines centrées* décrites ci-dessous. Bien que notre définition des graines (l,d) centrées soit basée sur les séquences arc-annotées, il existe une correspondance entre arborescences et séquences arc-annotées (*cf.* Chapitre 1).

Définition 4.7 (Graines (l,d) centrées) Soient T_1 et T_2 deux arborescences ordonnées et $A_1 = (S_1, P_1)$ et $A_2 = (S_2, P_2)$ les séquences arc-annotées correspondantes. Soient d et l deux entiers tels que $2d \leq l$. Une graine (l,d) centrée est un couple (i,j) de positions respectives dans A_1 et A_2 tels que :

- $0 \leq i \leq |A_1| - l - 1$ et $0 \leq j \leq |A_2| - l - 1$
- $P_1[i, i+l] = P_2[j, j+l]$
- $S_1[i+d, i+l-d] = S_2[j+d, j+l-d]$

En d'autres termes, une graine (l,d) centrée, (i,j) est définie par une mise en correspondance $M = \{(u,v)/x = 0 \ldots l, u = \alpha(i+x), v = \alpha(j+x)\}$. Manifestement M définie une paire de forêts dans T_1, T_2 telles que les racines des arborescences de chaque forêt sont dans la même fratrie et sont une mise en correspondance.

Nous associons un score à chaque graine telle que :

$$score(M) = l + \sum_{(u,v)\in M} f(u,v)$$

où $f(a,a) = 1$ et $f(a,b) = 0$ si $a \neq b$.

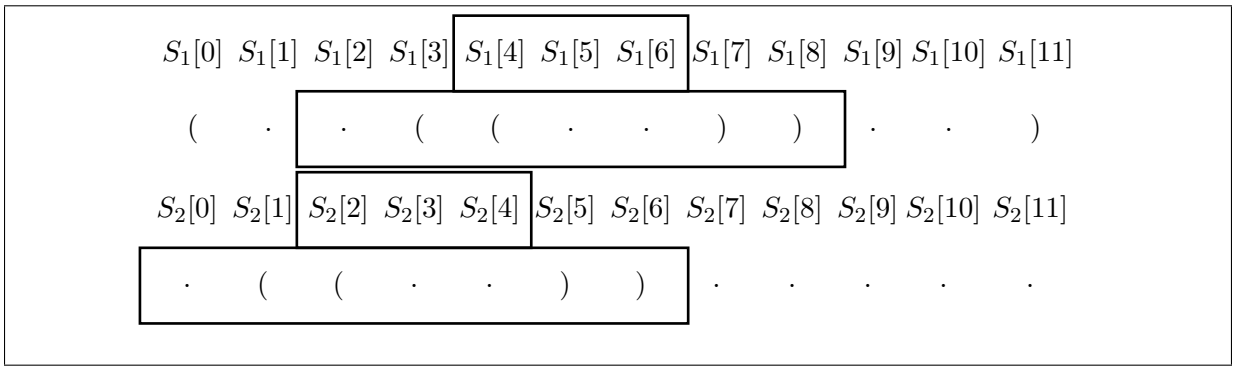


Figure 32 : Représentation de graines sur des séquences arc-annotées.

Ainsi une telle définition permet de mieux « conserver » la structure secondaire que la structure primaire, comme nous pouvons le constater Fig. 32. Lorsque la graine contient une base appariée et pas sa paire, nous considérons que le nœud interne représentant la paire est inclus dans la graine.

4.3.2 Propriétés des graines (l,d) centrées

Nous pouvons définir la similarité de graines (l,d) centrées entre deux arborescences.

Définition 4.8 (Similarité des graines (l,d) centrées) Soient T_1 et T_2 deux arborescences ordonnées, nous appelons C l'ensemble de toutes les graines (l,d)-centrées entre T_1 et T_2. La *similarité des graines (l,d)-centrées* entre T_1 et T_2, notée $css(l,d,T_1,T_2)$ est définie par le score maximum de chaînage sur C [ACFG11]. C'est, le sous-ensemble C' de C tel que : le score $score(C') = \sum_{M\in C'} score(M)$ est maximal sur toutes les possibilités de C'. Appelons C' la *c*haîne de graine optimale. La *n*ormalisation de la similarité des graines (l,d)-centrées est :

$$css_N(l,d,T_1,T_2) = \frac{css(l,d,T_1,T_2)}{|T_1| + |T_2|}$$

Nous observons maintenant une relation entre la distance d'édition avec un modèle de coût unitaire et les scores de similarité des graines (l,d)-centrées. Cette propriété permet de nous assurer que le chaînage sur nos graines ne gardera pas des arborescences trop éloignées.

Propriété 4 Soient $edition_{UN}$ une distance d'édition avec des coût unitaires, c'est-à-dire qu'elle utilise un coût de 1 pour les insertions, les délétions et les substitutions (u,v) telles que $u \neq v$, et un coût de 0 pour une substitution si $u = v$ et T_1 et T_2 deux arborescences ordonnées alors

$$edition_{UN}(T_1,T_2) \leq 1 - css_N(l,d,T_1,T_2).$$

Preuve Considérons C', la *chaîne de graine optimale* qui correspond à $css_N(l,d,T_1,T_2)$. Nous avons

$$css(l,d,T_1,T_2) = \sum_{M \in C'} \sum_{(i,j) \in M} 1 + f(e_1(i),e_2(j)) = |C'|l + \sum_{M \in C'} \sum_{(i,j) \in M} f(e_1(i),e_2(j))$$

maintenant calculons le score d'édition dans un modèle de coût unitaire associé avec une mise en correspondance définie par C' :

$$cost_u(C') = \sum_{M \in C'} \sum_{(i,j) \in M} 1 - f(e_1(i),e_2(j)) + \sum_{i \in T_1 \notin C'} 1 + \sum_{j \in T_2 \notin C'} 1$$

La première partie correspond à une substitution, puis les délétions (coût 1) et les insertions.

$$cost_u(C') = |C'|l - \sum_{M \in C'} \sum_{(i,j) \in M} f(e_1(i),e_2(j)) + |T_1| - |C'|l + |T_2| - |C'|l$$

Il est donc évident que $cost_u(C') = |T_1| + |T_2| - css(l,d,T_1,T_2)$ et $edit_u(T_1,T_2) \leq |T_1| + |T_2| - css(l,d,T_1,T_2)$ est la plus petite distance des coûts des possibles mises en correspondance.□

4.3.3 Applications aux structures secondaires d'ARN

Sur la Fig. 33 sont testées les graines (0-10), (1-7), (2-6) et (3-5) centrées sur des ARN. En ordonnée est représentée la distance d'édition Zhang-Shasha entre arborescences ordonnées et en abscisse le pourcentage de couverture définit par $\frac{2*\text{score chainage}}{2*|T_1|+2*|T_2|}$. Cette courbe bien que préliminaire nous permet de remarquer que le pourcentage de couverture et la distance d'édition sont corrélés, en effet plus le pourcentage de couverture est élevé plus la distance d'édition est faible. Notons tout de même que pour une distance de 0 le pourcentage peut être relativement faible (45%). Une étude approfondie des performances des graines $(l-d)$ centrées est nécessaire afin de déterminer les plus pertinentes.

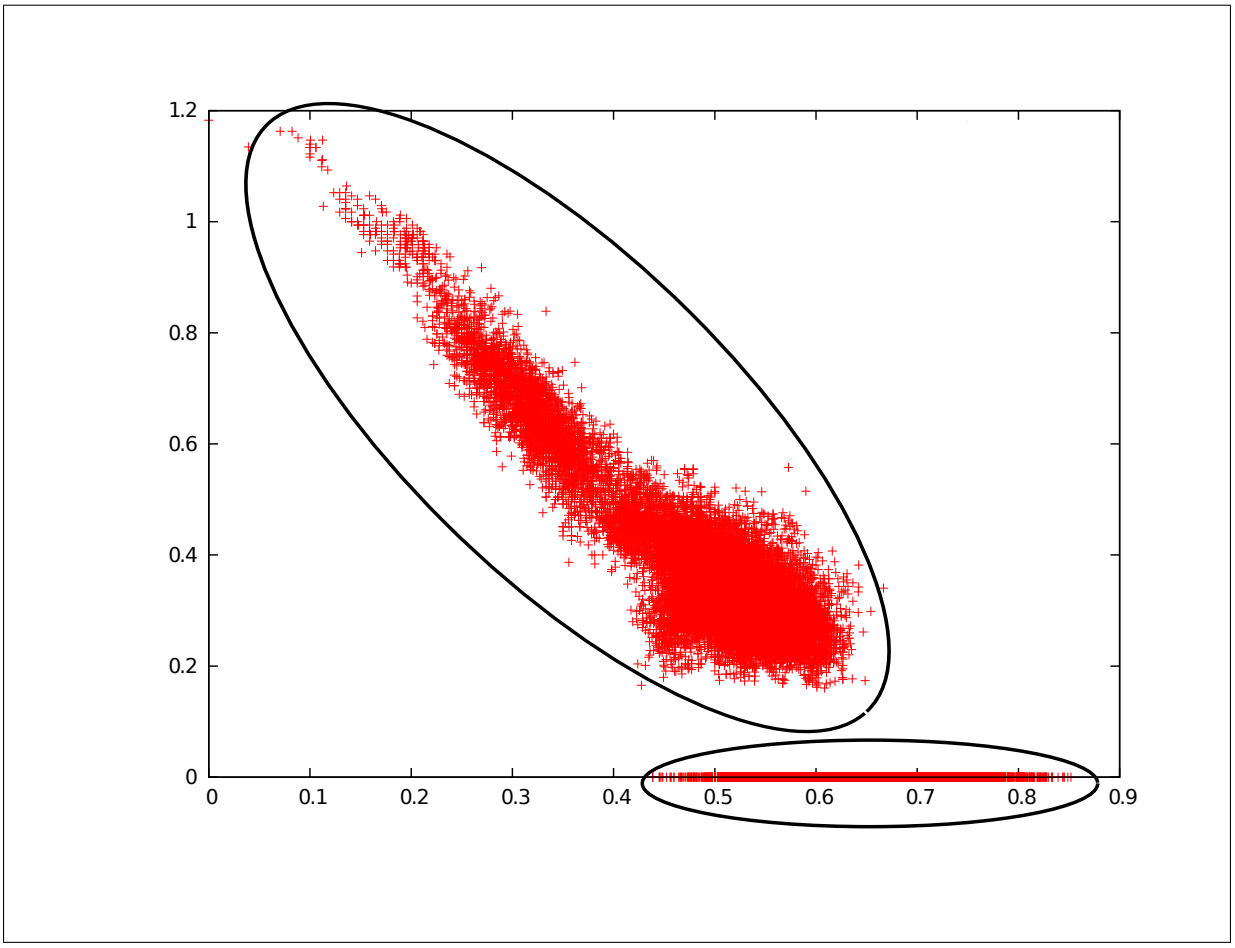


Figure 33 : Le pourcentage de couverture des graines en fonction de la distance unitaire de Zhang-Shasha avec chaînage sur les arborescences.

Nous avons proposé dans ce chapitre une nouvelle méthode de comparaison des structures secondaires d'ARN. La distance d'édition peut alors être calculée pour les structures proches de la requête. Il convient maintenant de tester ces méthodes sur des structures secondaires d'ARN afin de déterminer la taille des graines la plus adaptée. Une autre perspective serait de considérer des graines espacées qui pourraient avoir des meilleurs résultats comme dans le cas des séquences. D'un point de vue pratique, il faudrait également évaluer l'impact de la phase de prétraitement du chaînage.

Chapitre 5

Compression de structures arborescentes

Nous l'avons vu en introduction, l'augmentation de la quantité de données biologiques mais également de la taille de chaque donnée nécessite la mise en place d'outils appropriés [OGG+10]. Cela implique non seulement des traitements efficaces (Chapitre 4) mais également une représentation efficace prenant en compte la complexité des données biologiques disponibles. Il est important de noter qu'un codage optimal nécessite généralement un traitement complexe pouvant nécessiter le passage par un autre stockage temporaire. *A contrario*, un codage permettant un traitement efficace utilise souvent un espace mémoire non optimal. Il convient donc de trouver un compromis entre les phases de codage et de traitement.

Le problème de stockage est bien connu en infographie [Wil71], une façon de le traiter est d'utiliser les redondances au sein de l'objet et de les stocker une seule fois. Ce principe peut être appliqué pour la compression de données biologiques, en effet, des redondances sont présentes dans de nombreux objets biologiques, par exemple, dans les séquences nucléotidiques de l'ADN [GK00]. Elles sont, entre autres, utilisées pour diagnostiquer des maladies génétiques [SR95], pour évaluer un risque, tel que le cancer [TBS93] ou réaliser une empreinte génétique afin de déterminer une identité [But06]. La compression des séquences biologiques est d'ailleurs un problème largement étudié [RDDD96, CKL00].

Les séquences d'ADN ne sont pas les seuls objets biologiques comportant des redondances. Mandelbrot [Man82] a ainsi introduit le terme fractale pour désigner une forme géométrique pouvant être récursivement divisée en plusieurs parties dont chacune est une copie potentiellement approximative de taille réduite de l'ensemble, en s'appuyant sur des formes fractales observables dans la nature. Par exemple, sur la Fig. 34, nous pouvons observer cette répétition de formes sur (a) le chou romanesco et (b) le réseau d'alvéoles pulmonaires avec les bronches qui se divisent jusqu'aux alvéoles. Nous l'avons vu en introdution, un grand nombre d'objets biologiques peut être modélisés par des structures arborescentes, nous proposons ainsi des méthodes lesquelles déterminant les répétitions dans ces structures pour compresser la redondance.

Une particularité des données biologiques par rapport à des objets purement mathématiques est qu'elles peuvent contenir du *bruit*, issu de leurs potentielles natures évolutives [Rea84], cause de leurs environnements [FK50], ou encore une conséquence inévitable des techniques de collecte de données [SK95, KY07]. Avec une compression avec pertes adaptée, celui-ci pourrait tendre à disparaître ou à être lissé [Nat93, DBT01, LCLZ10]. Dans le cas où le bruit est une conséquence des techniques de récolte de données [SK95, KY07] une compression avec pertes pourrait alors corriger le problème. Notons tout de même que le bruit peut contenir une information biologique essentielle. L'analyse de l'expert reste donc incontournable.

(a) (b)

Figure 34 : Exemples d'objets biologiques de type fractal, sous licence Creative Commons. (a) La forme fractale du chou romanesco. (b) Le réseau d'alvéoles pulmonaires forme une surface fractale.

Une partie de ce travail a été réalisé à Calgary et à Montpellier en collaboration avec Christophe Godin (INRIA), Frédéric Boudon (CIRAD), Przemyslaw Prusinkiewicz (University of Calgary) et Pascal Ferraro (LaBRI) et présenté à International Workshop on Functional-Structural Plant Models (FSPM) 2010 à Davis [GFBG10] ou est en cours de publication.

Nous proposons dans cette partie une méthode de compression de structures biologiques arborescentes en les stockant sous forme de graphes acycliques orientés. Dans la Section 5.1, nous présentons des méthodes de compression de séquences et d'arborescences. Nous proposons, dans la Section 5.2 une méthode permettant de réaliser une compression avec pertes d'une partie de l'information contenue dans une arborescence non-ordonnée et une extension prenant en compte des informations supplémentaires contenues dans les étiquettes de nœuds. Nous évaluons cette méthode dans la Section 5.3 en l'appliquant à l'architecture des plantes.

5.1 Compression d'arborescences

L'objectif de la compression est de diminuer le nombre de bits nécessaires pour coder l'information. La compression est une opération qui consiste à passer de la représentation initiale d'un objet à une représentation plus compacte, c'est-à-dire nécessitant moins d'espace de stockage. D'un point de vue général, les méthodes de compression sont divisées en deux classes [Say00]. Elles se différencient sur la phase de décompression, l'opération qui consiste à passer de l'objet compressé à un objet non compressé. La compression sans perte est telle que l'objet de départ peut être reconstitué à partir de l'objet compressé de façon identique. À l'inverse, dans la compression avec perte, l'objet initial ne peut pas être reconstruit à l'identique (Fig. 35).

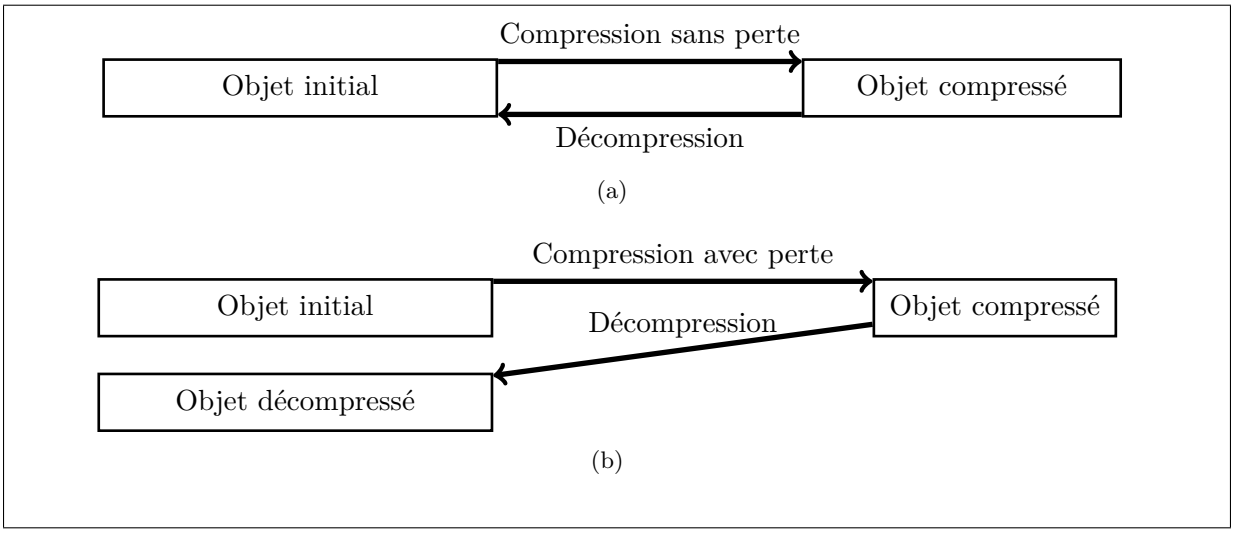


Figure 35 : Les deux types de compression. (a) Principe de la compression sans perte. (b) Principe de la compression avec perte.

La compression de données fait partie de la théorie de l'information qui regroupe, entre autres, le codage de l'information et la mesure de la redondance d'un objet que nous utilisons également Section 5.2. La compression de l'information est appliquée sur de nombreux objets notamment les documents, les fichiers textes, les images, le son ou la vidéo. L'un des exemples les plus connus en bioinformatique est celui de la compression des séquences d'ADN. Ces techniques permettent un stockage plus efficace de séquences d'ADN [MSI00] mais aident également à la compréhension de ces données [CLMT02].

Une brève description de la compression de séquences est présentée Section 5.1.1. Enfin, Section 5.1.2, nous verrons les méthodes de compression d'arborescences ainsi qu'une méthode de compression sans perte de la topologie d'une arborescence, dont les méthodes présentées Section 5.2 sont des extensions.

5.1.1 Compression de séquences

Compression sans perte

Une des premières méthodes de codage fut inventée en 1838 [Say00], pour une utilisation en télégraphie. Le code Morse utilise des codes plus courts pour les caractères « e » et « t » qui sont plus fréquents en anglais. Une représentation sous forme d'arborescence peut être obtenue telle que présentée Fig. 36. L'idée du code Morse est d'utiliser des symboles plus courts pour les caractères les plus répétés. Une technique alternative est de compresser les répétitions successives au sein d'une séquence. Par exemple, les techniques de code par plages (en anglais : « run-length encoding ») ont pour principe de rassembler un ensemble de plages de caractères identiques consécutifs par une paire (plage, nombre d'occurrences de la plage) [Sal07]. La compression peut donc utiliser des stockages utilisant moins d'espace pour un caractère ou pour un ensemble de caractères répétés.

