Alignement Multiple de Données Génomiques et Post-Génomiques : Approches Algorithmiques

Yves-Pol Denielou

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Informatique**

Arrêté ministérial : 7 Août 2006

Présentée par

Yves-Pol Deniélou

Thèse dirigée par **Marie-France Sagot**
et codirigée par **Alain Viari**

préparée au sein **de l'INRIA Rhône-Alpes**
et de **l'École Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique**

Alignement Multiple de Données Génomiques et Post-Génomiques : Approches Algorithmiques

Thèse soutenue publiquement le 5 **Novembre** 2010,
devant le jury composé de :

Mme Marie-Christine Rousset
Professeur de l'Université de Grenoble, Présidente
Mme Hélène Touzet
Directeur de Recherche CNRS au LIFL, Rapporteur
M. Thomas Schiex
Chercheur à l'INRA Toulouse, Rapporteur
M. David Vallenet
Chercheur CEA au Génoscope, Examinateur
Mme Marie-France Sagot
Directeur de Recherche au LBBE, Directeur de thèse
M. Alain Viari
Directeur de Recherche à l'INRIA Rhône-Alpes, Co-Directeur de thèse
M. Laurent Trilling
Professeur de l'Université de Grenoble, Invité

Résumé L'alignement multiple de réseaux biologiques a pour objectif d'extraire des informations fonctionnelles des données haut-débit représentées sous forme de graphes. Ceci concerne, par exemple, les données d'interaction protéines-protéines, les données métaboliques ou même les données génomiques. Dans un premier temps nous proposons un formalisme précis, qui s'appuie sur les notions de graphe de données stratifié et de multigraphe d'alignement (MGA), et qui définit les alignements multiples locaux en autorisant notamment un réglage de la conservation de la topologie entre les réseaux. Nous présentons ensuite un algorithme de construction et partitionnement "à la volée" du MGA, qui permet de traiter de façon efficace l'alignement de nombreux réseaux biologiques. Dans un second temps, nous étendons le formalisme pour parvenir à retrouver des alignements - que nous qualifions de "partiels" - lorsqu'il y a des nœuds manquants sur certains réseaux. Nous détaillons les algorithmes associés, puis nous proposons différentes améliorations, et des variantes adaptées à des problèmes biologiques particuliers.

Abstract Multiple alignment of biological networks is used to extract functional information from high-throughput data represented by graphs. This data can be protein-protein interactions, metabolic pathways or even the gene layout on a chromosome. We start by giving a precise formalism, based on the notions of layered datagraph and alignment multigraph (MGA), which defines local multiple alignments in the datagraph, allowing for example the tuning of the topology conservation between networks. Next, we present a new algorithm that builds and partitions the MGA "on the fly", which allows us to deal with alignment of numerous biological networks. In a second part, we extend the formalism to be able to recover alignments - which we call "partial" - when there are missing nodes on some networks. We explain the algorithms we have designed to compute those alignments, and then we give some improvements and some variants tailored to deal with specific biological problems.

Mots-Clés Alignement Multiple, Synténies

à Geneviève Deniélou, ma grand-mère

Remerciements

Merci à Alain Viari et Marie-France Sagot pour tout !

Merci à tous mes cobureaux successifs (j'en ai épuisés !) : Pedro, Sophie (et Olivier et Maéline !) et Valentina (et Cyril et Lara !), puis Sara, Eugenio, Anne (et UniPathway !) et Eric !

Merci aux Génostariens et Génostariennes, Bruno, Estelle, Pierre, Agnès, François etc ! Aux Ibissiens et Ibissiennes Hidde, Delphine, Arun, Diana, etc !

Merci à Frédéric Boyer, Vincent Navratil, Eric Tannier, Claire Lemaître, Amélie Véron pour les discussions algorithmiques ou biologiques !

Un grand merci à Françoise pour tout !

Merci à Samir, Wahid, Maxime, Maxence, Bob et Cédric !

Et enfin merci à tous ceux que j'ai oubliés !

Table des matières

Chapitre 1

Introduction

Depuis une dizaine d'années, le paysage de la recherche en bioinformatique a changé, passant de l'analyse des séquences génomiques à l'analyse de données diverses produites en masse par les techniques dites à haut débit. Une grande partie de ces données peut être représentée par des graphes [AA03]. Parmi les exemples les plus courants, on peut citer les données d'interaction entre protéines, qui peuvent être représentées sous la forme d'un graphe dont les nœuds sont des protéines et les arêtes représentent les interactions physiques qui les relient, ou encore les données métaboliques, qui peuvent être représentées sous la forme d'un graphe dont les nœuds sont les réactions et les arêtes relient deux réactions si le substrat de l'une est produit de l'autre. Enfin la disposition des gènes sur un chromosome peut elle-même se voir représentée sous la forme d'un graphe dont les nœuds sont les gènes et deux gènes sont reliés par une arête s'ils sont adjacents sur le chromosome.

Un des enjeux majeurs en biologie moléculaire est la compréhension de ces réseaux biologiques qui participent aux différentes fonctions au sein de la cellule. L'étude de ces réseaux offre la possibilité de comprendre le comportement de ces systèmes dont les entités coopèrent avec une précision remarquable sous des contraintes biologiques très fortes. Le but est de découvrir les principes sous-jacents qui gouvernent leur fonctionnement.

De la même façon que l'alignement multiple de séquences est essentiel pour l'analyse des séquences génomiques, l'alignement multiple de réseaux biologiques s'affirme comme un outil majeur dans l'élucidation des fonctions biologiques de ces réseaux.

En pratique, toute une classe de problèmes peut être formalisée comme une recherche locale d'organisations communes à un ensemble de réseaux biologiques.

Les résultats peuvent en effet être interprétés comme des synténies dans le cas de génomes, des complexes protéiques dans le cas de réseaux d'interaction protéine-protéine ou encore des voies métaboliques dans le cas de réseaux métaboliques.

Nous avons choisi de nous pencher sur le problème spécifique de l'alignement local multiple de réseaux biologiques, en cherchant à conserver une approche exacte et générique.

Cette thèse se base sur les travaux initiés par Frédéric Boyer durant sa thèse dans l'équipe [BML+05]. L'objectif du présent travail est, dans un premier temps, de rendre l'algorithme (C_3Part) plus efficace en pratique (notamment dans le cas d'un grand nombre de réseaux), puis, dans un second temps, de lever une partie des contraintes du formalisme introduit dans [BML+05], notamment pour pouvoir traiter le cas de l'alignement partiel de réseaux biologiques.

Ce manuscrit est organisé en cinq chapitres. Le premier détaille le contexte bibliographique pour chacun des types de données : données d'interaction entre protéines, données métaboliques et données génomiques.

Dans le second chapitre, nous commençons par formaliser le problème général de l'alignement local multiple de réseaux biologiques. Nous détaillons une première représentation des données sous la forme d'un graphe de données stratifié et le problème de recherche d'alignement local associé. Puis, nous donnons une seconde représentation des données sous la forme d'un multigraphe d'alignement, qui correspond à une représentation fusionnée des réseaux biologiques. Nous vérifions ensuite que les alignements locaux du graphe de données stratifié correspondent exactement à certains sous-graphes du multigraphe d'alignement que nous appellerons "*connectons*" .

Le troisième chapitre détaille l'algorithme que nous avons conçu pour calculer les connectons du multigraphe d'alignement sans avoir à calculer celui-ci dans son intégralité. Cela nous permet de traiter l'alignement multiple de nombreux réseaux tout en conservant une approche exacte.

Dans le chapitre quatre, nous donnons une définition moins contrainte des alignements locaux, en autorisant un certain nombre d'erreurs ou de nœuds manquants, et nous examinons les modifications que cela induit dans notre algorithme.

Enfin dans un dernier chapitre, nous présentons plusieurs améliorations et variantes possibles, ainsi qu'une application à l'analyse des perturbations induites par des virus sur l'interactome humain menée en collaboration avec V. Navratil.

Chapitre 2

Etat de l'art

2.1 Cadre général

Le problème de l'alignement de réseaux ressemble d'une certaine façon au problème classique de l'alignement de séquences. Ainsi les approches d'alignement de réseaux peuvent être locales ou globales, elles peuvent s'appliquer à deux réseaux seulement ou à plus de deux réseaux simultanément. De plus, de la même manière qu'il faut définir une mesure de similarité entre les symboles constituant les séquences, il faut aussi définir une mesure de similarité entre les nœuds de différents réseaux.

Cependant, les alignements de réseaux conservent certaines spécificités, concernant notamment la prise en compte de la topologie, qui n'existent généralement pas pour les séquences.

La littérature dans le domaine de l'alignement de réseaux biologiques a connu un très fort accroissement dans les cinq dernières années et les définitions proposées sont très disparates.

Afin d'essayer de structurer au mieux ce chapitre et de clarifier les questions posées, nous allons tenter, au préalable, de dégager quelques concepts communs à toutes ces approches et introduire quelques notations générales.

Tout d'abord, nous appelerons **réseaux primaires** les réseaux biologiques que l'on cherche à aligner. Ces réseaux primaires seront notés $G_1, G_2, \ldots G_n$, avec $G_i = (V_i, E_i)$, où V_i est l'ensemble des nœuds et E_i l'ensemble des arêtes [1].

1. Notons que nous ne faisons aucune hypothèse à ce stade sur la nature des V_i. Les nœuds peuvent représenter des gènes, des protéines ou des réactions chimiques.

Comme indiqué précédemment, afin de décider quels nœuds sont alignés, il est fréquemment fait usage, directement ou indirectement, d'un indice de dissimilarité entre les nœuds de ces réseaux primaires. Nous noterons $ds(v_i, v_j); v_i \in V_i, v_j \in V_j$ un tel indice [2].

Souvent, au lieu de travailler avec un indice de dissimilarité, certains auteurs choisissent de considérer une relation de correspondance (ou similarité) entre les nœuds des réseaux. Nous noterons R cette relation [3]. Il faut noter que R est souvent obtenue par seuillage de ds, c'est-à-dire que $v_i R v_j \Leftrightarrow ds(v_i, v_j) < \sigma$. On parlera dans ce cas de "relation seuil" : R_{seuil}. Lorsque les nœuds de V_i sont associés à des séquences nucléiques ou protéiques, il est fréquent d'utiliser pour ds un indice basé sur la similarité de séquence soit exacte (distance d'édition), soit approchée (P-value BLAST [AGM+90]).

Enfin, un concept important, sur lequel nous reviendrons largement dans les chapitres suivants, est celui d'une représentation fusionnée des réseaux primaires que nous appellerons **multigraphe d'alignement (MGA)** [4].

Nous donnons ici une définition informelle d'un MGA, mais nous reviendrons plus tard sur une définition plus précise.

Etant donnés n réseaux primaires $G_i = (V_i, E_i)$ et une correspondance R entre les nœuds de ces réseaux, l'idée est de sélectionner des n-uplets de nœuds, un provenant de chaque V_i - nous appellerons ces n-uplets des *empilements* - et de les relier avec des arêtes en se basant sur les ensembles E_i.

La figure 2.1 donne un exemple de multigraphe d'alignement pour trois réseaux biologiques, dans le cas où les empilements sont des cliques de la relation de correspondance R.

Comme nous le verrons plus tard, la construction explicite du MGA n'est pas toujours nécessaire (ni même désirable), mais elle est très utile pour formaliser et comprendre les différences entre les approches.

En nous basant sur cette représentation, nous allons distinguer, pour chaque méthode, quatre choix différents.

---

2. ds vérifie $\forall x, ds(x, x) = 0$ et $\forall x, y, ds(x, y) = ds(y, x)$. Pour que l'indice de dissimilarité soit une distance, il doit vérifier de plus l'inégalité triangulaire : $\forall x, y, z, ds(x, y) \leq ds(x, z) + ds(y, z)$.

3. Notons que contrairement au cas de l'indice de dissimilarité qui fait l'hypothèse que les v_i et v_j sont de même nature, cette relation autorise la mise en correspondance de nœuds de natures différentes ; par exemple des gènes (V_1) avec des réactions chimiques (V_2).

4. Le terme "multigraphe" désigne un graphe pourvu de plusieurs ensembles d'arêtes. Les graphes classiques sont inclus dans l'ensemble des multigraphes.

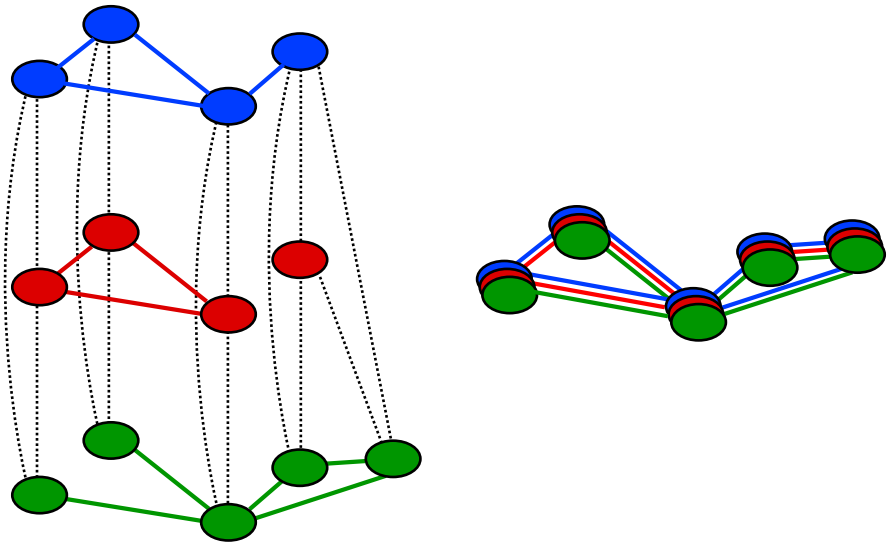


FIGURE 2.1 – Trois réseaux biologiques (en bleu, rouge et vert) sur la gauche, avec une relation de correspondance R représentée par des arêtes pointillées. Sur la droite, un exemple de multigraphe d'alignement : ce graphe comporte 5 empilements, reliés par des arêtes tirées des réseaux primaires. Intuitivement deux empilements sont reliés par une arête bleue s'ils correspondent dans le réseau biologique bleu au même nœud ou à des nœuds connectés.

– Choix 1 : comment définir un indice de dissimilarité ds ou une relation de correspondance R,
– Choix 2 : comment définir les empilements (i.e. comment aggréger les nœuds des différents V_i en se basant sur ds ou R),
– Choix 3 : comment relier ces empilements dans le multigraphe d'alignement,
– Choix 4 : comment sélectionner parmi les sous-graphes du multigraphe d'alignement ceux qui constituent les solutions du problème posé.