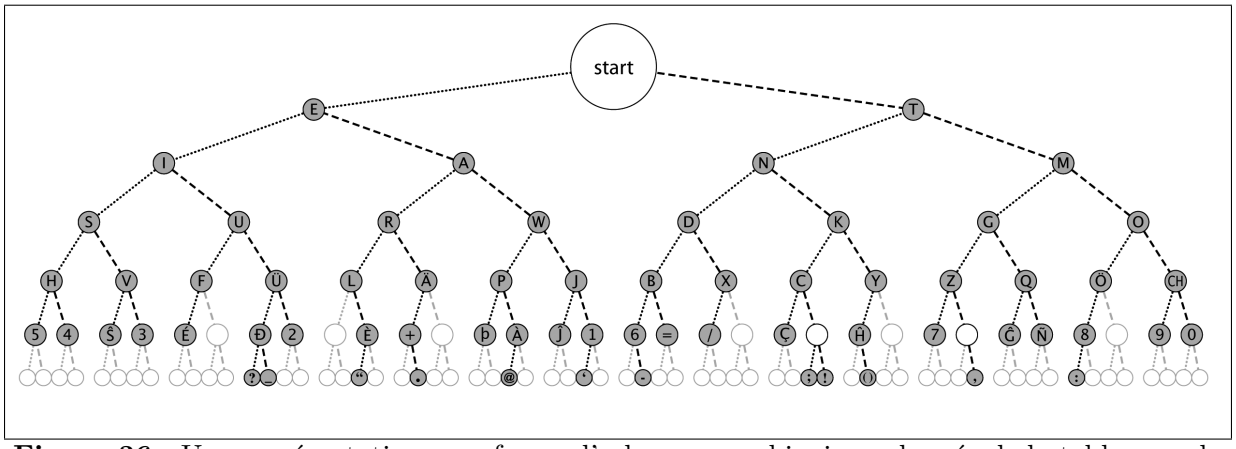


Figure 36 : Une représentation sous forme d'arborescence binaire ordonnée de la table pour la recherche dichotomique du code Morse (image libre de droit). Pour trouver un caractère à partir du code Morse, il faut partir de la racine (nœud « start »). Notons v_g son enfant gauche et v_d le droit, pour chaque symbole « . », la recherche continue dans la sous-arborescence enracinée en v_g, à l'inverse, le symbole « - »correspond à une recherche dans la sous-arborescence enracinée en v_d. Lorsque l'ensemble des symboles du code Morse ont été parcourus, l'étiquette du nœud courant est le symbole codé.

Méthodes statistiques Les méthodes de compression ont réellement commencé à être étudiées dans les années 1940 avec le développement de la théorie de l'information [Sal07]. Les premières méthodes développées sont des méthodes de compression statistique. D'un point de vue naïf, elles utilisent la répartition des caractères dans le texte, plus précisément leur fréquence d'apparition dans le but de compresser. Les codes petits sont alors utilisés pour les caractères les plus fréquents.

Fano et Shannon ont simultanément conçu un algorithme basé sur les statistiques [Fan39, Sha48, Sal07]. Le codage de Shannon-Fano est réalisé en plusieurs étapes. Premièrement, les symboles à compresser sont triés selon leurs probabilités d'apparition. Lorsque ces dernières ne sont pas connues, une phase de prétraitement à lieu afin de les calculer. L'ensemble des symboles est divisé en deux parties de façon à ce que les probabilités des deux parties soient le plus proche possible de l'égalité, la probabilité d'une partie étant égale à la somme des probabilités des différents symboles de cette partie. La première partie sera codée par des mots commençant par 0, la seconde par 1. Les parties obtenues sont subdivisées récursivement. Lorsqu'une partie ne contient qu'un seul élément, celui-ci est représenté par un code vide.

Le codage de Shannon-Fano a été un des précurseurs, cependant le codage de Huffman [Huf52] propose une solution statistique optimale dans le sens du nombre de symboles pour un codage d'un caractère avec probabilité connue [Sal07]. Chaque caractère est représenté par un nœud. Un poids est associé à chaque nœud correspondant à sa fréquence. Les deux nœuds avec les plus petits poids sont fusionnés afin de ne former qu'un seul nœud dont le poids est la somme des deux poids précédents. Cela est réitéré jusqu'à ce qu'il ne reste plus qu'un nœud. Le résultat peut être modélisé sous forme d'une arborescence binaire, issue du même principe que celui illustré Fig. 36 dans ce cas la branche gauche correspondra, par exemple, au caractère 0 et la branche droite au caractère 1.

Le codage arithmétique est assez similaire au codage de Huffman. Il permet de coder les symboles sur un nombre non-entier de bits, en effet il ne code pas caractère par caractère mais par chaînes de caractères. Il est meilleur du point de vue de l'entropie que le codage de Huffman sauf dans le cas où tous les poids sont des puissances de 2 [Sal07]. Cependant, compte tenu de la précision des ordinateurs, il est limité au codage de séquences relativement courtes.

Méthodes par dictionnaire Les méthodes par dictionnaires prennent en compte la redondance d'un ensemble de caractères de taille variable. En 1977, Lempel et Ziv [ZL77] ont suggéré l'idée de base de ce codage basé sur des pointeurs. Les mots répétés ou fréquents sont stockés dans un dictionnaire et remplacés par une adresse dans le dictionnaire. Le dictionnaire peut être calculé une seule fois ou évoluer dynamiquement.

Deux techniques de compression basées sur des dictionnaires appelés LZ78 et LZ77 ont été développées par Lempel et Ziv [ZL77, ZL78]. LZ77 [ZL77] n'a pas de dictionnaire proprement dit, cet algorithme utilise une technique de fenêtre glissante de longueur fixe. Les motifs lus précédemment dans la fenêtre servent de dictionnaire. Ainsi lorsqu'un motif a déjà été rencontré, il est remplacé par une référence vers la première occurrence. LZ78 [ZL78] utilise une approche complètement différente pour la construction du dictionnaire. Au lieu d'utiliser des motifs de longueur fixe à partir d'une fenêtre, il ajoute dans le dictionnaire le plus court motif non présent dans le dictionnaire. LZ78 nécessite de savoir trouver efficacement, dans un dictionnaire si un motif est présent.

Dans le milieu des années 1980, suite à des travaux de Terry Welch [Wel84], l'algorithme LZW, une variante de LZ78 a vu le jour. LZW est initialisé avec un dictionnaire contenant l'ensemble des caractères de l'alphabet. Au fur et à mesure de la lecture de la séquence à compresser, ce dictionnaire est complété par des sous-séquences de cette séquence. Cette méthode est utilisée pour la plupart des systèmes de compression à usage général.

Compression avec perte

Vers la fin des années 1980, les images numériques sont devenues plus fréquentes. Dans le début des années 1990, les méthodes de compression avec pertes ont également commencé à être utilisées. En ce qui concerne la compression de texte, ces techniques se heurtent généralement à la difficulté de garder la sémantique [WBM+94]. Sadeh [Sad96] a proposé un algorithme basé sur des motifs non exacts prouvant des propriétés asymptotiques.

Compression de séquences d'ADN

La compression de séquences biologiques n'est pas une tâche facile [CKL00]. Grumbach et Tahi [GT94] ont essayé de compresser des séquences d'ADN en utilisant les algorithmes de codage de Huffman [Huf52], les techniques arithmétiques et les algorithmes basés sur les méthodes de Lempel-Ziv dont LZ77 [ZL77], LZ78 [ZL78] et LZW [Wel84]. Les méthodes de compression présentées ci-dessus ne sont pas très efficaces, en général moins bon qu'un algorithme naïf [Lan04].

Compte tenu de la mauvaise performance des algorithmes traditionnels, deux méthodes de compression ont été proposées par Grumbach et Tahi, Biocompress [GT93] et Biocompress-2 [GT94]. Biocompress [GT93] est une combinaison d'une méthode proche de LZ77 [ZL77], dans laquelle chaque sous-séquence est encodée par une paire d'entiers représentant la longueur du

motif et sa position ainsi que d'un encodage dans lequel chaque caractère est codé par un nombre à deux bits. Biocompress2 [GT94] utilise en plus un codage arithmétique. Rivals *et al.* [RDDD96] ont proposé un autre algorithme de compression, Cfact [RDDD96] cherchant le plus long motif répété en utilisant un arbre des suffixes sur la séquence entière. Rivals *et al.* [RDD+97] ont également conçu un algorithme de compression permettant de détecter des répétitions en tandem de séquences d'ADN.

Notons que les séquences d'acides aminés des protéines peuvent également être compressées [MSI00].

5.1.2 Compression d'arborescences

Arborescences ordonnées

Le problème de la construction d'une compression d'une arborescence a été étudié dès les années 70 dans différents domaines. Le problème de compression des arborescences a ainsi été traité en éliminant les redondances internes apparaissant dans ces structures. Pour la réduction d'une arborescence ordonnée, divers algorithmes ont été proposés, ils possèdent une complexité allant de $O(n^2)$ à $O(n)$, en considérant n comme le nombre de nœuds dans l'arborescence, par exemple [DST80, CR96, BLM08].

Parmi ces méthodes, les méthodes de calcul d'une réduction sous forme de DAG sont très proches des algorithmes décrits pour tester si deux arborescences sont isomorphes. Les représentations d'arborescences sous forme de DAG sont très utilisées en informatique graphique dans lequel le processus de condensation d'une arborescence en graphe est appelé l'instanciation [Sut64]. Ce processus, tout d'abord utilisé par Sutherland [Sut63], rend possible le partage des nœuds représentant les mêmes objets dans une scène et ainsi évite les duplications non nécessaires des différentes instances du même objet graphique. Ce principe est maintenant usuellement implanté dans les scènes des applications d'infographie [SSdR03]. Il permet une manipulation efficace des très grandes scènes et est régulièrement appliqué dans le rendu complexe de scènes fractales ou de scènes modélisant des plantes.

Compression topologique des arborescences non-ordonnées sans perte

La méthode présentée dans cette section permet de compresser sans perte la topologie d'une arborescence non-ordonnée. Nous allons la définir plus précisément. La méthode de compression avec pertes présentée à la section suivante étant une extension.

Les arborescences auto-emboitées introduites par [Gre96] ne sont pas strictement autosimilaires, c'est-à-dire que les différentes parties de l'arborescences ne sont pas déduites par similitude par rapport au tout, mais possèdent une structure interne récursive. Ferraro et Godin [GF10] appellent de telles arborescences des arborescences auto-emboitées, pour insister sur leur structure récursive et sur leur proximité avec la notion d'autosimilarité. Les arborescences auto-emboitées sont telles que l'ensemble des sous-arborescences d'une certaine hauteur sont isomorphes (Déf. 1.20). Cette notion a été dérivée afin de donner la possibilité de compresser des arborescences non-ordonnées sans perte d'information en des DAG plus compacts.

Considérons la relation d'équivalence définie pour une arborescence isomorphe (c'est-à-dire deux sous-arborescences identiques au réétiquetage de leurs composants près) sur l'ensemble des sous-arborescences de l'arborescence T. Nous dirons que les nœuds v_x et v_y dans V sont équivalents si

et seulement si $T[v_x]$ et $T[v_y]$ sont isomorphes et on note par extension $v_x \equiv v_y$. Pour tout $v_x \in V$, posons $c(v_x)$ la classe d'équivalence de v_x.

Le graphe quotient $\mathcal{Q}(T) = (V_\mathcal{Q}, E_\mathcal{Q})$ ainsi obtenu depuis T utilisant la relation ci-dessus sur les nœuds de T est un DAG. $V_\mathcal{Q}$ est l'ensemble de classes équivalentes I sur V. $E_\mathcal{Q}$ est un ensemble de paires des classes d'équivalences telles que $(I,J) \in E_\mathcal{Q}$ si et seulement si $\exists (x,y) \in E$, $c(x) = I$ et $c(y) = J$.

Il est évident que $\mathcal{Q}(T)$ condense l'information topologique contenue dans l'arborescence T. Cependant, $\mathcal{Q}(T)$ ne contient pas nécessairement la même information que T.

Considérons un nœud x de T et notons $n(x,J)$ le nombre d'enfants de x qui sont de classe J :

$$n(x,J) = |\{z \in T | z \in enf(x) \text{ et } c(z) = J\}|$$

La quantité $n(x,J)$ est constante pour tout $x \in I$, et est ainsi indiquée par $n(I,J)$. Cette fonction définie sur les arcs de $\mathcal{Q}(T)$, est appelée la *distribution des signatures* de T [GF10].

Définition 5.1 (Réduction d'une arborescence) Soit T une arborescence et $\mathcal{Q}(T) = (V_\mathcal{Q}, E_\mathcal{Q})$ le graphe quotienté issu de T. La réduction $\mathcal{R}(T)$ de T est un graphe $(V_\mathcal{Q}, E_\mathcal{Q}^+)$, dans lequel $E_\mathcal{Q}^+$ est un multi-ensemble $\{((I,J), n(I,J))\}_{(I,J)\in E_\mathcal{Q}}$, n de la distribution des signatures T.

$\mathcal{R}(T)$ est ainsi un DAG $\mathcal{Q}(T)$ avec des étiquettes sur les arcs correspondants à $n(I,J)$.

Deux méthodes ont été proposées dans [GF10] pour calculer l'ensemble des sous-arborescences isomorphes.

Signature La première est basée sur une signature ajoutée à chaque nœud. Depuis les feuilles jusqu'à la racine, les signatures sont calculées comme étant l'union des signatures des enfants du nœud.

Distance d'édition La seconde méthode proposée part du calcul de la distance de l'arborescence avec elle-même, en utilisant la méthode de Zhang [Zha96] (décrite Section 3.3.1) ; les arborescences n'étant pas ordonnées. Les distances entre toutes les sous-arborescences sont alors connues et une distance nulle implique que les sous-arborescences sont isomorphes.

La complexité en temps du calcul de la réduction, en utilisant la méthode d'édition est en $O(|T|^2 deg(T) \log deg(T))$, en revanche celle basée sur les signatures est en $O(|T| deg(T) \log deg(T))$.

Un exemple de compression topologique sans perte est donné Fig. 37.

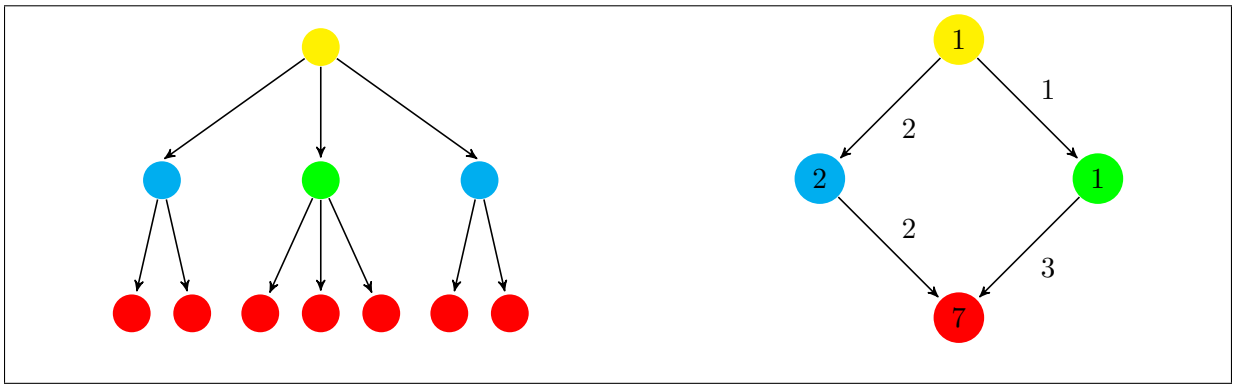


Figure 37 : Exemple de résultat de la compression topologique présentée dans [GF10].

Les méthodes de compression sont utilisées sur de nombreux objets. La compression sans perte définie sur des arborescences non-ordonnées, a été utilisée dans des cas pratiques pour définir une méthode de compression entre plantes. Hélas, bien que les plantes possèdent un caractère globalement redondant elles ne sont pas parfaitement répétitives, dues notamment aux erreurs introduites pour la mesure mais aussi aux aléas de croissance. Nous proposons aussi d'adapter cette méthode pour définir une méthode de compression avec perte. Pour améliorer le taux de compression et lisser les erreurs.

5.2 Compression des arborescences avec perte

Les structures biologiques arborescentes contiennent des répétitions, cependant ces structures peuvent posséder des répétitions *non exactes* et contenir des informations supplémentaires sur les étiquettes des nœuds. Nous proposons une méthode qui exploite l'autoredondance approximative d'une structure arborescente pour la compresser sur différents degrés tout en respectant un compromis entre le taux de compression et sa pertinence. Cette nouvelle méthode de compression permet d'augmenter l'efficacité des modèles, des simulations et des analyses de structures particulièrement redondantes en utilisant la représentation compressée à la place de la représentation initiale.

Dans ce manuscrit, nous utilisons d'abord les répétitions non exactes dans une arborescence non-ordonnée dans le but de calculer une compression avec pertes de ces données. Le résultat de la compression est un DAG qui possède un nombre maximum de nœuds pouvant être défini en entrée. Dans un second temps, nous proposons une solution pour compresser ces données lorsqu'elles contiennent des étiquettes sur les nœuds.

Dans la Section 5.2.1 nous présenterons le problème de compression topologique avec perte. Nous décrirons ensuite, Section 5.2.2, une méthode permettant de compresser avec pertes une structure arborescente étiquetée.

5.2.1 Compression avec pertes des arborescences non-ordonnées et non étiquetées

Partant d'une arborescence, le problème de compression avec pertes consiste à trouver une version compacte, c'est-à-dire nécessitant moins d'espace de stockage de l'arborescence, cette version compacte peut alors être décompressée en une nouvelle arborescence. Le principe de la compression avec pertes implique que l'arborescence de départ et sa version décompressée ne seront pas isomorphes. Cependant, dans un souci de qualité de résultats, nous ajoutons deux contraintes. La première sur la taille de la compression : nous souhaitons contrôler la taille de la version compressée en imposant le nombre de nœuds du graphe acyclique orienté, c'est-à-dire de la version compressée. La seconde contrainte souhaitée est de permettre d'obtenir une compression avec une perte contrôlée. Pour cela, nous imposons le fait que l'arborescence initiale et l'arborescence reconstruite soient proches autant que possible.

Formalisation du problème

Nous proposons de traiter le problème de compression avec pertes d'une arborescence non-ordonnée utilisant ces redondances.

L'idée du problème est de trouver à partir d'une arborescence T, sa compression R sous forme de DAG , tel que $|R| = c$ et tel que T_c l'arborescence reconstruite à partir de R respecte : $d(T, T_c) < \epsilon$.

Afin de donner une définition plus formelle, nous devons introduire quelques termes.

Nous avons vu la notion d'isomorphisme (Déf. 1.20). Le *ϵ-quasi-isomorphisme* sur les arborescences (Déf. 5.2) est une notion moins contrainte dans laquelle la distance entre les deux arborescences n'est pas nulle mais inférieure à un seuil.

Définition 5.2 (ϵ-quasi-isomorphisme) Soient T_1 et T_2 deux arborescences et $\epsilon \in \mathbb{R}^+$, T_1 et T_2 sont ϵ-quasi-isomorphisme si et seulement si $d(T_1, T_2) < \epsilon$.