La définition de ces choix n'est pas pertinente dans toutes les approches. En particulier, dans les approches utilisant un indice de dissimilarité, on n'opère pas de choix explicite des empilements (Choix 2) mais on considère implicitement tous les empilements possibles. Cela revient à considérer que les nœuds du multigraphe d'alignement sont les nœuds du produit cartésien $V_1 \times V_2 \times \ldots \times V_n$ complet.

Nous allons maintenant explorer les différentes approches d'alignement de réseaux biologiques de la littérature, en les classant en fonction du type de données biologiques traitées et en essayant de les replacer dans ce cadre général.

Nous traiterons successivement des réseaux d'interaction protéine-protéine puis des réseaux métaboliques pour terminer par les génomes qui comme nous le verrons peuvent également être considérés comme des réseaux particuliers.

Notations :

réseaux ‘‘primaires’’ : $G_1, G_2, \ldots G_n$; $G_i = (V_i, E_i)$; $v_i \in V_i$

indice de dissimilarité entre éléments de V_i et V_j : $ds(v_i, v_j)$

relation de correspondance R entre éléments de V_i et V_j : $v_i R v_j$

multigraphe d'alignement : $G = (\mathcal{V}, \mathcal{E})$

2.2 Données d'interactions protéine-protéine

Les réseaux d'interaction protéine-protéine (réseaux PPI) sont classiquement représentés sous la forme de graphes dont les nœuds sont des protéines et les arêtes représentent des interactions physiques entre ces protéines.

Ces interactions peuvent être retrouvées soit expérimentalement (*in vivo*), soit via des prédictions informatiques (*in silico*). Parmi les méthodes expérimentales, on peut citer la technique du double-hybride [ICO+01], la spectrométrie de masse et la méthode de purification d'affinité en tandem (Tap Tag [GBK+02]). Quant aux méthodes *in silico*, un exemple classique est la méthode de recherche de gènes

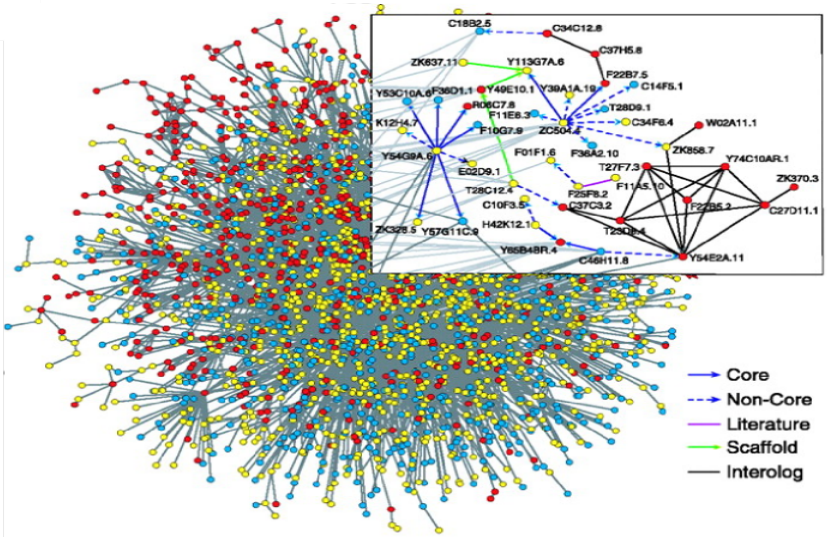


FIGURE 2.2 – Exemple de réseau PPI : l'interactome de *C.elegans*. Image extraite de [LAB+04].

fusionnés (Pierre de Rosette [Dat08]), d'autres méthodes peuvent utiliser des profils phylogénétiques [PMT+99] ou encore se baser sur le voisinage des gènes [GKM00].

Les méthodes d'alignement de réseaux PPI chercheront typiquement à aligner des réseaux correspondant à plusieurs espèces (chaque réseau primaire correspond donc à une espèce), afin d'identifier les complexes protéiques conservés dans l'évolution.

Nous commençons par détailler une par une les principales approches de la littératures, puis nous les regroupons dans la table 2.2.2.4. Dans cette table, on retrouve dans les différentes colonnes les choix que nous avons annoncés en introduction : le choix de sélection des empilements, le choix des arêtes entre empilements et enfin le choix des sous-graphes du multigraphe d'alignement qui seront rendus en résultat. Le choix de la relation de correspondance S n'est pas mentionné parce que toutes ces approches utilisent en pratique une relation seuillée calculée à partir de scores BLAST.

2.2.1 Approches 2 à 2

Dans la grande majorité des cas d'alignement 2 à 2 de réseaux PPI, la relation de correspondance S entre nœuds (Choix 1) est basée sur la similarité des séquences des protéines associées à ces nœuds. Il s'agit souvent d'une relation seuillée (S_{seuil})

calculée à partir d'un score d'alignement donné par BLAST. Bien entendu, dans le cas 2 à 2, les empilements (Choix 2) sont simplement des couples de protéines en correspondance.

2.2.1.1 HopeMap

Bien que relativement récente, HopeMap [TS09] est une des approches proposées les plus simples. Elle se décrit aisément dans le cadre général du multigraphe d'alignement (MGA) décrit dans le début de ce chapitre.

Choix 3 : définition des arêtes entre empilements
une arête est présente entre deux empilements $u = (u_1, u_2)$ et $v = (v_1, v_2)$ si et seulement si $(u_1, v_1) \in E_1$ et $(u_2, v_2) \in E_2$, c'est-à-dire si les arêtes sont présentes dans les deux graphes primaires. Dans la suite nous parlerons d'arêtes "*communes*".

Choix 4 : sous-graphes résultats
les résultats recherchés sont les composantes connexes du multigraphe d'alignement.

L'avantage principal de cette approche est sa rapidité, le calcul des composantes connexes restant linéaire dans le nombre de nœuds et d'arêtes du multigraphe d'alignement.

2.2.1.2 PathBLAST

PathBLAST définit un alignement comme un chemin élémentaire (chaque sommet du chemin n'est parcouru qu'une fois) de score maximal du MGA.

L'approche va en pratique construire le MGA puis chercher à retrouver dans ce graphe les chemins élémentaires de longueur fixée L qui maximisent un certain score.

Choix 3 : définition des arêtes entre empilements
En ce qui concerne la construction des arêtes reliant les empilements, les auteurs introduisent trois types d'arêtes différentes (cf figure 2.3).
Pour deux nœuds $u = (u_1, u_2)$ et $v = (v_1, v_2)$, le type de l'arête (u, v) va dépendre de la longueur des chemins entre u_1 et v_1 sur le premier graphe et entre u_2 et v_2 sur le second. Formellement l'arête sera :

- *directe* ssi $(u_1, v_1) \in E_1$ et $(u_2, v_2) \in E_2$
  (les deux chemins sont de longueur 1 : il s'agit des arêtes communes),
- *gap* ssi $(u_1, v_1) \in E_1$, $(u_2, v_2) \notin E_2$ et $\exists w_2 \in V_2$, $(u_2, w_2) \in E_2$ et $(w_2, v_2) \in E_2$

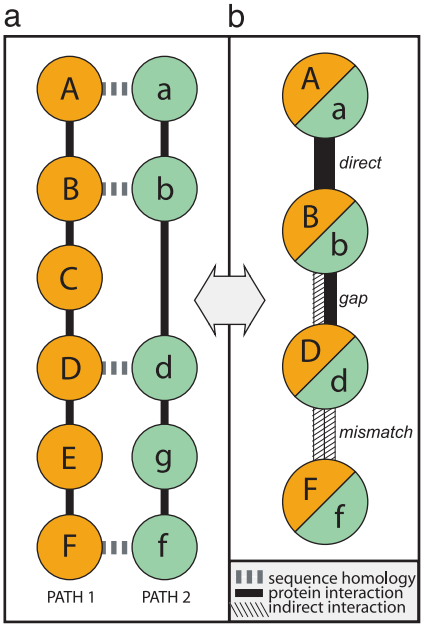


FIGURE 2.3 – Exemples d'arêtes directes, de mismatch et de gap. Les empilements (A, a) et (B, b) sont reliés par une arête directe parce que le chemin reliant A à B et le chemin reliant a à b sont tous les deux de longueur 1. Entre les empilements (B, b) et (D, d) par contre il s'agit d'une arête de gap, puisque le chemin reliant b à d est de longueur 1, mais celui reliant B à D est de longueur 2. Enfin (D, d) et (F, f) sont reliés par une arête de mismatch parce que les deux chemins correspondants sont de longueur 2. Figure extraite de PATHBLAST [KSK+03].

(un chemin de longueur 1, l'autre de longueur 2),

– *mismatch* ssi $(u_1, v_1) \notin E_1$, $(u_2, v_2) \notin E_2$, $\exists w_1 \in V_1$ tel que $(u_1, w_1) \in E_1$ et $(w_1, v_1) \in E_1$ et $\exists w_2 \in V_2$ tel que $(u_2, w_2) \in E_2$ et $(w_2, v_2) \in E_2$
(les deux chemins sont de longueur 2).

Choix 4 : résultats

Les sous-graphes résultats sont les chemins élémentaires de longueur fixée L et de score maximal.

Le score d'un chemin P est basé à la fois sur les arêtes et les nœuds :

$$S(P) = \sum_{v \in P} log(\frac{p(v)}{p_{random}}) + \sum_{e \in P} log(\frac{q(e)}{q_{random}})$$

où $p(v)$ et $q(e)$ mesurent respectivement la vraisemblance que l'empilement v corresponde à deux protéines homologues et que l'arête e corresponde à de véritables interactions physiques, et où p_{random} et q_{random} correspondent à l'espérance de $p(v)$ et $q(e)$ sur tous les nœuds et toutes les arêtes du multigraphe d'alignement. Le

détail des formules est donné dans [KSK+03].

Pour un graphe acyclique G (c'est-à- dire une forêt) donné et une longueur de chemin L fixée, le chemin de score maximal est trouvé par programmation dynamique en un temps linéaire en le nombre d'arêtes de G.
Puisque le multigraphe d'alignement n'est pas nécessairement acyclique, les auteurs ont recours à la suppression aléatoire d'arêtes pour transformer ce multigraphe en graphe acyclique. Ils remarquent qu'en moyenne le chemin de score maximal du MGA est conservé dans $\frac{2}{L!}$ des graphes acycliques, les auteurs décident donc de relancer le processus $5L!$ fois, puis de regrouper les résultats individuels suivant une heuristique gloutonne visant à limiter les redondances.

2.2.1.3 MaWISH

Dans cette approche [KKT+06], les auteurs introduisent la prise en compte des duplications de gènes. Pour ce faire, on dispose d'un score de similarité S compris entre 0 et 1 qui va s'appliquer aussi à des nœuds du même réseau. En pratique, pour deux protéines u et v, dès que $S(u, v) > 0$, les auteurs considèrent que les protéines sont en correspondance.

Notons maintenant $\Delta(u_1, u_2)$ la longueur du plus court chemin entre deux nœuds u_1 et u_2 du premier réseau et $\Delta'(v_1, v_2)$ la longueur du plus court chemin entre deux nœuds v_1 et v_2 du second réseau. Afin de définir les différents types d'arêtes, les auteurs utilisent un seuil $\bar{\Delta}$ sur la longueur de ces chemins.

Choix 3 : définition des arêtes entre empilements
Il y a trois types de pondérations différentes sur les arêtes entre empilements (cf figure 2.4).
Entre deux empilements (u_1, v_1) et (u_2, v_2), l'arête sera pondérée par un score de :

- *match* si ($S(u_1, u_2) > 0$ et $\Delta'(v_1, v_2) \leq \bar{\Delta}$) ou ($S(v_1, v_2) > 0$ et $\Delta(u_1, u_2) \leq \bar{\Delta}$)
- *mismatch* si ($S(u_1, u_2) > 0$ et $\Delta'(v_1, v_2) > \bar{\Delta}$) ou ($S(v_1, v_2) > 0$ et $\Delta(u_1, u_2) > \bar{\Delta}$)
- *duplication* si ($v_1 = v_2 \wedge S(u_1, u_2) > 0$) ou ($u_1 = u_2 \wedge S(v_1, v_2) > 0$) .

Notons qu'une arête peut être pondérée à la fois par un score de match ou mismatch et par un score de duplication. Si entre deux empilements il n'y a pas de match, mismatch ou duplication, l'arête n'est pas construite.

Choix 4 : sous-graphes résultats
Les solutions recherchées sont tous les sous-graphes de score supérieur à un seuil.

Le score d'un sous-graphe est défini comme la somme des pondérations de

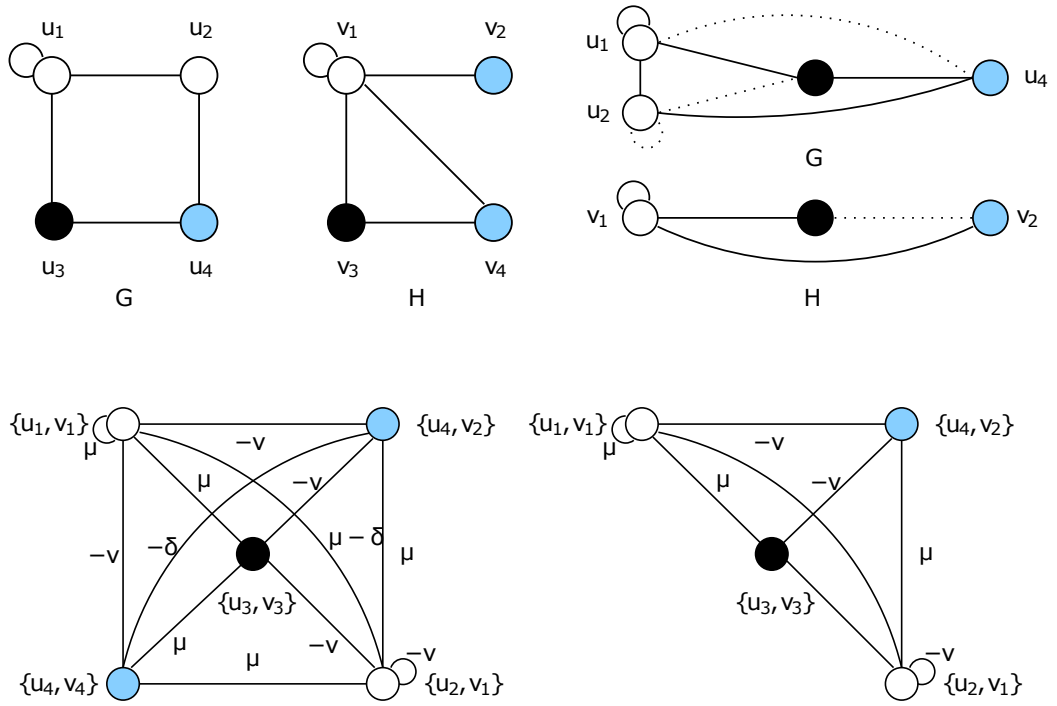


FIGURE 2.4 – Exemples d'arêtes de match (μ), mismatch (ν) et duplication (δ) pour un seuil $\bar{\Delta} = 1$. En haut à gauche, on donne les deux graphes à aligner, les nœuds de même couleur sont les nœuds similaires. Ces couples de nœuds similaires sont ensuite empilés dans le multigraphe d'alignement représenté en bas à gauche, les empilements sont reliés par des arêtes pondérées. Par exemple l'arête reliant les empilements (u_1, v_1) et (u_2, v_1) est à la fois une arête de match parce que le plus court chemin reliant u_1 à u_2 est de longueur 1 et que celui reliant v_1 à lui-même est de longueur 0, et une arête de duplication puisque $S(u_1, u_2) > 0$. En bas à droite, les auteurs donnent le sous-graphe résultat (dont le score est supérieur au seuil qu'ils ont fixé), et ils donnent l'alignement correspondant en haut à droite. Figure extraite de MaWISH [KKT+06].

ses arêtes, sachant que les arêtes de match ont des scores positifs, les arêtes de mismatch ont des scores négatifs et les arêtes de duplication peuvent avoir un score positif ou négatif.