Remarque 18 Nous pouvons noter que pour $\epsilon = 0$, le 0-quasi-isomorphisme revient à un isomorphisme.

Remarque 19 Un ϵ-quasi-isomorphisme ne définit pas une relation d'équivalence, en effet elle n'est clairement pas transitive.

Donnons maintenant une définition formelle du problème de la compression avec pertes d'une arborescence non-ordonnée.

Soient T une arborescence, $R(T)$ la compression sans perte de T et $c \in \mathbb{N}$ tel que $c \leq |R(T)|$. Le problème de compression avec pertes d'une arborescence non-ordonnée consiste à trouver un DAG D_T tel $|D_T| \leq c$. Considérons T_c la reconstruction de l'arborescence à partir de D_T et $\mathcal{T}$ l'ensemble des arborescences telles que leurs réductions aient pour taille c, on cherche T_c tel que $D(T, T_C) = \min_{T_k \in \mathcal{T}} D(T, T_k)$. En pratique nous proposons une heuristique, la distance entre T_c et T_k n'est pas minimum.

5.2.2 Algorithme de compression topologique avec perte

La compression avec pertes de données peut être découpée en différentes tâches. Premièrement, il convient de définir la structure de données à compresser. Il faut ensuite identifier les redondances qui pourront être traitées. Une fois cette étape passée, l'action de compression à proprement parler peut avoir lieu. La méthode peut ensuite être évaluée afin de déterminer son efficacité.

Nous nous intéressons dans cette section à la compression d'arborescence non-ordonnée sans étiquette que nous appelons compression topologique. La modélisation de l'architecture des plantes, par exemple, peut être réalisée de cette manière (voir Chapitre 2).

Identification des redondances

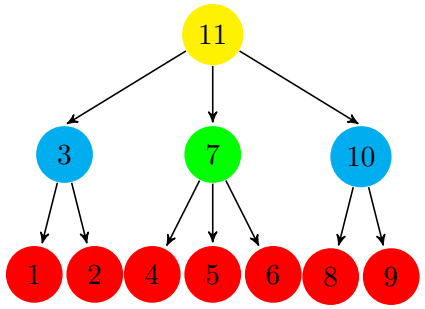
Il convient d'abord de trouver au sein de la structure l'ensemble des redondances. Nous décomposons cette étape en deux sous étapes, la comparaison de sous-arborescences et la création de classes de quasi-équivalences.

Comparaison des sous-arborescences Les méthodes de comparaison présentées Chapitre 3 permettent de calculer les distances entre deux sous-arborescences. Elles se basent sur le principe de la programmation dynamique qui, en fait, calcule l'ensemble des distances entre les sous-arborescences. Calculer la distance de l'arborescence contre elle-même permet alors

de calculer les distances entre chacune de ses sous-arborescences deux à deux. Soit $T = (V, E)$ une arborescence, calculer $d(T,T)$ revient alors à comparer ses sous-arborescences deux à deux $d(T[i],T[j])$ pour tout $(i,j) \in V^2$. Dans le cas non-ordonné nous pouvons utiliser l'algorithme d'édition contrainte d'arborescences non-ordonnées de Zhang [Zha96].

Les résultats des comparaisons sont stockés dans une matrice carrée M de taille $|T| \times |T|$ telle que $M(i,j) = d(T[i],T[j])$ pour tout $(i,j) \in V^2$. Les résultats des distances présents dans la matrice sont au minimum de 0, dans le cas de sous-arborescences parfaitement isomorphes. Un cas trivial est celui de la diagonale de la matrice, représentant la distance entre $T[i]$ et $T[i]$, qui ne contiendra que des valeurs nulles. Tout résultat ϵ strictement supérieur à 0 implique que les deux sous-arborescences ne sont pas strictement isomorphes mais ϵ-quasi-isomorphes. Nous pouvons préciser que plus le résultat est faible, plus les deux sous-arborescences sont proches. De plus, la matrice étant symétrique par rapport à la diagonale, seul le triangle supérieur (ou inférieur) peut être calculé.

Un exemple de matrice relative à une arborescence est donné Fig. 38. Dans cet exemple, les insertions et les délétions ont un score de 1 et les substitutions de 0. Les sous-arborescences complètes enracinées dans les nœuds possédant les numéros 1, 2, 4, 5, 6, 8 et 9 sont isomorphes deux à deux, de même pour les sous-arborescences complètes enracinées dans les nœuds possédant les numéros 3 et 10. Nous pouvons remarquer que la distance entre les sous-arborescences complètes enracinées dans les nœuds possédant les numéros 3 et 7 mais également celle entre 7 et 10 est de 1, ces sous-arborescences sont donc 1-quasi-isomorphes. La plus grande distance présente dans cette matrice est de 10, elle correspond à la distance entre les feuilles et la racine de l'arborescence.



(a)

	1	2	3	4	5	6	7	8	9	10	11
1	0	0	2	0	0	0	3	0	0	2	10
2	-	0	2	0	0	0	3	0	0	2	10
3	-	-	0	2	2	2	1	2	2	0	8
4	-	-	-	0	0	0	3	0	0	2	10
5	-	-	-	-	0	0	3	0	0	2	10
6	-	-	-	-	-	0	3	0	0	2	10
7	-	-	-	-	-	-	0	3	3	1	7
8	-	-	-	-	-	-	-	0	0	2	10
9	-	-	-	-	-	-	-	-	0	2	10
10	-	-	-	-	-	-	-	-	-	0	8
11	-	-	-	-	-	-	-	-	-	-	0

(b)

Figure 38 : Une arborescence et la matrice correspondante. (a) arborescence à compresser, les nœuds avec des couleurs similaires sont isomorphes. (b) Matrice de distances entre les sous-arborescences complètes enracinées des nœuds de l'arborescence.

Classes de quasi-équivalences La matrice M précédemment calculée contient donc toutes les distances de toutes les paires de sous-arborescences de T. Le DAG est construit suivant des

classes de quasi-équivalences déterminées depuis cette matrice de distance afin d'obtenir un DAG possédant c nœuds. Il faut définir les c sous-arborescences quasi-équivalentes qui ne définissent pas une équivalence (*cf.* Remarque 19). Pour cela nous proposons d'utiliser une méthode de classification.

Principe des méthodes de classification

Les méthodes de classifications ont pour but d'opérer des regroupements en classes homogènes d'un ensemble d'objets [JD88]. Les données se présentent sous la forme d'une matrice ayant défini un critère de distance (dissemblance) entre les individus. Il convient alors de procéder au regroupement des individus.

Les méthodes de classification regroupent des algorithmes de partitionnement hiérarchique, basés sur la densité, *etc.* [Mir96, JMF99]. Dans le cas des algorithmes de partitionnement, une partition en k classes, k étant imposé, est réalisée. Lorsque k est modifié, les résultats peuvent changer considérablement. Ainsi, les individus mis au sein d'une même classe avec un k donné peuvent être regroupés dans des classes différentes lorsqu'une classification avec $k+1$ classes est réalisée. Si cette technique est appliquée à la compression, le résultat de la compression peut alors être très altéré avec un changement du nombre de classes. Pour ces raisons, le clustering basé sur le partitionnement n'est pas souhaitable dans notre cas.

Les classifications hiérarchiques

Les classifications hiérarchiques ont pour principe de réaliser des suites de partitions emboîtées [JMF99]. Elles possèdent plusieurs avantages. Tout d'abord, la lecture de la classification permet de déterminer le nombre optimal de classes. Ensuite, le nombre de classes peut être choisi, allant d'une seule à n classes, n étant le nombre d'individus dans la matrice. Enfin, contrairement aux algorithmes de partitionnement précédemment cités, les données classées ensemble ne peuvent plus être classées dans des classes différentes en variant le nombre de classes.

Les classifications hiérarchiques sont descendantes [Gor87, JMF99] ou ascendantes [Gor87, JMF99]. Dans le cas descendant, en partant de l'ensemble des données regroupées au sein d'une classe unique, les classes sont successivement découpées. En revanche, dans le cas ascendant, elles consistent à fournir un ensemble de partitions en classes de moins en moins fines, obtenues par regroupements successifs de parties. Dans les deux cas, les premières classes données par l'algorithme ont une meilleure qualité que les dernières. En utilisant un parallèle avec la compression d'image, nous souhaitons qu'un objet peu compressé ait une distance entre l'objet initial et l'objet reconstruit proche. Nous privilégions donc la classification hiérarchique ascendante.

La méthode de classification hiérarchique ascendante [JMF99] est la suivante :

1. les n individus sont mis dans n classes singletons,
2. les distances entre les classes sont calculées,
3. les deux individus les plus proches sont regroupés formant ainsi $n-1$ classes,
4. les étapes 2 et 3 du processus sont répétées jusqu'à ce que tous les individus soient regroupés au sein d'une seule et même classe.

Ces classifications peuvent être illustrées sous forme d'arbre de classification ou dendrogramme [MS99], la Fig. 39 illustre une classification sous forme de dendrogramme obtenu depuis la matrice M Fig. 38.

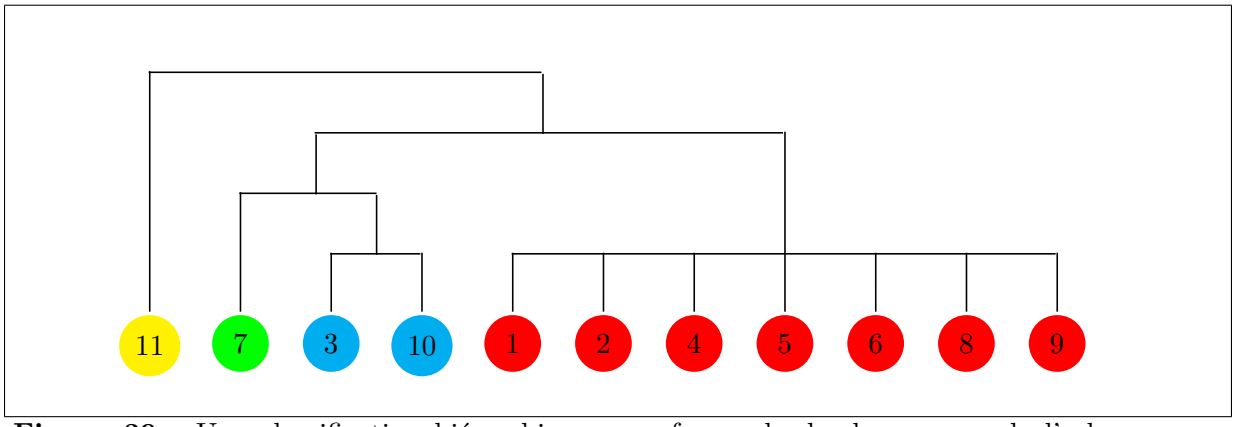


Figure 39 : Une classification hiérarchique sous forme de dendrogramme de l'arborescence présentée Fig. 38(a).

Lorsqu'on souhaite obtenir k classes, il est possible d'arrêter l'algorithme avant que tous les individus soient regroupés, en suivant la méthode de l'Algorithme 12 [Mur83].

Algorithme 12 $Classification(M,k)$: Calcul de k classes dans la matrice M.

Entrées : M : une matrice de distance entre individus, k : le nombre de classes souhaité.
Sorties : $C = \{C_1, \dots C_k\}$ un ensemble de k classes tel que chaque individu soit présent dans une et une seule classe.

1: $C \leftarrow \emptyset$
2: **Pour tout** individu $\in M$ **Faire**
3: Créer une classe C_i
4: $C_i \leftarrow$ {individu}
5: $C \leftarrow C \cup C_i$
6: **Fin pour**
7: **Tant que** $|C| > k$ **Faire**
8: Calculer les dissimilarités entre les classes
9: Rechercher les classes C_i et C_j les plus proches
10: $C \leftarrow C \backslash \{C_i, C_j\} \cup \{C_i \cup C_j\}$
11: **Fin tant que**

Un des points important des méthodes de classification hiérarchique consiste à définir une mesure entre les différentes classes. Ainsi, la ligne 8 de l'Algorithme 12 consiste à calculer les dissimilarités entre les classes. Nous connaissons la distance entre deux individus x et y mais il faut la définir entre deux classes contenant la réunion de plusieurs individus. De nombreuses méthodes existent. Soient C_1 et C_2 deux classes :

1. Le saut minimum (en anglais « single linkage ») qui est défini par

$$dissimilarité(C_1, C_2) = \min_{x \in C_1, y \in C_2} (d(x,y)).$$

L'algorithme de clustering utilisant le saut minimum a une complexité en temps en $O(n^2)$. Cette méthode peut forcer des individus à être dans la même classe, même si la plupart des individus de chaque classe ont une distance grande.

2. Le saut maximum (en anglais « complete linkage ») à l'inverse prend la distance maximum,

$$dissimilarité(C_1, C_2) = \max_{x \in C_1, y \in C_2} (d(x, y)).$$

   La complexité en temps est en $O(n^2 \log n)$. En utilisant le saut maximum, les classes obtenues sont compactes.

3. Le lien moyen (en anglais « average linkage »), situé entre les deux méthodes précédentes utilise la moyenne des distances,

$$dissimilarité(C_1, C_2) = \frac{\sum_{x \in C_1, y \in C_2} (d(x, y))}{|C_1||C_2|}.$$

   Celle-ci est couramment utilisée en phylogénie et plus connue sous le nom d'UPGMA. La complexité en temps est en $O(n^2)$.

4. Enfin, la distance de Ward vise à maximiser l'inertie inter-classe,

$$dissimilarité(C_1, C_2) = \frac{|C_1||C_2|}{|C_1| + |C_2|} d(G_1, G_2)$$

   avec G_1 et G_2 les centres de gravité des classes C_1 et C_2. La complexité est en $O(n^2)$ après un calcul préalable des distances Euclidiennes en $O(pn^2)$ avec p le nombre de variables. On peut donc considérer que ce calcul se fait en $O(pn^2)$.

D'autres mesures, que nous ne détaillerons pas dans ce manuscrit existent.

Après la phase de classification, nous avons donc un ensemble C de k classes avec $k \in [\![1, c]\!]$ telles que $\forall T_1, T_2 \in C_k, d(T_1, T_2) \leq \epsilon$. C est un ensemble de c classes de quasi-équivalences. L'idée est alors que chaque classe représente une sous-arborescence type $\hat{\mathrm{T}}_k$, proche des éléments de la classe C_k.

Construction de la compression

Les redondances ou quasi-redondances étant définies, il reste à les regrouper pour définir une version compressée de l'arborescence d'origine.

Quotientement de l'arborescence Les classes définissent une relation d'équivalence qui permettent de définir un quotientement de l'arborescence à partir duquel le DAG est construit. Les sous-arborescences complète enracinées en v_i dans la même classe sont ainsi mises dans le même quotientement. Les sous-arborescences dans la même classe impliquent donc que la racine de ces sous-arborescences est dans le même nœud du quotientement.

Un exemple est présenté Fig. 40, il s'appuie sur la hiérarchie de la Fig. 38.

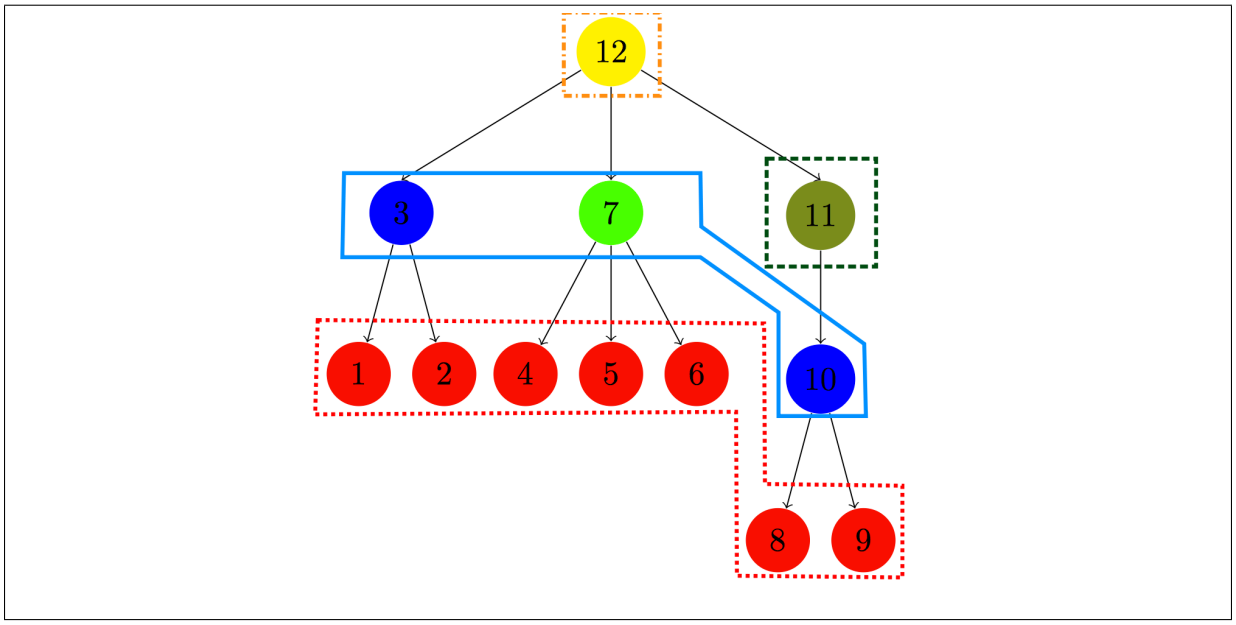


Figure 40 : Les classes issues de la classification définissent un quotientement d'une arborescence.

Construction du DAG Pour chaque classe C_k il convient de trouver une sous-arborescence T_k^* qui sera le représentant de la classe.

Lors de la compression sans perte, les sous-arborescences étant isomorphes, trouver le représentant de la classe dans la compression est évident. Dans le cas de la compression avec pertes, les sous-arborescences peuvent ne pas être isomorphes. C'est le cas pour les trois sous-arborescences issues des nœuds 3, 7 et 10. Cependant, afin d'obtenir une meilleure compression, une seule copie de la classe doit être gardée. Le choix de l'élément représentant est fondamental et complexe.

Plusieurs stratégies peuvent être mises en place :

1. Afin d'obtenir un grand gain d'un point de vue du stockage, il est possible de choisir la plus petite sous-arborescence : $T_k^* = T$ tel que $T \in C_k$ et $|T| = \min_{T_k \in C_k} |T_k|$. Cependant la distance entre l'arborescence de départ et la version reconstruite peut être élevée.
2. Prendre le plus grand : $T_k^* = T$ tel que $T \in C_k$ et $|T| = \max_{T_k \in C_k} |T_k|$, n'est pas optimal que cela soit pour la taille finale du codage mais également pour la perte de qualité.
3. Afin d'avoir une meilleure qualité dans la compression, il serait pertinent de définir la sous-arborescence moyenne, c'est-à-dire celle avec la plus petite distance entre les éléments de la classe : soit $\mathcal{T}$ l'ensemble des arborescences, $T_k^* = \min_{T \in \mathcal{T}} \sum_{T_k \in C_k} d(T_k, T)$. Cependant dans le cas de l'application biologique, nous risquons de créer des sous-arborescences qui n'étaient pas présentes initialement.
4. En nous inspirant des techniques de compression, nous proposons donc de prendre la sous-arborescence la plus fréquente dans les classes considérées et en cas de fréquence égales la plus petite de l'ensemble des plus fréquentes. Notons que nous ne permettons pas de boucle(Déf. 1.12) au sein du graphe ainsi seul les arcs entre deux classes sont autorisés.

En cas de fréquences à égalité, dans le but d'avoir une compression utilisant moins d'espace la plus petite sous-arborescence sera privilégiée. Un exemple est donné Fig. 41.

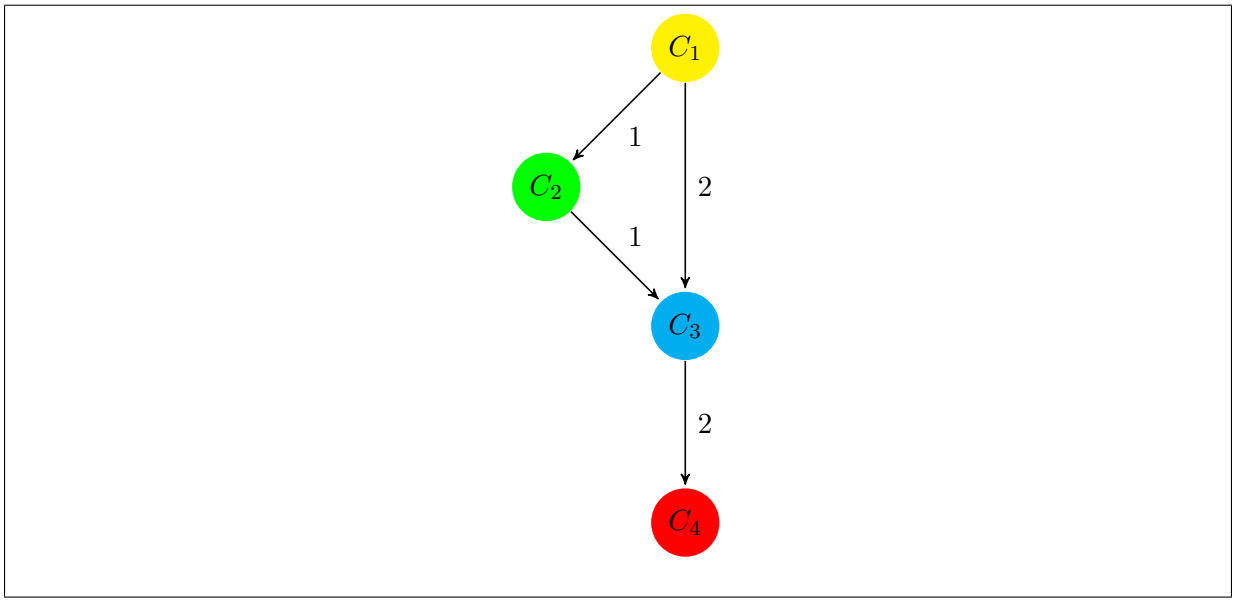


Figure 41 : Construction du DAG à partir du quotientement de la Fig. 40.

Nous allons simplifier le DAG obtenu, en effet ce DAG peut contenir des arcs avec des poids nuls. Dans ce cas, ces arcs sont supprimés du graphe. Il est alors possible d'avoir plusieurs nœuds racines dans le DAG. Ces nouvelles racines ne correspondent pas à la racine dans l'arborescence de départ, la reconstruction de l'arborescence à partir du DAG ayant lieu depuis la racine initiale jusqu'aux feuilles, il n'est plus pertinent de les laisser. Depuis le nœud qui correspond à la racine de l'arborescence, les nœuds tels qu'il n'existe pas un chemin entre la racine de l'arborescence et les feuilles sont supprimés du graphe ainsi que ses arcs adjacents. Ainsi le nombre de nœuds du graphe est borné par le nombre de classes.

Propriété 5 Le graphe obtenu est un DAG.

Preuve Afin de prouver que le graphe obtenu est un DAG , il faut vérifier que la relation d'ancestralité est conservée. Nous devons vérifier une contrainte sur la classification.

Soient $T = (V, E)$ une arborescence, $(v_x, v_y, v_z) \in V^3$ tels que v_x est un descendant de v_y et v_y est un descendant de v_z et C_x, C_y, C_z les classes auxquelles appartiennent les sous-arborescences enracinées en v_x, v_y et v_z, nous souhaitons que $C_x = C_z$ si et seulement si $C_y = C_x$. Par définition de la distance, nous savons que $d(v_x, v_z) \leq d(v_x, v_y)$. Lors du premier passage dans la boucle des lignes 7-10 de l'Algorithme 12, la condition est alors respectée. Tant que les classes C_x, C_y, C_z ne sont pas modifiées, elles le restent. Si ces classes sont modifiées nous avons plusieurs cas :

- C_x et C_y sont regroupées,
- C_y et C_z sont regroupées,
- C_x ou C_y ou C_z est regroupée avec une autre classe notée C_i sous-arborescence complète enracinée en v_i.

Dans les deux premiers cas, la condition est vérifiée. Le dernier cas implique un nouveau calcul des dissimilarités entre les classes. Alors C_i a été plus proche de la classe avec laquelle elle a fusionné que des autres classes, nous noterons la classe la plus proche C_j. Donc $d(v_i, v_j) \leq d(v_i, v_k)$ avec $v_k \neq v_j$. Nous avons trois cas :

- $C_x = C_j$, alors la nouvelle classe a une distance plus faible avec C_y qu'avec C_z,

– $C_y = C_j$, alors la nouvelle classe a une distance avec C_x et C_z plus proche que la distance entre C_x et C_z,
– $C_z = C_j$, alors la nouvelle classe a une distance plus faible avec C_y qu'avec C_x.□

Sur la Fig. 42 est illustrée une arborescence, le graphe obtenu et le graphe simplifié.

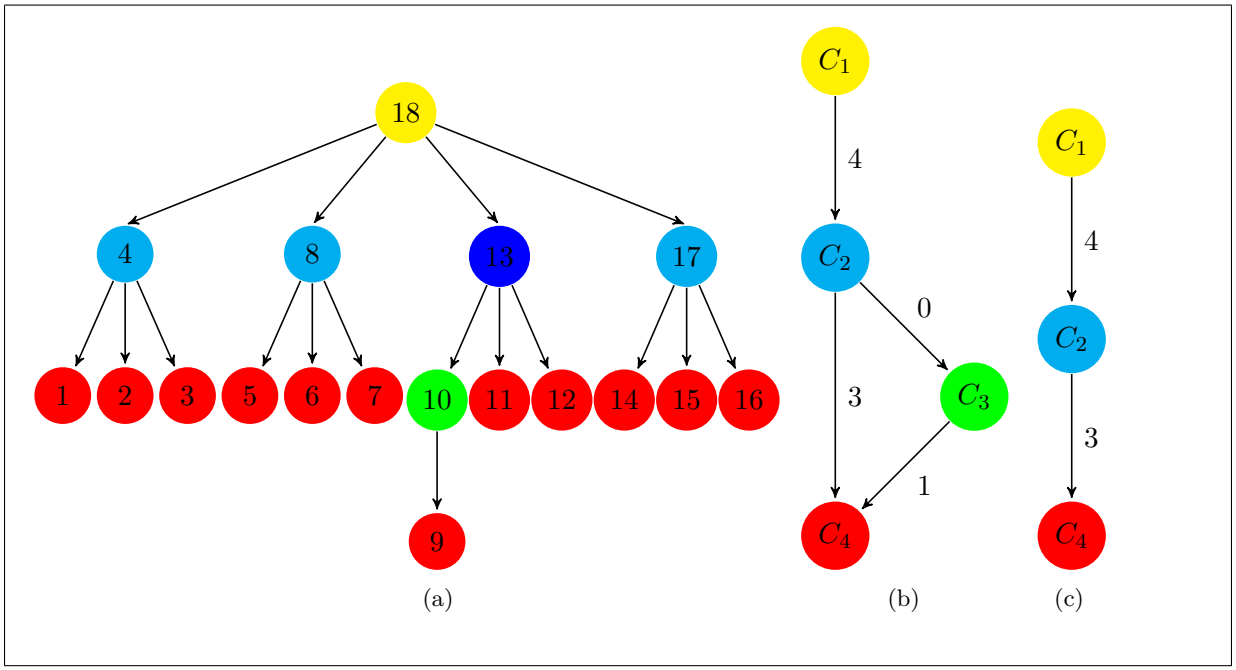


Figure 42 : Simplification du graphe à partir des arcs de poids nuls. (a) Une arborescence à compresser, les classes obtenues après classification de la matrice de distance entre sous-arborescences sont $C_1 = \{18\}$, $C_2 = \{4,8,13,17\}$, $C_3 = \{10\}$ et $C_4 = \{1,2,3,5,6,7,9,11,12,14,15,16\}$. (b) Le graphe obtenu avec les sous-arborescences représentatives est le graphe $G = \{V,E\}$ avec $V = \{C_1,C_2,C_3,C_4\}$ et $E = \{(C_1,C_2,4),(C_1,C_2,4),(C_2,C_3,0),(C_2,C_4,3),(C_3,C_4,1)\}$. (c) Le graphe simplifié qui ne possède que 3 nœuds, C_1, C_2 et C_4.

Algorithme

La méthode de compression est schématisée Fig. 43 et détaillée Algorithme 13 . La complexité est en $O(|T_1||T_2|(\mathrm{Deg}(T_1)+\mathrm{Deg}(T_2))log((\mathrm{Deg}(T_1)+\mathrm{Deg}(T_2)))$, celle ci étant due à la comparaison [Zha96] de la ligne 1, décrite dans le Chapitre 3.

Algorithme 13 *Compression*(T, c) : Calcul de la compression de T sous forme d'un DAG avec une taille inférieure ou égale à c.

Entrées : T : une arborescence non-ordonnée, c : le nombre de classes souhaité.
Sorties : D un DAG tel que $|D| \leq c$.

1: Calculer la distance de Zhang entre T et lui-même, stocker les distances entre sous-arborescences dans une matrice M
2: Classifier M en c classes avec l'algorithme de classification hiérarchique de Ward [War63]
3: Créer les nœuds du DAG D à partir des classes
4: Définir le représentant de chaque classe et mettre à jour D

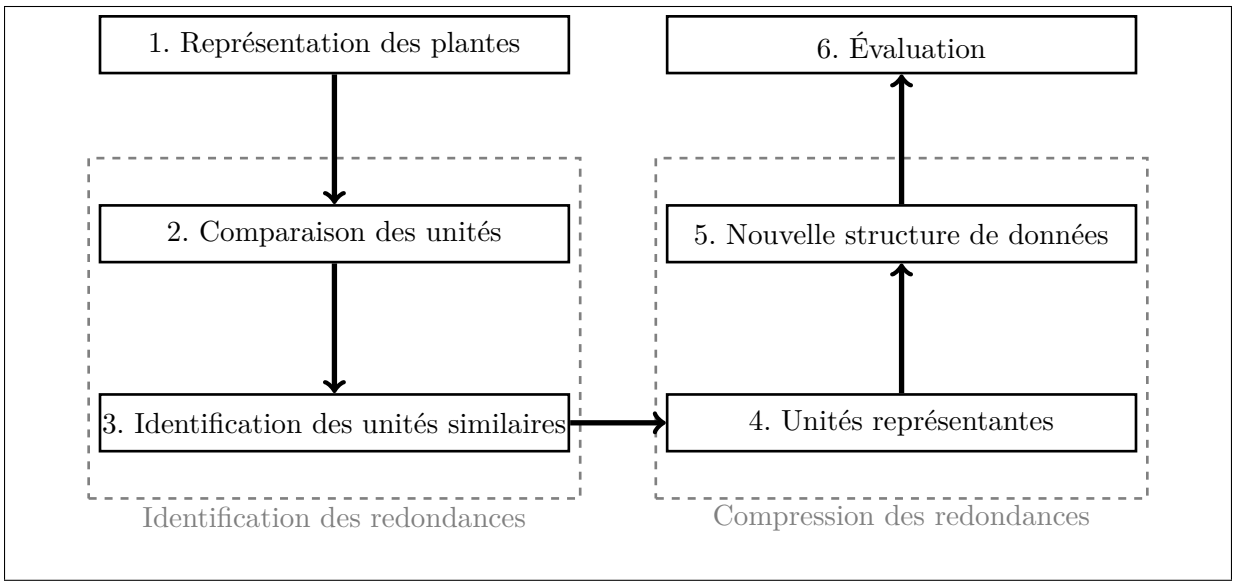


Figure 43 : Schéma de la méthode de compression d'arborescences.

5.2.3 Compression avec pertes d'arborescences non-ordonnées étiquetées

Nous avons vu que les arborescences servent à modéliser des objets biologiques et qu'elles sont généralement étiquetées. Nous proposons une extension de la méthode précédente en prenant en compte des étiquettes sur les nœuds, représentant par exemple la géométrie.

Définissons, la compression sans perte avec étiquettes. Considérons la relation d'équivalence définie pour les arborescences isomorphes avec étiquettes (Déf. 1.21), c'est-à-dire deux sous-arborescences identiques sur l'ensemble des sous-arborescences de l'arborescence T. Nous dirons que les nœuds v_x et v_y dans V sont équivalents si et seulement si $T[v_x]$ et $T[v_y]$ sont isomorphes avec étiquettes et on note par extension $v_x \equiv v_y$. La classe d'équivalence permet de définir la réduction d'une arborescence étiquetée comme dans le cas non étiqueté avec un ajout d'étiquettes sur les nœuds. Le problème de compression avec pertes d'une arborescence étiquetée est alors défini comme suit. Soient T une arborescence étiquetée, $R(T)$ la compression sans perte de T et $c \in \mathbb{N}$ tel que $c \leq |R(T)|$. Le problème de compression avec pertes d'une arborescence non-ordonnée étiquetée consiste à trouver un DAG D_T tel $|D_T| \leq c$. Considérons T_c la reconstruction

de l'arborescence avec étiquettes à partir de D_T et $\mathfrak{T}$ l'ensemble des arborescences étiquetées telles que leurs réductions aient pour taille c, on cherche T_c tel que $D(T,T_C) = \min_{T_k \in \mathfrak{T}} D(T,T_k)$.

Il est possible d'envisager une compression avec pertes de la topologie mais en conservant les étiquettes (ou un sous-ensemble de celles-ci), par exemple dans un dictionnaire. Cependant, en pratique, les étiquettes sont généralement définies sur un alphabet très grand et ce type de compression ne semble pas applicable pour notre cas d'étude. La résolution de ce problème bien que proche du précédent est cependant bien plus complexe et indiscutablement lié à l'application.

Identification des redondances

Algorithme de comparaison d'arborescences avec étiquettes La comparaison peut être réalisée avec une méthode équivalente à celle présentée dans la section précédente. Cependant les matrices de score pour les coûts des opérations d'édition sont dépendantes des étiquettes.

L'algorithme de Zhang [Zha96] présenté Section 3.3.1 peut donc s'appliquer si une mesure de dissimilarité entre étiquettes a été définie. Le détail des méthodes de définitions de ces mesures dépend fortement de l'application et ne sont pas le propos de ce manuscrit. Nous ne présenterons que celles liées à l'architecture des plantes que nous avons mises en pratique. Dans le cas de l'architecture des plantes étiquetée avec la liste des paramètres de géométrie, il convient de trouver une mesure entre deux géométries. Une première solution consiste à comparer les étiquettes de même type ensemble, par exemple la largeur entre deux nœuds et à affecter un coefficient pour chaque élément de l'étiquette. Nous obtenons ainsi des formules du type :

$$\text{coût}(\text{sub}(v_i, v_j)) = \sum_{i \in (\text{Liste des étiquettes})} \alpha_i \times \text{distance}(\text{entre les étiquettes de même type}),$$

avec α_i le coefficient associé à l'étiquette i. Une autre méthode consiste à modéliser l'énergie nécessaire pour passer d'une géométrie à une autre. Nous pouvons obtenir des formules assez proches en prenant indépendamment les énergies pour transformer chaque paramètre ou définir une énergie globale de transformation. Ces deux techniques nécessitent cependant la potentielle manipulation d'un grand nombre de paramètres.

Classes de quasi-équivalences avec étiquettes La classification peut se réaliser de manière identique au cas sans étiquette à partir de la matrice créée lors de l'exécution de l'algorithme de Zhang [Zha96].

Construction de la compression

Le quotientement est réalisé comme dans le cas topologique. La construction du DAG en revanche pose un nouveau problème pour la définition de l'étiquette représentative de la classe. Ce problème n'a pas de solution absolue. Selon le contexte, le résultat optimal peut varier. Nous pouvons proposer trois méthodes :

1. Une première méthode consiste à de prendre l'étiquette la plus fréquente. Bien que cette solution soit envisageable lorsque les étiquettes sont discrètes, dans le cas de variables continues, elle n'est plus adaptée.
2. Une seconde méthode a pour principe la définition d'une étiquette que nous dirons par plage. Dans le cas d'étiquettes géométriques, l'étiquette par plage consiste à donner un

intervalle dans lequel les étiquettes sont comprises. Pour la phase de décompression une fonction probabiliste peut être utilisée.

3. Il est également possible de prendre la moyenne des étiquettes, lorsqu'une telle notion peut être définie, dans un cas d'étiquettes numériques par exemple.

Dans certains cas, il n'est pas pertinent de fusionner les étiquettes. Prenons par exemple, le cas de l'orientation des branches. Nous avons vu, Section 2.2.2, qu'il existait deux modélisations de l'orientation, une absolue et une relative, illustrées Fig. 21. Si l'orientation est définie de façon absolue, chaque orientation sera différente dans l'arborescence de départ et une compression utilisant la même orientation pour différentes parties de la plante entrainera des chevauchements. Utiliser une orientation relative ne résout pas l'ensemble des problèmes. Prenons le cas de la Fig. 44. Si l'orientation des trois feuilles est mise en commun, elles se chevaucheront.

Figure 44 : Cas d'une compression de l'orientation problématique.

Ces étiquettes dites incompatibles sont alors placées sur les arcs du DAG. Dans le but d'obtenir un objet très compressé, nous ne pouvons pas garder l'ensemble des étiquettes sur les arcs. Nous les plaçons alors sur l'arc entrant. Si la racine possède une telle étiquette, une racine fictive ayant un arc sortant vers l'ancienne racine peut-être ajoutée. Afin de ne pas trop augmenter l'espace mémoire occupé par le DAG, un compromis consiste à garder autant d'étiquettes sur l'arc que la valuation de l'arc. Pour sélectionner les étiquettes pertinentes, nous proposons de réaliser une seconde classification avec n_e classes, n_e correspondant à la valeur sur l'arc. Cette classification a lieu uniquement sur les étiquettes des nœuds de la classe. La valeur de chaque étiquette correspond, comme dans le cas précédent, soit à l'étiquette la plus fréquente de la classe, soit à une étiquette par plage, soit à une étiquette moyenne. Sur la Fig. 45, (a) représente une arborescence, on suppose que les classes sont définies par la profondeur du nœud, le DAG (b) obtenu avec les étiquettes sur les arcs, les étiquettes sur les arcs correspondent à l'étiquette la plus fréquente dans les classes obtenues. En pratique, l'utilisateur doit définir les étiquettes qui ne pourront pas être compressées au niveau des nœuds mais au niveau des arcs et doit le définir en entrée de l'algorithme.