Les auteurs montrent que le problème de recherche d'un sous-graphe de score supérieur à un seuil est NP-complet (par réduction du problème de la clique maximum 2.4) et ont donc recours à une heuristique gloutonne pour le résoudre. Ils commencent par le nœud qui possède le plus de voisins en match, ils agglomèrent itérativement ces voisins tant que le score ne diminue "pas trop". En pratique, cela signifie qu'un gain négatif pourra être accepté (afin d'éviter les mauvais maxima locaux). Lorsqu'un sous-graphe de poids localement maximal a été ainsi identifié, les nœuds du multigraphe d'alignement sont marqués et la procédure est répétée sur les nœuds restants.

2.2.1.4 IsoRank

La caractéristique principale de cette approche [SXB07] est qu'il s'agit d'un alignement global (et non pas local) de deux réseaux.

L'algorithme ne construit pas explicitement le multigraphe d'alignement, mais recherche une mise en bijection ("mapping") entre tous les nœuds des deux réseaux primaires qui maximise un certain score.
Ce score fait intervenir deux composantes :

- la similarité en séquence des nœuds : $ds(p_i, p_j)$,
- la similarité des voisins de ces nœuds.

L'intuition est qu'un couple (p_i, p_j) est un "bon match" si leurs séquences sont similaires et si leurs voisins sont également de "bons matchs". Le calcul du score R_{ij} de (p_i, p_j) fait donc appel à une formule récursive dépendant du score de ses voisins ($N(v_i)$ est le voisinage de v_i dans son réseau PPI) :

$$R_{ij} = \sum_{\substack{v_u \in N(v_i) \\ v_w \in N(v_j)}} \frac{R_{uw}}{|N(v_u)||N(v_w)|}.$$

En pratique, les auteurs utilisent une moyenne entre ce score de voisinage topologique sur les réseaux et la similarité de séquence. La formule finale est exprimée sous forme matricielle (équation aux valeurs propres) : $R = \alpha AR + (1-\alpha)E$ où E est la matrice de similarité de séquence, α est un paramètre,
et $A_{ij,uw} = \begin{cases} \frac{1}{|N(v_u)||N(v_w)|} & \text{si } v_u \in N(v_i), v_w \in N(v_j), \\ 0 & \text{sinon.} \end{cases}$

Bien entendu cette matrice A est de taille considérable (N^4 si N est le nombre de protéines par espèce) mais elle est très creuse et les auteurs ont donc recours aux

algorithmes utilisés sur ce type de matrice (technique "power method" [GVL96]). Une fois l'équation résolue, c'est-à-dire une fois la matrice R calculée, les auteurs en extraient un alignement global via une heuristique gloutonne : on sélectionne la paire de plus haut score, on supprime la ligne et la colonne correspondantes dans R et on recommence.

Cette approche a été ensuite étendue au cas multiple (IsoRankN) [SXB08]. Pour cela, les auteurs commencent par appliquer la méthode IsoRank (sans l'heuristique gloutonne finale) à chaque paire de réseaux, ce qui leur permet de construire un graphe n-parti complet (les n "étages" du graphe correspondent chacun à un réseau primaire) dont les arêtes sont pondérées par le score de similarité IsoRank (en pratique cela correspond donc à $\frac{n(n-1)}{2}$ matrices, une par paire de réseaux). Dans ce graphe, ils vont ensuite rechercher de façon heuristique des clusters de nœuds fortement similaires.

L'intérêt de cette approche est qu'elle se généralise à d'autres données que les données d'interaction protéine-protéine comme nous le verrons plus loin. D'une manière générale, elle établit une correspondance intéressante entre un problème de "mapping" prenant en compte la topologie des réseaux et une équation aux valeurs propres. Cette approche a d'ailleurs initialement été proposée pour le classement de pages web par Google (PageRank [PBMW98]) d'où le nom IsoRank choisi par les auteurs.

2.2.2 Approches multiples

Dans les approches multiples, le choix de la relation de correspondance (Choix 1) est identique au cas 2 à 2 : il s'agit le plus souvent d'une relation seuillée S_{seuil} basée sur le score de similarité des séquences protéiques (lui-même le plus souvent calculé par BLAST). En revanche, l'élément nouveau sera le choix de l'aggrégation des nœuds primaires pour former les empilements de taille > 2 (Choix 2).

2.2.2.1 NetworkBLAST

NetworkBLAST [SSK+05] est une adaptation de PathBLAST - dont nous avons parlé plus haut - à l'alignement de trois réseaux.

Choix 2 : définition des empilements
les empilements sont des composantes connexes de la relation S, c'est-à-dire :
$u = (u_1, u_2, u_3) \in \mathcal{V} \Leftrightarrow u_1, u_2, u_3$ forme une composante connexe de S

Choix 3 : définition des arêtes entre empilements
comme dans le cas de HopeMap, une arête est présente dans le MGA si les nœuds

correspondants dans tous les graphes sont directement connectés, c'est-à-dire : $e = (u, v) \in \mathcal{E} \Leftrightarrow \forall i (u_i, v_i) \in E_i$

Choix 4 : sélection des sous-graphes
parmi les sous-graphes du multigraphe d'alignement, l'algorithme - suivant l'option choisie - peut rendre comme résultat l'ensemble des chemins de longueur maximale ou bien les "clusters" (sous-graphes dont la densité est supérieure à un seuil donné).

2.2.2.2 NetworkBLAST-M

L'approche de NetworkBLAST-M [KBS08] est basée - comme NetworkBLAST - sur la recherche de sous-réseaux "denses" dans chacun des réseaux primaires. Les auteurs travaillent directement sur l'ensemble des réseaux primaires plus la relation de correspondance (ils l'appellent "layered data graph", nous l'appellerons plus tard "graphe de données stratifié"). La "densité" sera en pratique assurée par une maximisation heuristique d'un score. Voyons comment ce score est défini.

Les auteurs disposent dans chacun des réseaux primaires d'arêtes pondérées, c'est-à-dire qu'on dispose pour chaque interaction d'un indice de confiance dans l'interaction. C'est cet indice de confiance qui permet de calculer un score d'interaction (en pratique, un rapport de vraisemblance) entre deux protéines u et v : $p(u, v)$. La formule exacte est donnée dans [KBS08].

Etant donnée une espèce i et le réseau primaire associé $G_i = (V_i, E_i)$, un sous-graphe $G'_i \subseteq G_i$ induit par l'ensemble de nœuds $V'_i \subseteq V_i$ aura pour score $L(V'_i) = \sum_{u,v \in V'_i} p(u, v)$.