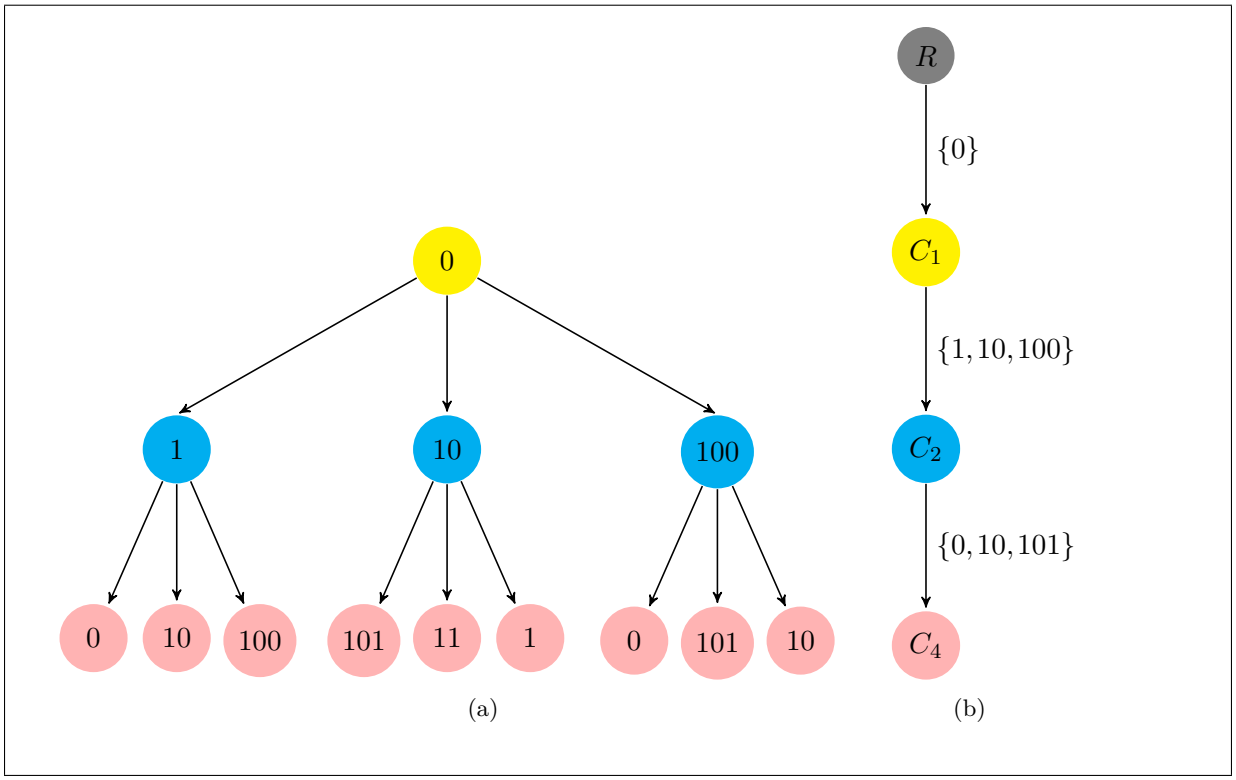


Figure 45 : Construction du DAG dans le cas d'étiquettes incompatibles. (a) Une arborescence étiquetée, les valeurs des étiquettes sont placées sur le nœud. (b) La construction du DAG avec étiquettes incompatibles sur les arcs.

5.3 Application de la compression à l'architecture des plantes

Mondet *et al.* [MCM+09] ont également montré de l'intérêt pour la compression d'architecture des plantes dans le but de transmettre des données à travers le web. Dans notre cas, une compression purement géométrique a été proposée dans laquelle les branches sont encodées seulement par des différences avec un modèle ou en définissant une géométrie moyenne regroupant des branches avec une complexité similaire.

Les plantes présentent généralement des structures qui impliquent des modèles complexes. Bien que les structures des plantes soient répétitives, elles ne sont pas identiques : les organes, les axes et les branches à différentes positions sont souvent similaires. Le caractère répété des plantes a été très étudié par les concepts biologiques tels que, par exemple, la réitération dans les arbres introduite dans les années 70 [HOT78, Bar91], ou la relation de paracladia introduite par Frijters et Lindenmayer [FL76] qui décrit la redondance des fleurs.

Cette propriété est inhérente à la compréhension de la biologie des plantes et joue également un rôle important dans l'analyse formelle [GC98] et la simulation [dREC89, PL90] des plantes. Ce caractère répétitif de la structure des plantes a été très exploité dans la modélisation des plantes basées sur les fractales (par exemple [Smi84, PH89, Bar00, FGP05]). En premier lieu Prusinkiewicz *et al.* [PL90] ont utilisé la notion de projection des branches dans lequel les unités fondamentales sont dupliquées dans les inflorescences. Cependant, les répétitions, comme la symétrie ne sont pas facilement quantifiables, cette propriété est généralement considérée comme

présente ou absente. Les plantes réelles ou simulées peuvent être autosimilaires uniquement à un certain degré. En botanique, les premières études de structures emboîtées, semblables à des structures fractales, sont dues à Arber [Arb50]. Néanmoins, identifier l'organisation autosimilaire chez les plantes est un problème difficile. Celle-ci se développe au court d'une plus longue période et est donc soumise à l'influence de l'environnement. Les systèmes ramifiés des arbres résultent cependant de processus répétitifs, dans lesquels différents méristèmes suivent des séquences d'états similaires et produisent des structures similaires. Ces structures sont caractérisées par les biologistes en termes de programme morphogénétique ou d'âge physiologique. Plusieurs tentatives de quantification de l'autosimilarité des plantes, utilisant une représentation sous forme d'arborescence ont été faites durant la dernière décennie [Pru04, FGP05, BGP+06].

Afin de tester la qualité de la méthode, nous l'avons appliquée à l'architecture des plantes. Dans ce chapitre, nous donnons, Section 5.3.1 la méthode d'évaluation de la compression que nous avons utilisée. Dans la Section 5.3.2 nous appliquons la méthode à différents types d'architecture des plantes.

5.3.1 Évaluation de la compression

Le taux de compression est une mesure de la performance d'un algorithme de compression de données informatiques.

La qualité de la compression est souvent exprimée en utilisant le ratio de compression [Sal07] :

$$cr = \frac{\text{taille de la sortie}}{\text{taille de l'entrée}}.$$

Pour une arborescence enracinée étiquetée non-ordonnée T et un DAG correspondant à la compression D dans notre cas, nous pouvons également considérer dans un premier temps que le nombre de nœud est la taille de référence. Le ratio de compression cr est le nombre de nœuds dans D divisé par le nombre de nœuds dans T :

$$cr = \frac{|D|}{|T|}.$$

Nous pouvons aussi considérer la compression au niveau des arcs où $|T|$ et $|D|$ sont respectivement le nombre d' arc dans T et D. Il existe cependant des mesures plus précises de la taille des graphes.

Considérons qu'une arborescence est codée par des pointeurs, ce qui est généralement le cas. La taille du pointeur vers son parent est généralement une constance. Dans le cas topologique, nous avons donc le nombre de nœuds fois le pointeur. Dans le cas géométrique, la taille correspond au nombre de nœuds fois la taille du pointeur fois la taille des étiquettes.

Une arborescence non-ordonnée étiquetée avec n nœuds peut être stockée en $O((n-2)log_2 n)$ bits (un bit contenant un caractère 0 ou 1) en utilisant le codage de Prüfer [LPV03]. Un encodage linéaire existe dans le cas des arborescences ordonnées [Jac89]. Bien qu'elles soient compactes, de tels encodages ne sont pas très efficaces pour les applications.

Une adaptation du code Prüfer a été proposée sur les DAG avec étiquettes sur les nœuds [Ste03]. Malheureusement, ils ne prennent pas en compte les étiquettes sur les arcs. Une variante peut être proposée pour un DAG avec n sa taille et E son ensemble d'arcs $(\log_2(n)+\log_2(\max(\sigma(e_i)|e_i \in E)+1)\times|E|-1$ bits avec $\sigma(e_i)$ les étiquettes sur les arcs, suivant l'Algorithme 14. Un exemple est donné Fig. 46. Le codage se déduit à partir du tableau, pour chaque nœud on écrit pour chaque arc le code du parent selon un ordre postfixe, la signature de l'arc puis un 0 si l'arc suivant

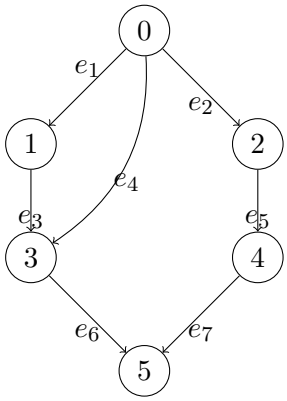
concerne le même nœud ou 1 si ça en considère un nouveau. Notons que nous pouvons ne pas coder la racine qui n'a pas d'arc entrant. Sur cet exemple cela donne :

- Nœud 0 : ne rien coder
- Nœud 1 :
  - père du nœud 1 : 0
  - signature de l'arc : 2
  - changement de nœud : 1
- Nœud 2 :
  - père du nœud 2 : 0
  - signature de l'arc : 1
  - changement de nœud : 1
- Nœud 3 :
  - père du nœud 3 : 0
  - signature de l'arc : 2
  - changement de nœud : 0
  - père du nœud 3 : 1
  - signature de l'arc : 4
  - changement de nœud : 1
- *etc.*

Algorithme 14 *EncodageDAG(D)* : Calcul d'un encodage efficace d'un DAG avec des étiquettes sur les nœuds et les arcs

Entrées : $G = (V, E)$ un DAG.
Sorties : L'encodage du DAG.

```
Code ← nouvel encodage binaire vide
CompVertices ← 0
Pour tout (v_i ∈ V) Faire
  CompEdges ← 0
  Pour tout (e_k ∈ E tel que e_k = (v_j, v_i)) Faire
    Code ← Concaténer(Code, encodage binaire de (v_j) sur log_2(n) bits)
    Code ← Concaténer(Code, encodage binaire de (σ(e_k) − 1) sur log_2(max(σ(e_i)|e_i ∈ E)) bits)
    Si (CompEdges < nbEdgesVi) Alors
      Code ← Concaténer(Code,0)
    Fin si
  Fin pour
  Code ← Supprimer(0)
  CompVertices ← CompVertices + 1
  Si (CompVertices < n) Alors
    Code ← Concaténer(Code,1)
  Fin si
Fin pour
Retourner Code
```



(a)

Nœuds : n_i^2	0	1	2	3	3	4	5	5
arcs : $e_i = (n_i^1, n_i^2)$	-	1	2	3	4	5	6	7
Pères : n_i^1	-	0	0	1	0	2	3	4
Signatures : $\sigma(e_i)$	-	2	1	4	2	3	3	1

(b)

Figure 46 : Un DAG, la représentation sous forme de tableau permettant un encodage efficace basé sur [Ste03]. (a) Un DAG avec 6 nœuds et 7 arcs avec étiquettes. $\sigma(e_1) = 2$, $\sigma(e_2) = 1$, $\sigma(e_3) = 4$, $\sigma(e_4) = 2$, $\sigma(e_5) = 3$, $\sigma(e_6) = 3$ and $\sigma(e_7) = 1$. (b) La représentation sous forme de tableau des valeurs à coder, sur les lignes le numéro du nœud, de l'arc, du père du nœud et la signature de l'arc, chaque colonne correspondant à un arc.

Pour évaluer la distance entre l'arborescence original et l'arborescence reconstruite T_c, nous utilisons la distance d'édition contrainte entre arborescences non-ordonnées [Zha96] entre T et T_c. Un dernier moyen d'évaluation est d'évaluer les différences de rendus. Des travaux sont en cours pour l'architecture des plantes tel que [PGP10].

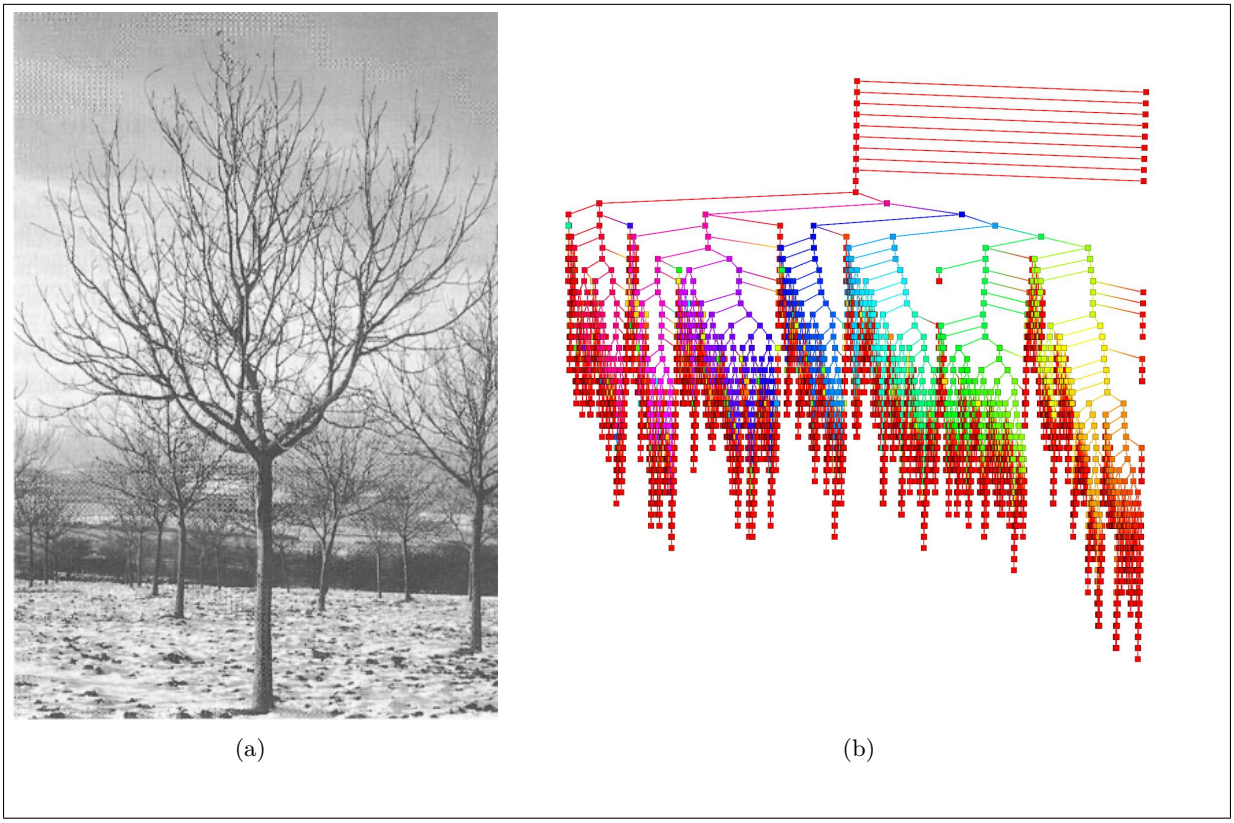
5.3.2 Implémentation et base de données

La méthode de compression a été implantée sous le logiciel Vplants dans le package self-similarity. VPlants [1] est un ensemble d'outils pour l'analyse, la modélisation et la simulation d'architecture des plantes et de son développement à différentes échelles (tissu, organe, plante, *etc.*), développé avec d'OpenAlea [PDKB+08], un projet open source principalement destiné à la communauté de modélisation des plantes par l'équipe Virtual Plants à Montpellier. Nous utilisons une modélisation de l'architecture des plantes sous forme d'arborescence non-ordonnée étiquetée sur les nœuds. Les étiquettes comportent plusieurs informations que sont l'orientation, sous forme de quaternions [Ple89], la longueur et le diamètre sous forme d'entiers.

1. Site de Vplants : http ://openalea.gforge.inria.fr/wiki/doku.php ?id=packages :vplants :vplants

Compression topologique

Dans un premier temps, nous avons compresser uniquement la topologie à partir d'un noyer digitalisé [SRG97]. Sa photographie ainsi qu'une représentation de sa topologie sont présentées Fig. 47. Deux nœuds de la même couleur signifient que les sous-arborescences enracinées en ces nœuds sont isomorphes. L'arborescence d'origine possède 6427 nœuds, le DAG représentant la compression sans perte a une taille de 920. L'histogramme (Fig. 48) représentant les nombres d'occurrences des distances entre sous-arborescences permet de confirmer que de nombreuses sous-arborescences sont isomorphes (distance à 0).



(a) (b)

Figure 47 : Photographie du noyer extraite de [SRG97] et sa représentation topologique réalisée avec le logiciel tulip [Aub03].

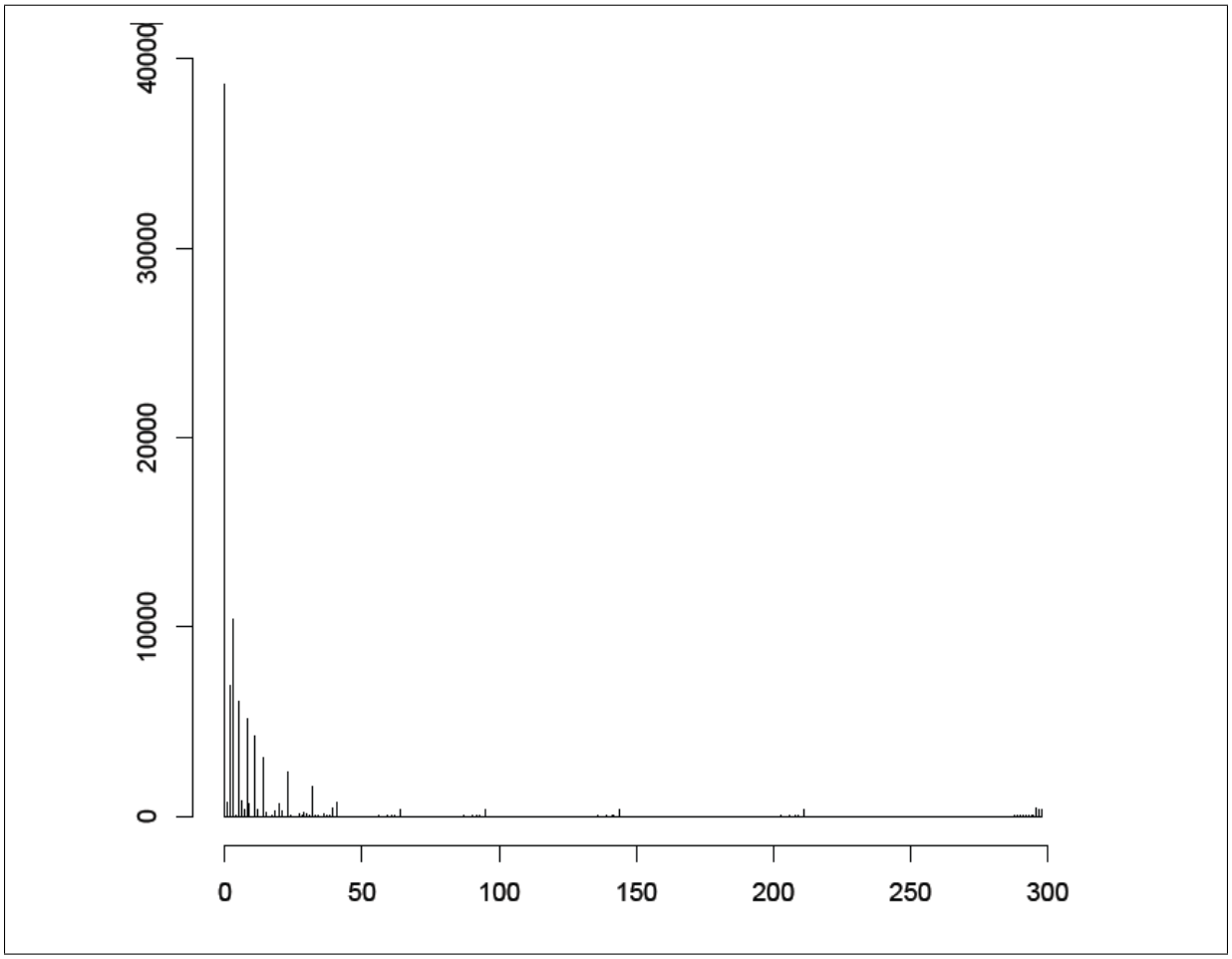


Figure 48 : Histogramme de la matrice issue des comparaisons des sous-arborescences du noyer. En abscisse la valeur de la distance d'édition entre sous-arborescences, en ordonnée le nombre de fois où cette valeur est présente dans la matrice.