Etant donné maintenant un sous-graphe $G' = (V', E')$ du multigraphe d'alignement G, il aura pour score $S(G') = \sum_{i=1}^{n} L(V'_i)$ où $V'_i = \bigcup_{(v_1,\ldots v_n) \in V'} \{v_i\}$. C'est intuitivement la somme des scores de ses "projections" sur chacun des réseaux primaires.

Avec cette définition, l'objectif est de rechercher les sous-réseaux du multigraphe d'alignement de fort score (forte "densité"). Il s'agit d'un problème difficile, même sur un unique réseau avec des poids d'arêtes égaux à 1 ou -1. Les auteurs le rapprochent du problème du "Cluster Editing" - c'est-à-dire le problème consistant à trouver le nombre minimal d'opérations d'ajout et de suppression d'arête permettant de transformer un graphe donné en union disjointe de cliques - problème

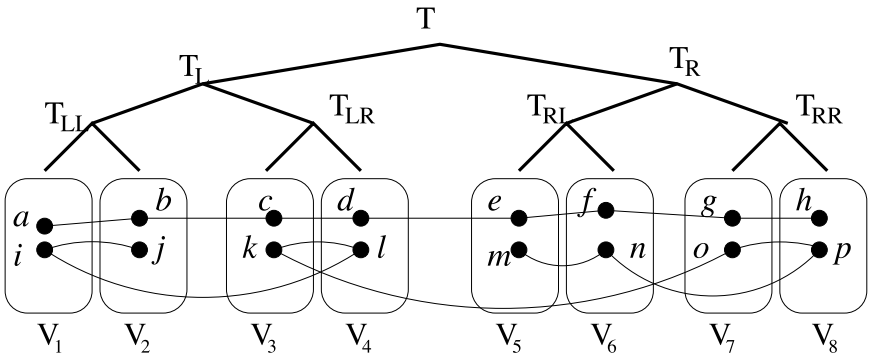


FIGURE 2.5 – Exemples de chemins compatibles avec un arbre T. La notation parenthésée de l'arbre est la suivante : $(((1,2),(3,4)),((5,6),(7,8)))$. Prenons l'empilement (i,j,k,l,m,n,o,p). Pour qu'il soit compatible avec T, il faut que pour chaque nœud interne de l'arbre (clade) il forme un chemin. C'est bien le cas pour T_{LL} (c'est-à-dire $\{1,2\}$) puisque i est relié à j. Pour T_{LR}, T_{RL} et T_{RR}, c'est aussi vrai. Voyons si c'est aussi le cas pour T_L ($\{1,2,3,4\}$) et T_R ($\{5,6,7,8\}$) : on peut exhiber le chemin $j-i-l-k$ dans le premier cas, et le chemin $m-n-p-o$ dans le second. Enfin c'est vrai aussi pour T car $j-i-l-k-o-p-n-m$ est un chemin. Figure extraite de [KBS08].

NP-complet, *cf* [**?**]. Ils choisissent donc d'utiliser une heuristique gloutonne pour le résoudre.

La méthode propose deux choix différents d'empilements.

Choix 2 : définition des empilements

- chemins avec une topologie identique (c'est-à-dire correspondant à la même suite de réseaux),
- chemins compatibles avec un arbre phylogénétique donné en entrée (cf exemple sur la figure 2.5).

Définissons plus précisément cette notion intéressante de chemins compatibles avec un arbre phylogénétique. Considérons un arbre binaire T représenté par son expression parenthésée : chaque couple de parenthèses représente un embranchement dans la phylogénie, c'est-à-dire un nœud interne de l'arbre. Chaque espèce (feuille) correspond dans le problème qui nous occupe à un réseau primaire repéré par un indice. Considérons pour chaque couple de parenthèses, l'ensemble des indices compris entre ces parenthèses. Appelons chacun de ces ensembles d'indices un clade, notons $\mathcal{C}(T)$ l'ensemble des clades de T.

Un empilement $(v_1, \dots v_n)$ est un chemin compatible avec T si et seulement si

pour chacun des clades $I \in \mathcal{C}(T)$, l'empilement réduit aux indices de I contient un chemin pour la relation de correspondance S.

L'intérêt de cette approche est double, le premier est l'introduction des empilements en arbre, le second est qu'il s'agit à notre connaissance du premier article où les auteurs évitent le calcul explicite du multigraphe d'alignement. Nous reprendrons cet exemple dans la section 4.5.

2.2.2.3 GRAEMLIN

GRAEMLIN est une approche progressive d'alignement multiple, qui commence par aligner deux réseaux primaires initiaux, puis étend l'alignement à un nouveau réseau, et ainsi de suite.

L'algorithme [FNS+06] commence par l'énumération de tous les clusters "denses" de d points (d-clusters) dans chacun des 2 réseaux primaires initiaux. La définition de la densité est très voisine de celle utilisée dans NetworkBLAST-M (basée sur un indice de confiance des interactions). Il met ensuite en correspondance ("mapping") les protéines des d-clusters. Chaque paire de d-clusters se voit affectée un score correspondant au meilleur mapping entre ces deux d-clusters. Toutes les paires de clusters de score supérieur à un seuil fixé sont conservées et constitueront des "graines". Les "mappings" correspondant à chacune de ces graines sont ensuite étendus progressivement par une méthode gloutonne et GRAEMLIN retient le "mapping" de score final maximal.

Les deux réseaux initiaux donnent ainsi naissance à trois sous-réseaux, le premier correspond à la partie alignée des deux réseaux, et les deux autres aux parties non-alignées de chacun des deux réseaux initiaux.

La procédure continue en alignant le réseau suivant dans l'arbre phylogénétique avec chacun de ces trois graphes.

2.2.2.4 CAPPI

L'originalité de la méthode CAPPI [DT07] est qu'elle cherche à reconstruire explicitement le réseau PPI ancestral, appelé réseau CAPPI (pour Conserved Ancestral PPI), ce qui lui permet ensuite de "projeter" l'information obtenue sur les réseaux afin de déterminer l'alignement.

Afin de retrouver le réseau ancestral, les auteurs commencent par partitionner l'ensemble des protéines des réseaux primaires en classes disjointes (l'algorithme utilisé est TRIBE-MCL [EVDO02]) en utilisant les E-values BLAST comme mesure de distance entre protéines (Choix 2). Idéalement ces classes correspondent

donc chacune à un ensemble de protéines homologues, c'est-à-dire qui possèdent un ancêtre commun. Chaque classe correspond donc à un nœud du réseau ancestral.

L'idée est ensuite de retrouver les interactions entre ces nœuds du réseau ancestral. En pratique, les auteurs construisent le graphe complet, et associent à chaque arête une probabilité d'interaction.

Cette probabilité est obtenue en deux étapes. Pour chaque classe, les auteurs commencent par reconstruire une phylogénie (par une combinaison de CLUSTALW [THG94], PROTDIST et de Neighbor Joining - ces deux méthodes sont issues du package PHYLIP [Fel89]), qui est réconciliée avec l'arbre phylogénétique donnée en entrée. En se basant sur ces phylogénies et sur un modèle de duplication-divergence (décrit dans [DT07]), les auteurs se ramènent ensuite à un problème d'inférence de réseau Bayésien, ce qui leur permet de calculer la probabilité *a posteriori* d'interaction entre protéines à la racine des arbres (connaissant les probabilités d'interaction de ses feuilles, c'est-à-dire des protéines des réseaux primaires).