La compression avec pertes permet cependant de diminuer le nombre de nœuds. La Fig. 49 représente le pourcentage du nombre de nœuds dans le DAG compressé avec pertes par rapport au nombre de nœuds de la compression sans perte en fonction de la distance normalisée par la somme de la taille des deux arborescences. Nous pouvons remarquer que pour une diminution de 50% des nœuds, la distance entre l'arborescence d'origine et l'arborescence reconstruite reste faible (inférieur à 0,2).

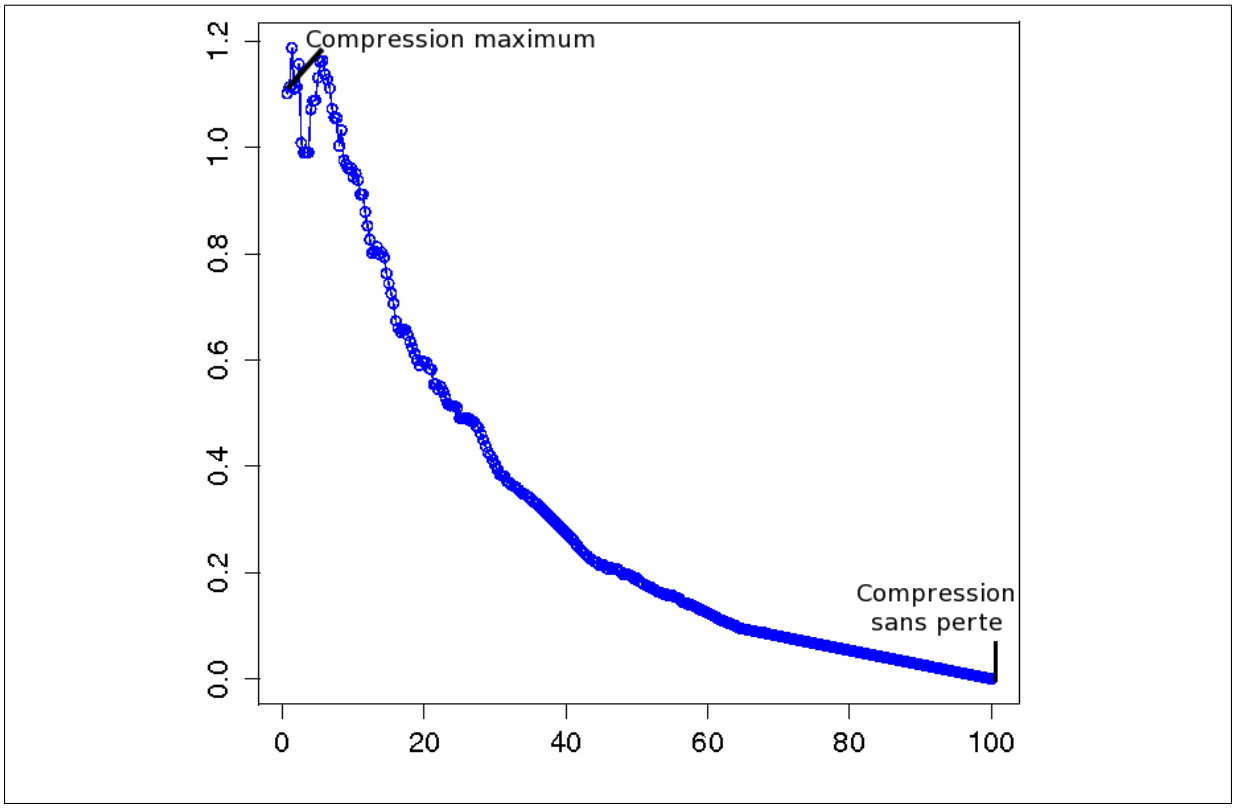


Figure 49 : Évolution en abscisse du pourcentage du nombre de nœud du DAG compressé avec pertes par rapport au nombre de nœuds du DAG issu de la compression sans perte en fonction de la distance normalisée par la taille des sous-arborescences (en ordonnée).

Compression géométrique

Nous avons appliqué la compression géométrique au noyer précédemment cité (Fig. 50). La technique de compression sans perte ne permet pas de compresser de nœuds, les étiquettes étant toutes différentes. Pour une compression avec pertes avec un ratio de compression de 0,66, les arborescences initiales et reconstruites ont des représentations proche. Les branches proches des feuilles sont cependant celles qui ont été le plus altérées.

L'inflorescence du lilas (la disposition des fleurs sur les branches du lilas) a été modélisée en utilisant la notion de mise en correspondance des branches [PMKL01] : soient deux branches du même ordre, la plus petite branche est identique, aux effets de tropisme près, c'est-à-dire des orientations, à la partie supérieure de la branche du dessus. Cela implique la construction d'une plante parfaitement auto-emboitée. Cela permet un meilleur taux de compression. Cependant quand un certain seuil est dépassé, la plante compressée montre des effets visibles de perte. Un exemple de compression du lilas est présenté Fig. 51.

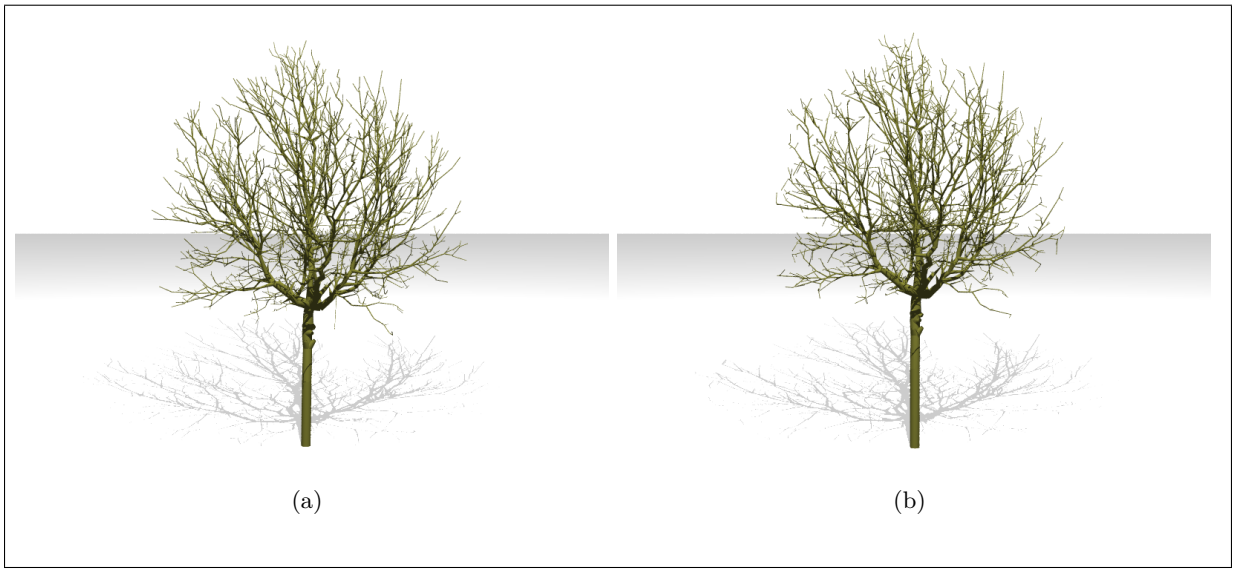


Figure 50 : Résultat de la compression topologique et géométrique avec pertes du noyer. (a) Représentation du noyer sans compression. (b) Représentation du même noyer après compression, cr = 0,66.

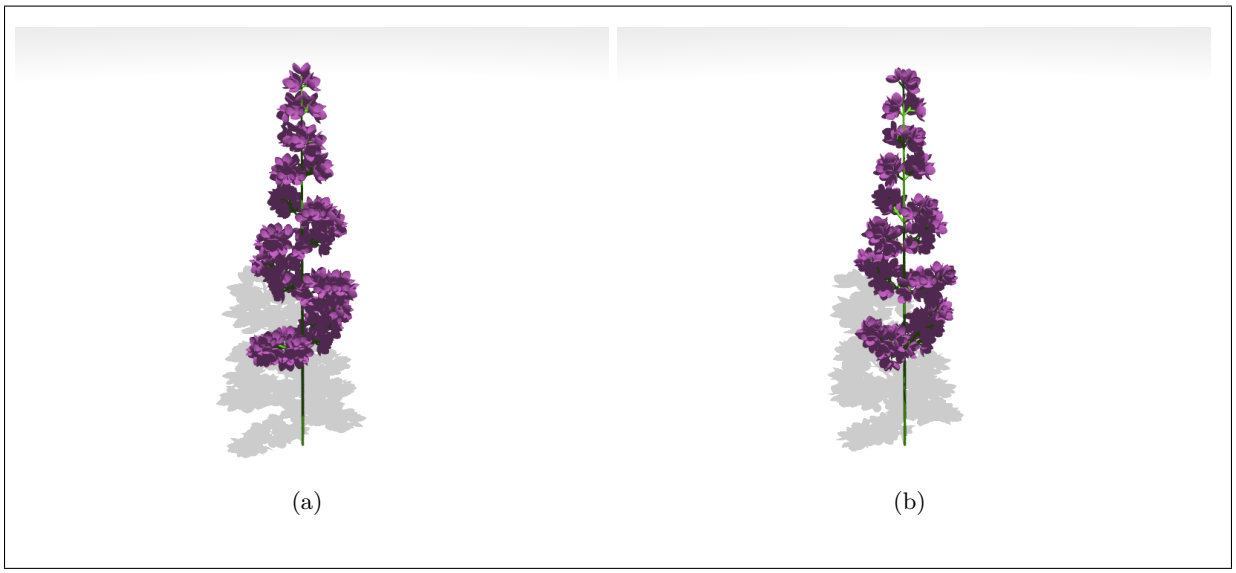


Figure 51 : Résultat de la compression topologique et géométrique avec pertes du lilas. (a) Représentation du lilas sans compression. (b) Représentation du même lilas après compression, cr=0,58.

Cette technique est complémentaire à l'approche de compression orientée géométriquement telle que celle développée par [MCM+09], dans laquelle les modèles géométriques de branches entières sont compressés, ils ne conservent cependant pas les unités fondamentales. Les deux techniques pourraient en principe être combinées et permettraient une compression efficace de l'architecture des plantes.

Conclusion et perspectives

En partant du constat de l'augmentation de la quantité de données biologiques, nous avons présenté dans ce mémoire des méthodes de traitement des masses de données biologiques arborescentes redondantes.

Stockage des données

Nous avons étudié la compression avec pertes des données biologiques arborescentes non-ordonnées avec et sans étiquettes sur les nœuds en imposant que l'objet compressé soit un DAG de taille définie en entrée de notre problème. Pour cela, il a fallu déterminer les sous-arborescences proches à l'aide du calcul d'une distance d'édition. Contrairement aux sous-arborescences isomorphes, elles ne définissent pas une relation d'équivalence. Nous avons donc proposé de calculer des classes de quasi-équivalences à l'aide d'une méthode de classification. Une méthode de sélection d'un représentant de chaque sous-arborescence et étiquette a dû être mis en place. Nous avons finalement proposé une technique de compression qui a été implémentée et testée sur des représentations de l'architecture des plantes.

Bien que notre méthode ait été testée sur quelques plantes réelles et simulées, afin d'analyser la robustesse de la méthode, la mise en place de tests de plus grandes importances sur différentes données nous permettrait d'étudier l'influence de différentes architectures de plantes sur la compression. De plus, l'étude d'un potentiel lien entre compression et autosimilarité pourrait aider à la quantification de l'autosimilarité dans de telles structures. En effet, nous pouvons supposer que plus les données sont autosimilaires, plus la compression sera efficace. D'un point de vue algorithmique, la quantification de la distance entre arborescence initiale et arborescence reconstruite et le développement d'une méthode non heuristique permettant de trouver l'arborescence la plus proche de l'arborescence initiale sont deux pistes primordiales à explorer.

L'extension de cette méthode aux arborescences ordonnées permettrait enfin la compression d'autres objets modélisés par des arborescences telles que les structures secondaires d'ARN. La compression avec pertes des arborescences étiquetées pose de nombreuses interrogations quand au choix d'une étiquette représentative. Une discussion avec des experts sur les données biologiques étudiées s'avère enfin fondamentale pour l'optimisation de notre méthode.

Recherche dans des bases de données

En nous intéressant à la recherche dans des bases de données d'un ensemble de structures arborescentes proches d'une structure arborescente requête, nous avons défini le principe de

chainage sur les arborescences ordonnées. La méthode proposée possède une complexité linéaire en temps et en espace avec un ensemble S de m graines de taille cumulée bornée. Il est néanmoins nécessaire de définir le concept de graines sur ces structures. D'un point de vue applicatif, nous proposons, dans le cas des structures secondaires d'ARN des graines (l, d)-centrées.

Là encore, malgré une analyse réductrice, il convient de tester l'influence des paramètres l et d sur les différentes familles de cette molécule. D'un point de vue théorique, comme dans le cas des graines sur les séquences d'autres modèles de graines, telles que des graines espacées, pourraient être envisagés et l'étude de leurs propriétés parait indispensable. Enfin, la finalité de nos travaux est le développement d'une méthode de type FASTA ou BLAST, c'est-à-dire d'un outil permettant à un utilisateur à partir de la structure secondaire d'ARN, requête de son choix, de déterminer les structures secondaires d'ARN qui sont proches et de leur donner un score de confiance.

Cette méthode peut ensuite être étendue à d'autres objets biologiques modélisés par des structures arborescentes non-ordonnées telles que l'architecture des plantes. Pour cela, la méthode de chainage doit être étendue aux arborescences semi-ordonnées et non-ordonnées et des modèles de graines pertinents doivent être définis.

Navigation dans des bases de données

Enfin, nous avons vu que les données biologiques peuvent se présenter sous forme de séquences. Nous nous sommes d'ailleurs inspirés des méthodes de traitement sur ce type de modélisation. Cependant, elles peuvent également être modélisées par des graphes. Une perspective ambitieuse est d'étendre les notions de filtrage et de compression à des modèles plus complexes. Du point de vue de la compression, des méthodes ont été proposées pour identifier des structures tertiaires à l'aide de sous-graphes maximaux isomorphes mais également pour calculer la distance entre graphe à partir des sous-graphes communs maximaux. Cependant, le problème d'isomorphisme de sous-graphe commun maximum est NP-complet. Enfin, développer une méthode de filtrage sur une compression sous forme de DAG demande une définition de graine (sous-graphe connexe, sous-graphe étant la compression d'une arborescence, *etc.*) et surtout une extension de la méthode de chaînage aux DAG enracinés. Ce gain significatif dans le traitement des données biologiques à la fois lors de la phase de stockage et de la phase d'analyse serait une réponse à l'augmentation de la masse de données biologiques disponible.

Index

Références bibliographiques

[ACFG] Julien Allali, Cédric Chauve, Pascal Ferraro et Anne-Laure Gaillard : Efficient chaining of seeds in ordered trees. *Journal of Discrete Algorithms*. (à apparaitre). 59

[ACFG11] Julien Allali, Cédric Chauve, Pascal Ferraro et Anne-Laure Gaillard : Efficient chaining of seeds in ordered trees. *In Proceedings of the 21st international conference on Combinatorial algorithms*, IWOCA'10, pages 260–273, Berlin, Heidelberg, 2011. Springer-Verlag. 59, 86

[Ada92] W. T. Adams : Gene dispersal within forest tree populations. *New Forests*, 6:217–240, 1992. 10.1007/BF00120646. 24

[AGM+90] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers et David J. Lipman : Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990. 5, 59, 60, 61, 62

[All04] Julien Allali : *Comparaison de structures secondaires d'ARN*. Thèse de doctorat, Université de Marne-la-Vallée, 2004. 41, 54

[Alt89] Sidney Altman : Ribonuclease P: an enzyme with a catalytic RNA subunit. *Advances in Enzymology and Related Areas of Molecular Biology*, 62(1):36, 1989. 23

[Alu05] Srinivas Aluru, éditeur. *Handbook of Computational Molecular Biology*. CRC Press, 2005. 59, 60

[Arb50] A Arber, éditeur. *The natural philosophy of plant form*. Cambridge: Cambridge University Press, 1950. 109

[AS05] Julien Allali et Marie-France Sagot : A new distance for high level RNA secondary structure comparison. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2:3–14, janvier 2005. 38

[AS06] Tero Aittokallio et Benno Schwikowski : Graph-based methods for analysing networks in cell biology. *Briefings in Bioinformatics*, 7(3):243–255, 2006. 25

[Aub03] David Auber : Tulip : A huge graph visualisation framework. *In* P. Mutzel et M. Jünger, éditeurs : *Graph Drawing Softwares*, Mathematics and Visualization, pages 105–126. Springer-Verlag, 2003. x, 112

[BA01] Elizabeth Bullitt et Stephen R. Aylward : Analysis of time-varing images using 3-D vascular models. *In Proceedings of the 30th on Applied Imagery Pattern Recognition Workshop*, pages 9–16, Washington, DC, USA, 2001. IEEE Computer Society. 3

[Bar91] Daniel Barthélémy : Levels of organization and repetition phenomena in seed plants. *Acta Biotheoretica*, 39(3-4):309–323, 1991. 108

[Bar00] Michael Fielding Barnsley : *Fractals everywhere*. Academic press, London (UK), second édition, 2000. 108

[BEH91] Daniel Barthélémy, Claude Edelin et Francis Hallé : *Canopy architecture*, pages 1–20. John Wiley & sons, Raghavendra A. S. edition édition, 1991. 34

[Bel93] Adrian D. Bell : *Les plantes à fleur. Guide morphologique illustré*. Masson, 1993. 34

[BGP+06] Frédéric Boudon, Christophe Godin, Olivier Puech, Christophe Pradal et Hervé Sinoquet : Estimating the fractal dimension of plants using the two-surface method: An analysis based on 3D-digitized tree foliage. *Fractals*, 14(3):149–163, 2006. 109

[Bil05] Philip Bille : A survey on tree edit distance and related problems. *Theoretical Computer Science*, 337:217–239, 2005. 4, 46

[BJM61] Sydney Brenner, François Jacob et Matthew Meselson : An unstable intermediate carrying information from genes to ribosomes for protein synthesis. *Nature*, 192:1227–1232, 1961. 23

[BLM08] Giorgio Busatto, Markus Lohrey et Sebastian Maneth : Efficient memory representation of XML document trees. *Information Systems*, 33(4-5):456–474, 2008. 94

[BMRB96] Bernadette Bouchon-Meunier, Maria Rifqi et Sylvie Bothorel : Towards general measures of comparison of objects. *Fuzzy Sets System*, 84:143–153, Décembre 1996. 41

[Bon09] Michaël Bon : *Prédiction de structures secondaires d'ARN avec pseudo-nœuds*. Thèse de doctorat, École polytechnique, 2009. 29

[Bou04] Frédéric Boudon : *Représentation géométrique de l'architecture des plantes*. Thèse de doctorat, Université de Montpellier II, 2004. 36, 37

[Bro07] Daniel G. Brown : *A survey of seeding for sequence alignment*, chapitre 6. John Wiley and Sons, 2007. 59, 62

[BSM+09] Emmanuelle M. Bayer, Richard S. Smith, Therese Mandel, Naomi Nakayama, Michael Sauer, Przemyslaw Prusinkiewicz et Cris Kuhlemeier : Integration of transport-based models for phyllotaxis and midvein formation. *Genes & Development*, 23(3):373–384, 2009. viii, 24

[But06] John M. Butler : Genetics and genomics of core short tandem repeat loci used in human identity testing. *Journal of Forensic Sciences*, 51(2):253–265, 2006. 89

[CB86] Thomas R. Cech et Brenda L. Bass : Biological catalysis by RNA. *Annual Review of Biochemistry*, 55:599–629, 1986. 23

[CB97] Yves Caraglio et Daniel Barthélémy : Revue critique des termes relatifs à la croissance et à la ramification des tiges des végétaux vasculaires. *In* I. N. R. A. édition, éditeur : *Modélisation et simulation de l'architecture des végétaux*, chapitre Part I, pages 11–87. Science Update, 1997. 34

[CCB+06] Angélique Chanal, Virginie Chapon, Karim Benzerara, Mohamed Barakat, Richard Christen, Wafa Achouak, Frédéric Barras et Thierry Heulin : The desert of tataouine: an extreme environment that hosts a wide diversity of microorganisms and radiotolerant bacteria. *Environmental Microbiology*, 8(3):514–525, 2006. 1

[CDR09] Michel Chilowicz, Étienne Duris et Gilles Roussel : Syntax tree fingerprinting for source code similarity detection. *In 17th IEEE International Conference on Program Comprehension (ICPC'09)*, pages 243–247, Vancouver, BC, Canada, mai 2009. IEEE Computer Society. 3

[CJ10] Ludovic Cottret et Fabien Jourdan : Graph methods for the investigation of metabolic networks in parasitology. *Parasitology*, 137(9):1393–1407, août 2010. 25

[CKL00] Xin Chen, Sam Kwong et Ming Li : A compression algorithm for DNA sequences and its applications in genome comparison. *In Proceedings of the fourth annual international conference on Computational molecular biology*, RECOMB '00, pages 107–, New York, NY, USA, 2000. ACM. 89, 93

[CLMT02] Xin Chen, Ming Li, Bin Ma et John Tromp : Dnacompress: fast and effective dna sequence compression. *Bioinformatics*, 18(12):1696–1698, 2002. 91

[CLRS04] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest et Clifford Stein : *Introduction à l'Algorithmique*. Sciences sup. Dunod, Paris, France, deuxième édition, 2004. 3, 19, 24

[CLZu02] Maxime Crochemore, Gad M. Landau et Michal Ziv-ukelson : A subquadratic sequence alignment algorithm for unrestricted cost matrices. *In Symposium of Discrete Algorithms (SODA)*, pages 679–688, 2002. 45

[CP07] Miguel Cantamutto et Mónica Poverene : Genetically modified sunflower release: Opportunities and risks. *Field Crops Research*, 101(2):133 – 144, 2007. 24

[CPJ+98] Francis S. Collins, Ari Patrinos, Elke Jordan, Aravinda Chakravarti, Raymond Gesteland, LeRoy Walters, the members of, the DOE et NIH planning groups : New Goals for the U.S. Human Genome Project: 1998-2003. *Science*, 282(5389):682–689, octobre 1998. 2

[CR93] Andrea Califano et Isidore Rigoutsos : FLASH: A Fast Look-Up Algorithm for String Homology. *In Proceedings of the 1st International Conference on Intelligent Systems for Molecular Biology*, pages 56–64. AAAI Press, 1993. 62

[CR96] Shenfeng Chen et John H. Reif : Efficient lossless compression of trees and graphs. *In IEEE Data Compression Conference (DCC)*, 1996. 94

[Cri58] Francis Harry Compton Crick : On protein synthesis. *Symposia of the Society for Experimental Biology*, 12:138–163, 1958. 23

[Cri66] Francis Harry Compton Crick : Codon-anticodon pairing: The wobble hypothesis. *Journal of Molecular Biology*, 19(2):548–555, 1966. 29

[Dah08] Ralf Dahm : Discovering DNA: Friedrich Miescher and the early years of nucleic acid research. *Human Genetics*, 122:565–581, 2008. 23

[Dar72] Charles Darwin : *The Origin of Species*. John Murray, 1872. 25

[DBT01] Laurent C. Duval et Van Bui-Tran : Compression denoising: using seismic compression for uncoherent noise removal. *In Proceedings of EAGE conference and technical exhibition*. European Association Geoscientists Engineering, 2001. 89

[DD06] Michel-Marie Deza et Elena Deza : *Dictionary of Distances*. Elsevier Science, Amsterdam, The Netherlands, octobre 2006. 41

[DDP09] Kévin Darty, Alain Denise et Yann Ponty : VARNA: Interactive drawing and editing of the RNA secondary structure. *Bioinformatics*, 25(15):1974–1975, 2009. viii, 30, 33

[DGT+08] Jennifer Daub, Paul P. Gardner, John Tate, Daniel Ramsköld, Magnus Manske, William G. Scott, Zasha Weinberg, Sam Griffiths-Jones et Alex Bateman : The RNA WikiProject: Community annotation of RNA families. *RNA*, 14(12):2462–2464, 2008. 5, 84

[DH00] C. Dolan et J. Humphrey : Governance and Trade in Fresh Vegetables: The Impact of UK Supermarkets on the African Horticulture Industry. *Journal of Development Studies*, 37(2):147–176, 2000. 24

[DJO+07] Neli Darie, V. Jâşcanu, M. Ognean, Cristina Busuioc, S. Lungu et Claudia Felicia Ognean : Studies on obtaining high nutritional value products from wheat germ. *Journal of Agroalimentary Processes and Technologies*, 13(2):439–442, 2007. 24

[DMRW09] Erik D. Demaine, Shay Mozes, Benjamin Rossman et Oren Weimann : An optimal decomposition algorithm for tree edit distance. *ACM Transactions on Algorithms*, 6(1):Article 2, 2009. 41, 47, 85

[dREC89] Philippe de Reffye, Éric Elguero et Evelyne Costes : Growth units construction in trees: a stochastic approach. *Acta Biotheoretica*, 39:325–342, 1989. 108

[dREF+88] Phillippe de Reffye, Claude Edelin, Jean Françon, Marc Jaeger et Claude Puech : Plant models faithful to botanical structure and development. *SIGGRAPH Comput. Graph.*, 22:151–158, June 1988. 24

[DST80] Peter J. Downey, Ravi Sethi et Robert Endre Tarjan : Variations on the common subexpression problem. *Journal of ACM*, 27(4):758–771, 1980. 94

[DT03] Serge Dulucq et Hélène Touzet : Analysis of tree edit distance algorithms. *In Proceedings of the 14th annual symposium on Combinatorial Pattern Matching (CPM*, pages 83–95. Springer-Verlag, 2003. 41

[EK72] Jack Edmonds et Richard M. Karp : Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19:248–264, April 1972. 48

[Eps08] Charles L. Epstein : *Introduction to the mathematics of medical imaging*. SIAM, 2 édition, 2008. 4

[Eul36] Leonhard Euler : Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, 8:128–140, 1736. vii, 7, 8

[Eva99] Patricia Anne Evans : *Algorithms and complexity for annotated sequence analysis*. Thèse de doctorat, University of Victoria, Victoria, B.C., Canada, 1999. 18, 32

[Fan39] Robert Mario Fano : The transmission of information. Rapport technique 65, Research Laboratory of Electronics, M.I.T., USA, 1939. 92

[FDM+10] Romain Fernandez, Pradeep Das, Vincent Mirabet, Éric Moscardi, Jan Traas, Jean-Luc Verdeil, Grégoire Malandain et Christophe Godin : Imaging plant growth in 4d: robust tissue reconstruction and lineaging at cell resolution. *Nature Methods*, 7:547–553, jul 2010. 2

[FG00] Pascal Ferraro et Christophe Godin : A distance measure between plant architectures. *Annals of Forest Science*, 57(5/6):445–461, 2000. viii, 27, 36, 37

[FGP05] Pascal Ferraro, Christophe Godin et Przemyslaw Prusinkiewicz : Toward a quantification of self-similarity in plants. *Fractals*, 13(2):1–25, 2005. 108, 109

[FGS90] Christine Froidevaux, Marie-Claude Gaudel et Michèle Soria : *Types de données et algorithmes*. Collection Informatique. McGraw-Hill, 1990. 3, 24

[FK50] P. Fatt et B. Katz : Some observations on biological noise. *Nature*, 166:597–598, octobre 1950. 3, 89

[FL76] D. Frijters et Aristid Lindenmayer : *Developmental descriptions of branching patterns with paracladial relationships*, pages 57–73. North-Holland, 1976. 108

[Fry93] John C. Fry : *Biological Data Analysis: A Practical Approach*. Oxford University Press, Inc., New York, NY, USA, 1st édition, 1993. 1

[FXM+98] Andrew Fire, Siqun Xu, Mary K. Montgomery, Steven A. Kostas, Samuel E. Driver et Craig C. Mello : Potent and specific genetic interference by double-stranded RNA in Caenorhabditis elegans. *Nature*, 391(6669):806–811, 1998. 23

[Gal07] Esra Galun : *Plant Patterning: Structural and Molecular Genetic Aspects*. World Scientific Publishing Company, août 2007. 34

[Gas07] Olivier Gascuel : *Mathematics of Evolution and Phylogeny*. Oxford University Press, Inc., New York, NY, USA, 2007. 4

[GC98] Christophe Godin et Yves Caraglio : A multiscale model of plant topological structures. *Journal of Theoretical Biology*, 191:1–46, 1998. viii, 3, 36, 39, 108

[GDT+08] Paul P. Gardner, Jennifer Daub, John G. Tate, Eric P. Nawrocki, Diana L. Kolbe, Stinus Lindgreen, Adam C. Wilkinson, Robert D. Finn, Sam Griffiths-Jones, Sean R. Eddy et Alex Bateman : Rfam: updates to the RNA families database. *Nucleic Acids Research*, 37(Database issue):D136–D140, 2008. 5, 29, 84

[GDT+10] Paul P. Gardner, Jennifer Daub, John Tate, Benjamin L. Moore, Isabelle H. Osuch, Sam Griffiths-Jones, Robert D. Finn, Eric P. Nawrocki, Diana L. Kolbe, Sean R. Eddy et Alex Bateman : Rfam: Wikipedia, clans and the "decimal" release. *Nucleic Acids Research*, 2010. 2, 5, 84

[GF10] Christophe Godin et Pascal Ferraro : Quantifying the degree of self-nestedness of trees. Application to the structural analysis of plants. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 7, 2010. ix, 4, 94, 95

[GFBG10] Anne-Laure Gaillard, Pascal Ferraro, Frédéric Boudon et Christophe Godin : Lossy compression of plant architectures. *In* D. Da Silva T. de Jong, éditeur : *6th International Workshop on Functional-Structural Plant Models*, pages 12–15, 2010. 90

[GGCC97] Christophe Godin, Yann Guédon, Evelyne Costes et Yves Caraglio : Measuring and analysing plants with the AMAPmod software. *In* M. Michalewicz, éditeur : *Plants to Ecosystems: Advances in Computational Life Sciences*, pages 53–84. CSRIO Publishing, Collingwood, Victoria, Australia, 1997. 38

[Gil86] Walter Gilbert : Origin of life: The RNA world. *Nature*, 319(618), 1986. 23

[GJBM+03] Sam Griffiths-Jones, Alex Bateman, Mhairi Marshall, Ajay Khanna et Sean R. Eddy : Rfam: an rna family database. *Nucleic Acids Research*, 31(1):439–441, 2003. 2, 5, 84

[GJMM+05] Sam Griffiths-Jones, Simon Moxon, Mhairi Marshall, Ajay Khanna, Sean R. Eddy et Alex Bateman : Rfam: annotating non-coding RNAs in complete genomes. *Nucleic Acids Research*, 33(Database issue):161–168, 2005. 5, 29, 84

[GK00] Mathieu Giraud et Grégory Kucherov : Maximal Repetitions and Application to DNA sequences. *In* M.-F. Sagot G. Caraux, O. Gascuel, éditeur : *Journées Ouvertes : Biologie, Informatique et Mathématiques - JOBIM'2000*, pages 165–172, Montpellier/France, 2000. AGRO. Colloque avec actes et comité de lecture. internationale. 89

[GMP96] Mikhail S. Gelfand, Andrey A. Mironov et Pavel A. Pevzner : Gene Recognition Via Spliced Sequence Alignment. *Proceedings of the National Academy of Sciences of the United States of America*, 93(17):9061–9066, 1996. 62, 63

[God00] Christophe Godin : Representing and encoding plant architecture: A review. *Annals of Forest Science*, 57(5-6):413–438, 2000. viii, 34, 36, 38

[Gor87] A. D. Gordon : A Review of Hierarchical Classification. *Journal of the Royal Statistical Society. Series A (General)*, 150(2):119–137, 1987. 99

[GPT00] K. Gadow, T. Pukkala et M. Tomé : *Sustainable forest management*. Managing forest ecosystems. Kluwer Academic Publishers, 2000. 24

[Gre89] N. Greene : Voxel space automata: modeling with stochastic growth processes in voxel space. *SIGGRAPH Comput. Graph.*, 23:175–184, July 1989. 37

[Gre96] Raymond Greenlaw : Subtree isomorphism is in DLOG for nested trees. *International Journal of Foundations of Computer Science*, 7(2):161–168, 1996. 94

[GT93] Stéphane Grumbach et Fariza Tahi : Compression of dna sequences. *In Proceedings of the IEEE Symposium on Data Compression*, pages 340–350, 1993. 93

[GT94] Stéphane Grumbach et Fariza Tahi : A new challenge for compression algorithms: Genetic sequences. *In Information Processing & Management*, pages 875–886, 1994. 93, 94

[Gus97] Dan Gusfield : *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, 1997. 59, 60, 78, 81, 82, 83

[GY03] Jonathan L. Gross et Jay Yellen : *Handbook of Graph Theory (Discrete Mathematics and Its Applications)*. CRC, 1 édition, décembre 2003. 9

[Hae79] Ernst Heinrich Philipp August Haeckel : *The Evolution of Man*. 1879. viii, 25

[Ham50] Rw Hamming : Error Detecting and Error Correcting Codes. *Bell System Technical Journal*, 26(2):147–160, 1950. 42

[Har92] John C. Hart : The object instancing paradigm for linear fractal modeling. *In Proceedings of the conference on Graphics interface '92*, pages 224–231, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc. 4

[Her07] Claire Herrbach : *Étude algorithmique et statistique de la comparaison de structures secondaires d'ARN*. Thèse de doctorat, Université de Bordeaux 1, 2007. 41, 53, 54

[HFS+94] I. L. Hofacker, W. Fontana, P. F. Stadler, L. S. Bonhoeffer, M. Tacker et P. Schuster : Fast folding and comparison of RNA secondary structures. *Monatshefte für Chemie / Chemical Monthly*, 125:167–188, 1994. 10.1007/BF00818163. 31

[HK04] Thomas Hernandez et Subbarao Kambhampati : Integration of biological sources: Current systems and challenges ahead. *Sigmod Record*, 33:51–60, 2004. 1

[HO70] Francis Hallé et Roelof Arend Albert Oldeman : *Essai sur l'architecture et la dynamique de croissance des arbres tropicaux*. Masson, 1970. 35

[Hog11] Paulien Hogeweg : The roots of bioinformatics in theoretical biology. *PLoS Computational Biology*, 7(3), 03 2011. 1

[HOT78] Francis Hallé, Roelof Arend Albert Oldeman et Philip Barry Tomlinson : *Tropical trees and forests. An architectural analysis*. Springer Verlag, New York, 1978. 34, 108

[HP91] Paul H. Harvey et Mark D. Pagel : *The comparative Method in Evolutionary Biology*. Oxford Series in Ecology and Evolution, Oxford England, 1991. 4

[HS65] Juris Hartmanis et Richard Edwin Stearns : On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965. 19

[HSS+58] Mahlon B. Hoagland, Mary Louise Stephenson, Jesse F. Scott, Liselotte I. Hecht et Paul C. Zamecnik : A soluble ribonucleic acid intermediate in protein synthesis. *Journal of Biological Chemistry*, 231(1):241–257, 1958. 23

[HSS96] Ivo L. Hofacker, Peter Schuster et Peter F. Stadler : Combinatorics of RNA secondary structures. *Discrete Applied Mathematics*, 89, 1996. 29

[HTGK03] Matthias Höchsmann, Thomas Töller, Robert Giegerich et Stefan Kurtz : Local similarity in RNA secondary structures. *In Proceedings of CSB 2003*, pages 159–168. IEEE, 2003. 31

[Huf52] David Huffman : A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the IRE*, 40(9):1098–1101, septembre 1952. 92, 93

[HWBB09] Steffen Heyne, Sebastian Will, Michael Beckstette et Rolf Backofen : Lightweight comparison of RNAs based on exact sequence-structure matches. *Bioinformatics*, 25(16):2095–2102, 2009. 66, 68, 69, 72, 80, 81, 84, 85

[Jac89] Guy Jacobson : Space-efficient static trees and graphs. *In 30th Annual Symposium on Foundations of Computer Science*, pages 549–554, octobre-novembre 1989. 109

[JD88] Anil K. Jain et Richard C. Dubes : *Algorithms for clustering data*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988. 99

[JLMZ02] Tao Jiang, Guohui Lin, Bon Ma et Kaizhong Zhang : A general edit distance between RNA structures. *Journal of Computational Biology*, 9(2):371–388, 2002. viii, 41, 53, 54, 85

[JMF99] A. K. Jain, M. N. Murty et P. J. Flynn : Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, septembre 1999. 99

[JMN+10] Stéphan Jacquet, Takeshi Miki, Rachel Noble, Peter Peduzzi et Steven Wilhelm : Viruses in aquatic ecosystems: important advancements of the last 20 years and prospects for the future in the field of microbial oceanography and limnology. *Advances in Oceanography and Limnology*, 1(1):97–141, 2010. 1

[JMT92] Deborah Joseph, Joao Meidanis et Prasoon Tiwari : Determining DNA sequence similarity using maximum independent set algorithms for interval graphs. *In Proceedings of SWAT 1992*, Lecture Notes in Computer Science, pages 326–337. Springer, 1992. 62, 64, 69, 78

[JRG08] Stefan Janssen, Jens Reeder et Robert Giegerich : Shape based indexing for faster search of RNA family databases. *BMC Bioinformatics*, 9, 2008. 66

[JWZ95] Tao Jiang, Lusheng Wang et Kaizhong Zhang : Alignment of trees - An alternative to tree edit. *In* Maxime Crochemore et Dan Gusfield, éditeurs : *Combinatorial Pattern Matching*, volume 807 de *Lecture Notes in Computer Science*, pages 75–86. Springer Berlin / Heidelberg, 1995. 50

[Kal99] Jaan Kalda : On the fractality of the biological tree-like structures. *Discrete Dynamics in Nature and Society*, 3(4):297–306, 1999. 3

[Ken02] W. James Kent : BLAT–the BLAST-like alignment tool. *Genome Research*, 12:656–664, 2002. 61

[Kle98] Philip N. Klein : Computing the edit-distance between unrooted ordered trees. *In Proceedings of the 6th annual European Symposium on Algorithms (ESA*, pages 91–102. Springer-Verlag, 1998. 41

[KLMT02] Uri Keich, Ming Li, Bin Ma et John Tromp : On spaced seeds for similarity search. *Discrete Appl. Math*, 138:253–263, 2002. 62

[KNR05] Gregory Kucherov, Laurent Noé et Mihkail Roytberg : A unifying framework for seed sensitivity and its application to subset seeds. *In Proceedings of the International Moscow Conference on Computational Molecular Biology (MCCMB), Moscow (Russia)*, pages 195–196, juillet 2005. 62