Les auteurs appliquent enfin un seuil au graphe complet dont ils viennent de pondérer les arêtes et s'intéressent aux composantes connexes ainsi trouvées. Ces composantes sont appelées "modules ancestraux". La qualité de ces modules est mesurée en croisant les regroupements ainsi obtenus avec des catégories fonctionnelles déjà connues (MIPS [MHPF98], GO [ABB+00]). Notons que ce seuil peut être déterminé par l'algorithme lui-même : le principe est d'assurer un seuil suffisant pour que les sous-graphes soient statistiquement significatifs par rapport à un modèle statistique.

2.3 Données métaboliques

Actuellement, les données métaboliques proviennent de deux sources principales : KEGG [KG00] et BIOCYC [CFF+06]. Dans les deux cas il s'agit de bases de données regroupant des réactions chimiques et des catalyseurs (protéines). Les données se présentent sous la forme d'un grand réseau, éventuellement découpé en "pathways". Dans le cas de KEGG la base est multi-organismes, dans le cas de BIOCYC elle est orientée sur un organisme précis (par exemple la base ECOCYC [KCVGC+05] est dédiée à *Escherichia coli*).

Typiquement, un réseau primaire peut se présenter sous la forme :

– d'un graphe dont les nœuds sont les réactions et les arêtes relient deux réactions lorsque le substrat de l'une est produit de l'autre (**graphe des réactions**), ce graphe est parfois nommé graphe d'enzymes en assimilant les nœuds non pas aux réactions mais aux enzymes qui les catalysent. En théorie on devrait distinguer les deux types de graphes car la relation réaction - en-

Méthode	définition des empilements	définition des arêtes	sous-graphes résultats	référence
HopeMap	paires	communes	CC	[TS09]
PathBLAST	paires	directes / mismatch / gap	chemins de longueur L de poids maximal	[KSK+03]
MaWISH	paires	match / mismatch / duplication	sous-graphes de score $> \sigma$	[KKT+06]
NetworkBLAST	CC ($n = 3$)	communes	clusters denses ou chemins de longueur maximale	[SSK+05]
NetworkBLAST-M	chemins de même topologie ou chemins guidés par la phylogénie	toutes les arêtes	clusters denses sur chaque réseau	[KBS08]

Table 2.1 – Différentes approches d'alignement de réseaux PPI et les choix correspondants. La colonne *définition des empilements* donne la condition utilisée pour sélectionner les n protéines empilées suivant la relation de correspondance S. "CC" signifie composantes connexes. La colonne *définition des arêtes* explique quelles arêtes des réseaux PPI vont relier les empilements dans le multigraphe d'alignement, "communes" signifie que seules les arêtes communes à tous les réseaux sont utilisées. Enfin la colonne *sous-graphes résultats* explique quels types de sous-graphes sont extraits du multigraphe d'alignement, par exemple NetworkBLAST [SSK+05] retrouve les chemins de longueur maximale, alors que NetworkBLAST-M [KBS08] maximise une somme de scores, un par réseau, s'assurant que les sous-graphes sont "denses" sur chaque réseau. Les approches IsoRank [SXB07], GRAEMLIN [FNS+06] et CAPPI [DT07] n'apparaissent pas dans ce tableau car elles n'entrent pas bien dans le formalisme du multigraphe d'alignement.

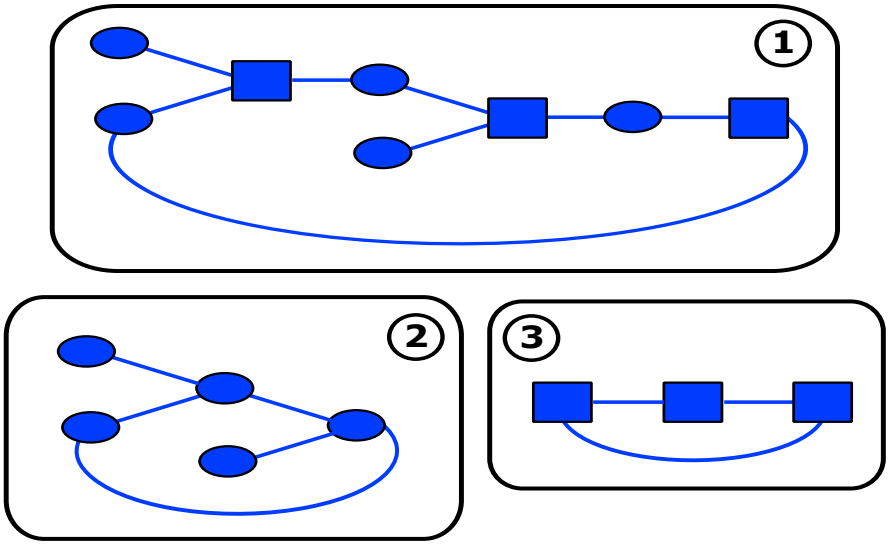


FIGURE 2.6 – Exemple de graphe métabolique sous la forme d'un graphe biparti "biochimique" (1), puis sous la forme d'un graphe des composés (2) et enfin sous la forme d'un graphe des réactions (3). Les réactions sont représentées par des rectangles, et les composés par des ronds.

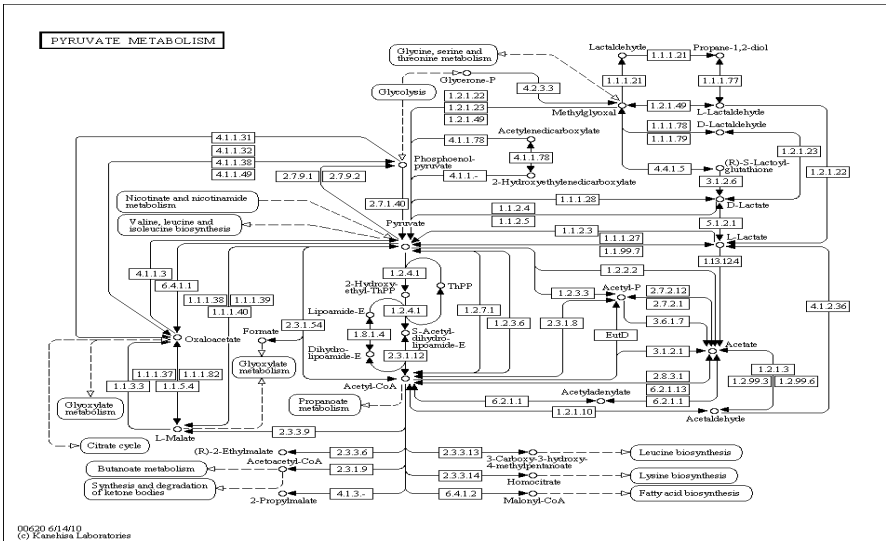


FIGURE 2.7 – Exemple de réseau métabolique : il s'agit d'un graphe biochimique, les nœuds rectangulaires sont des enzymes, les nœuds ronds sont des composés chimiques, les nœuds rectangulaires avec des coins arrondis renvoient vers d'autres réseaux. Image extraite de KEGG [KG00].

zyme n'est pas bijective (loin s'en faut), mais, en l'état actuel des données disponibles, il s'agit d'une approximation très courante.
- d'un graphe dont les nœuds sont les métabolites et une arête relie deux métabolites si l'un des deux peut être produit à partir de l'autre par une réaction chimique (**graphe des composés**).
- ou d'un graphe biparti possédant deux types de nœuds : les nœuds réaction et les nœuds métabolites, une arête relie un nœud réaction à un nœud métabolite si ce métabolite est substrat ou produit de la réaction (**graphe biochimique**) [5].