[KY07] Lev Klebanov et Andrei Yakovlev : How high is the level of technical noise in microarray data. *Biology Direct*, page 9, 2007. 3, 89

[Lan04] J. Kevin Lanctot : *Some String Problems in Computational Biology*. Thèse de doctorat, University of Waterloo, 2004. 93

[LC03] Zoé Lacroix et Terence Critchlow : *Bioinformatics: Managing Scientific Data*. Morgan Kaufmann, 2003. 1

[LCLZ10] Xiao Li, Xuehong Cao, Chen Li et Yue Zhang : A method for doppler weather radar raw data compression based on wavelet transform modulus maxima denoising. *In 12th IEEE International Conference on Communication Technology (ICCT)*, pages 424–427, 2010. 89

[LDB11] Antoine Lambert, Jonathan Dubois et Romain Bourqui : Pathway preserving representation of metabolic networks. *In Computer Graphics Forum special issue on 13th Eurographics/IEEE-VGTC Symposium on Visualization*, 2011. vii, 7, 8

[Lev66] Vladimir I. Levenshtein : Binary codes capable of correcting deletions, insertions, and reversals. Rapport technique 8, 1966. 41, 43

[LL52] Kaj Ulrik Linderstrøm-Lang : *Lane Medical Lecture. Proteins and Enzymes*. Stanford University, 1952. 29

[LP85] David J. Lipman et William R. Pearson : Rapid and sensitive protein similarity searches. *Science*, 227(4693):1435–1441, 1985. 5, 59, 60, 61, 62, 82

[LPD+08] Magali Leroy, Magali Prigent, Murielle Dutertre, Fabrice Confalonieri et Michael Dubow : Bacteriophage morphotype and genome diversity in seine river sediment. *Freshwater Biology*, 53(6):1176–1185, 2008. 1

[LPR+08] Antonio Lozano, Ron Y. Pinter, Oleg Rokhlenko, Gabriel Valiente et Michal Ziv-Ukelson : Seeded tree alignment. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 5(4):503–513, 2008. 66, 84

[LPV03] Laszlo Lovasz, Jozsef Pelikan et Katalin L. Vesztergombi : *Discrete Mathematics*. Springer, 2003. 109

[Man82] Benoît B. Mandelbrot : *The Fractal Geometry of Nature*. W. H. Freeman and Company, first édition, 1982. 4, 89

[Mar10] Diana Marco : *Metagenomics: Theory, Methods and Applications*. Caister Academic Press, 2010. 1

[MBG+10] Alberto Magi, Matteo Benelli, Alessia Gozzini, Francesca Girolami, Francesca Torricelli et Maria Luisa Brandi : Bioinformatics for next generation sequencing data. *Genes*, 1(2):294–307, 2010. 4

[MCM+09] Sébastien Mondet, Wei Cheng, Géraldine Morin, Romulus Grigoras, Frédéric Boudon et Wei Tsang Ooi : Compact and progressive plant models for streaming in networked virtual environments. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 5(3):1–22, août 2009. 108, 115

[Mir96] Boris Mirkin : *Mathematical classification and clustering*. Kluwer Academic Press, 1996. 99

[Moo65] Gordon E. Moore : Cramming more components onto integrated circuits. *Electronics*, 38(8), April 1965. 3

[Mor96] Burkhard Morgenstern : Multiple DNA and protein sequence alignment based on segment-to-segment comparison. *Proc. Natl. Acad. Sci. USA*, 93:12098–12103, 1996. 62

[Mou04] David W. Mount : *Bioinformatics: Sequence and Genome Analysis, Second Edition*. Cold Spring Harbor Laboratory Press, 2 édition, Juillet 2004. 4, 26

[MS99] Christopher D. Manning et Hinrich Schütze : *Foundations of statistical natural language processing*. MIT Press, Cambridge, MA, USA, 1999. 99

[MSI00] Toshiko Matsumoto, Kunihiko Sadakane et Hiroshi Imai : Biological sequence compression algorithms. *Genome Informatics*, 11:43–52, 2000. 91, 94

[MTL02] Bin Ma, John Tromp et Ming Li : Pattern hunter: faster and more sensitive homology search. *Bioinformatics*, 18:440–445, 2002. 62

[Mur83] Fionn Murtagh : A survey of recent advances in hierarchical clustering algorithms. *The Computer Journal*, 26(4):354–359, 1983. 100

[MV07] Oliver Mason et Mark Verwoerd : Graph theory and networks in biology. *In IET Systems Biology*, pages 89–119, 2007. 7

[Nat93] Balas K. Natarajan : Filtering random noise via data compression. *In Data Compression Conference*, pages 60–69, 1993. 89

[NBCS97] Martin A. Nowak, Maarten C. Boerlijst, Jonathan Cooke et John Maynard Smith : Evolution of genetic redundancy. *Nature*, 388(6638), Juillet 1997. 3

[NK05] Laurent Noé et Gregory Kucherov : YASS: enhancing the sensitivity of DNA similarity search. *Nucleic Acids Research*, 33 (web-server issue):W540–W543, juillet 2005. 61, 62

[NW70] Saul B. Needleman et Christian D. Wunsch : A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, mars 1970. 41, 44, 45

[OA05] Enno Ohlebusch et Mohamed I. Abouelhoda : *Handbook of Computational Molecular Biology*, chapitre Chaining Algorithms and Applications in Comparative Genomics. CRC Press, 2005. 61, 64, 69, 78, 81, 82

[OF09] Aïda Ouangraoua et Pascal Ferraro : A constrained edit distance algorithm between semi-ordered trees. *Theoretical Computer Science*, 410:837–846, mars 2009. 41, 50

[OGG+10] Seán I. O'Donoghue, Anne-Claude C. Gavin, Nils Gehlenborg, David S. Goodsell, Jean-Karim K. Hériché, Cydney B. Nielsen, Chris North, Arthur J. Olson, James B. Procter, David W. Shattuck, Thomas Walter et Bang Wong : Visualizing biological data-now and in the future. *Nature methods*, 7(3 Suppl):S2–S4, mars 2010. 89

[Oua07] Aïda Ouangraoua : *Développement d'outils conceptuels et algorithmiques pour l'analyse de structures biologiques arborescentes*. Thèse de doctorat, Université de Bordeaux 1, 2007. viii, 16, 27, 37, 39

[Pal00] Bernhard Palsson : The challenges of in silico biology. *Nature Biotechnology*, 18(11):1147–1150, novembre 2000. 3

[PBF+10] Chakkrit Preuksakarn, Frédéric Boudon, Pascal Ferraro, Jean-Baptiste Durand, Eero Nikinmaa et Christophe Godin : Reconstructing plant architecture from 3d laser scanner data. *In* D. Da Silva T. de Jong, éditeur : *6th International Workshop on Functional-Structural Plant Models*, pages 16–18, 2010. 2

[PBS+06] Jakob Skou Pedersen, Gill Bejerano, Adam Siepel, Kate Rosenbloom, Kerstin Lindblad-Toh, Eric S. Lander, Jim Kent, Wabb Miller et David Haussler : Identification and classification of conserved RNA secondary structures in the human genome. *PLoS Computational Biology*, 2(4):e33, 2006. 59

[PDKB+08] Christophe Pradal, Samuel Dufour-Kowalski, Frédéric Boudon, Christian Fournier et Christophe Godin : Openalea: A visual programming and component-based software platform for plant modeling. *Functional Plant Biology*, 35(9 & 10):751–760, 2008. 111

[Pet06] Pierre Peterlongo : *Filtrage de séquences d'ADN pour la recherche de longues répétitions multiples*. Thèse de doctorat, Université de Marne-la-Vallée, 2006. 43

[PGP10] Wojciech Palubicki, Christophe Godin et Przemyslaw Prusinkiewicz : Towards a quantitative assessment of architectures for fspm. *In 6th International Workshop on Functional-Structural Plant Models*, pages 7–9, 2010. 111

[PH89] Przemyslaw Prusinkiewicz et James Hanan : *Lindenmayer systems, fractals and plants*. Springer, 1989. 108

[PJ88] William R. Pearson et Lipman David J. : Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences of the United States of America*, 85(8):2444–2448, 1988. 5, 59, 60, 61, 62

[PL90] Przemyslaw Prusinkiewicz et Aristid Lindenmayer : *The Algorithmic Beauty of Plants*. Springer, second édition, 1990. 4, 37, 108

[Ple89] Daniel Pletinckx : Quaternion calculus as a basic tool in computer graphics. *The Visual Computer*, 5:2–13, 1989. 10.1007/BF01901476. 111

[PMKL01] Przemyslaw Prusinkiewicz, Lars Mündermann, Radoslaw Karwowski et Brendan Lane : The use of positional information in the modeling of plants. *In Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 289–300, New York, NY, USA, 2001. ACM. 114

[PRB85] Cornelis W. Pleij, Krijn Rietveld et Leendert Bosch : A new principle of RNA folding based on pseudoknotting. *Nucleic Acids Research*, 13(5):1717–1731, 1985. 31

[Pru04] Przemyslaw Prusinkiewicz : *Thinking in Patterns: Fractals and Related Phenomena in Nature*, chapitre Self-similarity in plants: Integrating mathematical and biological perspectives, pages 103–118. Novak, 2004. 109

[PSN+96] Jari Perttunen, Risto Sievänen, Eero Nikinmaa, Hannu Salimen, Hannu Saarenmaa et Julli Väkevä : LIGNUM: A Tree Model Based on Simple Structural Units. *Annals of Botany*, 77(1):87–98, 1996. 35

[PY88] Christos Papadimitriou et Mihalis Yannakakis : Optimization, approximation, and complexity classes. *In Proceedings of the twentieth annual ACM symposium on Theory of computing*, STOC '88, pages 229–234, New York, NY, USA, 1988. ACM. 47

[RDD+97] É Rivals, Olivier Delgrange, Jean-Paul Delahaye, Max Dauchet, M.-O. Delorme, A. Hénaut et E. Ollivier : Detection of significant patterns by compression algorithms: the case of approximate tandem repeats in dna sequences. *Computer applications in the biosciences : CABIOS*, 13(2):131–136, 1997. 94

[RDDD96] Éric Rivals, Max Dauchet, Jean-Paul Delahaye et Olivier Delgrange : Compression and genetic sequence analysis. *Biochimie*, 78(5):315 – 322, 1996. 89, 94

[Rea84] Darryl Reanney : Molecular biology: Genetic noise in evolution? *Nature*, 307: 318–319, janvier 1984. 3, 89

[RH94] Peter M. Room et Jim S. Hanan : Virtual cotton: a new tool for research, management and training. *In Proceedings of the world cotton research conference*, pages 14–17, 1994. 4

[RIMS03] David Rizo, José Manuel Iñesta et Francisco Moreno-Seco : Tree-structured representation of musical information. *In Pattern Recognition and Image Analysis*, volume 2652 de *Lecture Notes in Computer Science*, pages 838–846. Springer Berlin / Heidelberg, 2003. 3

[Rob96] David F. Robinson : A symbolic framework for the description of tree architecture models. *Botanical Journal of the Linnean Society*, 121(3):243–261, 1996. 34

[Ros81] Juan Ross : *The radiation regim and the architecture of plant stands*. Dr. W. Junk publishers, La Hague, Pays-Bas, 1981. 34

[RP97] William R. Remphrey et Przemyslaw Prusinkiewicz : Quantification and modelling of tree architecture. *In* Marek T. Michalewicz, éditeur : *Plants to Ecosystems*, volume 1 de *Advances in Computational Life Sciences*, chapitre 3, pages 45–52. CSIRO Publishing, P.O. Box 1139, Collingwood 3066, Australia, février 1997. 38

[RQ04] François Rechenmann et Isabelle Quinkal : Entre biologie, informatique et mathématiques : la bioinformatique. *Interstices*, mars 2004. http://www.interstices.info/display.jsp?id=c_6607. 1

[RUC+10] Jane B. Reece, Lisa A. Urry, Michael L. Cain, Steven A. Wasserman, Peter V. Minorsky et Robert B. Jackson : *Campbell Biology*. Benjamin Cumming, 9 édition, 2010. 2, 3, 23, 33

[SAB+77] Frederick Sanger, G. M. Air, B. G. Barrell, N. L. Brown, Alan R. Coulson, C. A. Fiddes, C. A. Hutchison, P. M. Slocombe et M. Smith : Nucleotide sequence of bacteriophage phi X174 DNA. *Nature*, 265(5596):687–695, février 1977. 2

[Sad96] Ilan Sadeh : Universal data compression algorithm based on approximate string matching. *Probability in the Engineering and Informational Sciences*, 10:465–486, 1996. 93

[Sal07] David Salomon : *Data Compression: The Complete Reference*. Springer-Verlag, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 2007. With contributions by Giovanni Motta and David Bryant. 91, 92, 93, 109

[Say00] Khalid Sayood : *Introduction to data compression (2nd ed.)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000. 90, 91

[SB92] H Sinoquet et R Bonhomme : Modeling radiative transfer in mixed and row intercropping systems. *Agricultural and Forest Meteorology*, 62(3-4):219–240, 1992. 37

[Sel77] S. Selkow : The tree-to-tree editing problem. *Information Processing Letters*, 6(6):184–186, décembre 1977. 41

[Sha48] Claude Elwood Shannon : A mathematical theory of communication. *Bell system technical journal*, 27, 1948. 92

[Sha88] Bruce A. Shapiro : An algorithm for comparing multiple RNA secondary structures. *Computer Applications in the Biosciences*, 4(3):381–393, 1988. viii, 23, 30, 31

[SK95] Thomas Schreiber et Holger Kantz : Noise in chaotic data: Diagnosis and treatment. *CHAOS*, 5:133–142, 1995. 3, 89

[SM02] Richard Sherlock et John D. Morrey : *Ethical issues in biotechnology*. G - Reference, Information and Interdisciplinary Subjects Series. Rowman & Littlefield, 2002. 24

[Smi84] Alvy Ray Smith : Plants, fractals and formal languages. *Computer Graphics*, 18(3):1–10, 1984. 108

[SR95] Grant R. Sutherland et Robert I. Richards : Simple tandem dna repeats and human genetic disease. *Proceedings of the National Academy of Sciences*, 92(9):3636–3641, 1995. 89

[SRG97] Hervé Sinoquet, Pierre Rivet et Christophe Godin : Assessment of the three-dimensional architecture of walnut trees using digitising. *Silva Fennica*, 31(3):265–273, 1997. x, 112

[SS00] Cyril Soler et François Sillion : Hierarchical instantiation for radiosity. *In Eurographics Workshop on Rendering Techniques '00, June, 2000*, pages 173–184, Brno, Tchéquie, juin 2000. Springer. 4

[SSdR03] Cyril Soler, François X. Sillion et Philippe de Reffye : An efficient instantiation algorithm for simulating radiant energy transfer in plant models. *ACM Transactions on Graphics*, 22(2):204–233, 2003. 4, 94

[Ste03] Bertran Steinsky : Efficient coding of labeled directed acyclic graphs. *Software Computing*, 7:350–356, 2003. x, 109, 111

[STMK98] Hervé Sinoquet, Sornprach Thanisawanyangkura, Hatem Mabrouk et Poonpipope Kasemsap : Characterization of the light environment in canopies using 3d digitising and image processing. *Annals of Botany*, 82(2):203–212, 1998. 35

[Sut63] Ivan Edward Sutherland : Sketchpad - A man-machine graphical communication system. *In Proceedings of the Spring Joint Computer Conference*, 1963. 94

[Sut64] Ivan E. Sutherland : Sketch pad a man-machine graphical communication system. *In Proceedings of the SHARE design automation workshop*, DAC '64, pages 6.329–6.346, New York, NY, USA, 1964. ACM. 4, 94

[SVR+06] Peter Steffen, Björn Voss, Marc Rehmsmeier, Jens Reeder et Robert Giegerich : RNAshapes: an integrated RNA analysis package based on abstract shapes. *Bioinformatics*, 22(4), 2006. 66

[SW81] Temple F. Smith et Michael S. Waterman : Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195 – 197, 1981. 45

[SYHK64] K. Shinozaki, K. Yoda, K. Hozumi et T. Kira : A Quantitative Analysis of Plant Form – The Pipe Model Theory I. Basic Analyses. *Japanese Journal of Ecology*, 14(3):97–105, juin 1964. 35

[Syl78] James Joseph Sylvester : Chemistry and algebra. *Nature*, 17(432):284, 1878. 7

[SZ90] Bruce A. Shapiro et Kaizhong Zhang : Comparing multiple RNA secondary structures using tree comparisons. *Computer applications in the Biosciences*, 6(4): 309–318, 1990. 31, 38

[Tai79] Kuo-Chung Tai : The tree-to-tree correction problem. *Jounal of the ACM*, 26(3): 422–433, 1979. 41, 46, 47, 53

[Tar83] Robert Endre Tarjan : *Data structures and network algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1983. 48

[TBS93] Stephen N. Thibodeau, Gary Bren et Daniel Schaid : Microsatellite instability in cancer of the proximal colon. *Science*, 260(5109):816–819, 1993. 89

[Tuo02] Ilkka Tuomi : The Lives and Death of Moore's Law. *First Monday*, 7(11), Novembre 2002. 3

[Tur37] Alan Mathison Turing : On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1): 230–265, Janvier 1937. 19

[Wal92] Brian H. Walker : Biodiversity and ecological redundancy. *Conservation Biology*, 6(1):18–23, 1992. 3

[War63] Joe Ward : Hierarchical Grouping to Optimize an Objective Function. *Journal of the American Statistical Association*, 58(301):236–244, mars 1963. 105

[WBM+94] Ian H. Witten, Timothy C. Bell, Alistair Moffat, Craig G. Nevill-Manning, Tony C. Smith et Harold Thimbleby : Semantic and generative models for lossy text compression. *The Computer Journal, Volume 37, Issue*, 2, 1994. 93

[WC53] James D. Watson et Francis Harry Compton Crick : A structure for deoxyribose nucleic acid. *Nature*, 171:737–738, 1953. 29

[Wel84] Terry A. Welch : A Technique for High-Performance Data Compression. *Computer*, 17(6):8–19, juin 1984. 93

[Whi79] James White : The plant as a metapopulation. *Annual Review of Ecology and Systematics*, 10:109–145, 1979. 4, 34

[Wil71] Robin Williams : A survey of data structures for computer graphics systems. *ACM Comput. Surv.*, 3:1–21, March 1971. 89

[XGC07] Hui Xu, Nathan Gossett et Baoquan Chen : Knowledge and heuristic-based modeling of laser-scanned trees. *ACM Transactions on Graphics*, 26, octobre 2007. 2

[Zha96] Kaizhong Zhang : A constrained edit distance between unordered labeled trees. *Algorithmica*, 15(3):205–222, 1996. 41, 48, 95, 98, 104, 106, 111

[ZL77] Jacob Ziv et Abraham Lempel : A universal algorithm for sequential data compression. *IEEE TRANSACTIONS ON INFORMATION THEORY*, 23(3):337–343, 1977. 93

[ZL78] Jacob Ziv et Abraham Lempel : Compression of Individual Sequences via Variable-Rate Coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978. 45, 93

[ZS84] Michael Zuker et David Sankoff : RNA secondary structures and their prediction. *Bulletin of Mathematical Biology*, 46(4):591–621, juillet 1984. viii, 3, 23, 27, 30, 31

[ZS89] Kaizhong Zhang et Shasha Shasha : Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal of Computing*, 18(6): 1245–1262, 1989. 16, 41, 47, 85

[ZSS92] Kaizhong Zhang, Rick Statman et Dennis Shasha : On the editing distance between unordered labeled trees. *Information Processing Letters*, 42:133–139, mai 1992. 16, 41, 47