Du point de vue de la chimie, ces graphes sont naturellement orientés des substrats vers les produits et les arêtes sont pondérées par la stœchiométrie. Dans le cas du graphe des réactions par exemple, prenons deux réactions r_1 et r_2, telles que le substrat de r_2 soit le produit de r_1, l'arête (r_1, r_2) sera orientée de r_1 vers r_2. La figure 2.6 donne un exemple de réseau métabolique représenté sous chacune de ces trois formes : graphe biochimique, graphe des composés et graphe des réactions.

Pourtant, dans de nombreuses approches les graphes seront considérés comme non-orientés, parce que peu de réactions sont irr/'eversibles et que, en règle générale, cette information de réversibilité est absente des bases de données [6].
Une question centrale dans l'analyse de réseaux métaboliques est leur décomposition en modules, ou voies métaboliques (cf [LCTS08] pour une introduction détaillée à l'analyse structurelle des réseaux métaboliques). Malgré la relative rareté des données métaboliques, on trouve donc déjà dans la littérature un bon nombre d'approches d'alignement de réseaux métaboliques.

La majorité des approches se limite au traitement d'une requête : un "petit" graphe (graphe requête) qu'elles vont comparer à un grand graphe (graphe cible) issu d'une base de données, en pratique, elles procéderont à un alignement de ces deux réseaux. Dans une première sous-partie, nous allons présenter des approches de ce type avant de détailler des approches de la littérature qui effectuent des alignements entre réseaux entiers (2 à 2 ou multiples).

Les enzymes sont classées suivant leurs propriétés fonctionnelles dans une classification établie par l'IUBMB (*International Union of Biochemistry and Molecular Biology*). Dans cette classification, une enzyme reçoit un code à 4 nombres (EC number), chaque nombre de gauche à droite correspond à une catégorie de plus en plus spécifique de la classification.

Il est donc tentant d'employer comme mesure de similarité entre enzymes une

5. Certaines approches peuvent enrichir ce graphe biochimique d'un troisième type de nœuds, correspondant aux enzymes, qui sont alors liées aux réactions qu'elles catalysent.

6. Notons que la représentation la plus complète est celle d'un hypergraphe : les arêtes reliant les substrats (resp. produits) à une réaction sont regroupées en une seule arête de l'hypergraphe.

mesure basée sur ces nombres.

Plus précisément soient e_1 et e_2 deux enzymes respectivement associées aux EC numbers E_1 et E_2. On peut définir un indice de dissimilarité de la façon suivante :
$ds(e_1, e_2) = 4 -$ (longueur du plus long préfixe commun à E_1 et E_2)
et la relation S par $e_1 S e_2 \leftrightarrow ds(e_1, e_2) < \sigma$.
On notera S_{EC} ce type de relation de correspondance basée sur les EC numbers.

La définition précédente présente plusieurs problèmes :
- elle ne permet pas de traiter correctement le cas des enzymes multifonctionnelles (plusieurs EC possibles),
- la formule ne tient pas compte du fait que les classes (niveaux hiérarchiques) de la classification EC ne sont pas du tout uniformément peuplées (certaines classes sont très abondantes, d'autres rares).

Un indice prenant en compte le second aspect a été proposé par Tohsato et al [TMH00] : $ds(e_1, e_2) = log(C_{1,2})$ où $C_{1,2}$ est le nombre d'EC numbers qui ont comme préfixe le plus long préfixe commun à E_1 et E_2 (c'est-à-dire le nombre total de classes EC sous le nœud commun le plus proche de e_1 et e_2 dans la hiérarchie).

Comme pour les approches d'alignement de données d'interaction entre protéines, nous commençons par présenter les approches une par une, avant de résumer le tout dans la table 2.2.

2.3.1 Approches par requêtes

Dans le cadre des approches par requêtes, l'objectif est de retrouver, pour une voie métabolique donnée, toutes ses occurrences (exactes ou approximatives) dans une base de réseaux métaboliques. Les réseaux métaboliques traités apparaissent généralement sous la forme de **graphes des réactions**. Ce problème général est typiquement exprimé sous la forme d'un problème d'isomorphisme (ou d'homéomorphisme) de sous-graphe.

Un isomorphisme de graphe est une application entre deux graphes G_1 et G_2 qui respecte la structure de ces graphes. Il s'agit formellement d'une application bijective $f : V_1 \to V_2$ telle que $\forall u, v \in V_1$, $(u, v) \in E_1 \Leftrightarrow (f(u), f(v)) \in E_2$.

Le problème d'isomorphisme de sous-graphe consiste à décider s'il existe un isomorphisme entre un graphe motif et un sous-graphe du graphe cible.

La notion d'homéomorphisme est moins stricte, les deux graphes doivent être isomorphes modulo la suppression d'un nombre quelconque de nœuds de degré 2 dans l'un et l'autre des deux graphes. Intuitivement, il s'agit cette fois de rechercher

des occurences approximatives de la voie métabolique.

À noter que dans le cas général, les problèmes d'isomorphisme ou d'homéomorphisme de sous-graphe sont NP-complets.

2.3.1.1 MetaPathwayHunter

MetaPathwayHunter [PRYLZU05a] est une approche par requête, c'est-à-dire limitée à l'alignement d'un réseau requête sur un réseau cible.

Cette problématique particulière est exprimée ici comme une recherche de sous-graphe homéomorphe au graphe requête - c'est-à-dire de topologie identique, modulo des délétions de nœuds de degré 2.

Constatant que dans les bases de données métaboliques les circuits sont assez rares, les auteurs se limitent au cas où l'on restreint les réseaux métaboliques à des "arbres multi-sources" [7].

Grâce à cette restriction, l'algorithme parvient à trouver tous les arbres homéomorphes à l'arbre requête en temps polynomial. Il leur affecte ensuite un score permettant d'ordonner les résultats par "pertinence".

Ce score est inspiré de ce qui est fait classiquement pour l'alignement de séquences, il prend en compte :
- la similarité des nœuds mis en correspondance (substitution) par la mesure de dissimilarité de Tohsato *et al* précédemment définie [TMH00],
- l'insertion ou la délétion de nœuds de degré 2 (homéomorphisme).

2.3.1.2 Wernicke *et al*

L'article de Wernicke *et al* [WR07] détaille une approche par requête, appliquée au graphe des réactions, qui, contrairement au cas précédent, n'impose pas de contrainte de topologie. En particulier les cycles sont autorisés (dans la requête et la cible).

Les auteurs commencent par présenter un algorithme de backtracking "naïf" qui énumére tous les sous-graphes isomorphes au graphe requête (problème NP-complet) et retourne celui qui maximise le score (la formule est similaire à celle utilisée dans MetaPathwayHunter [PRYLZU05b]).

7. Un arbre multi-sources est un graphe orienté sans circuit (DAG) qui devient un arbre lorsqu'on lui retire son orientation (c'est-à-dire qui ne possède pas de cycles, même lorsqu'on ne tient pas compte de l'orientation).

L'algorithme est exponentiel, y compris pour une requête sous forme d'arbre. Les auteurs font le choix de réduire l'espace de recherche en introduisant un paramètre $f \in [0, 1]$ qui limite les occurences de gaps.

En pratique, pour un réglage $f = 0$, la recherche sera exhaustive et extrêmement lente, alors que pour le réglage $f = 1$, la recherche sera rapide mais sans garantie d'optimalité.

S. Wernicke and F. Rasche ([56]) ont formulé le problèmede l'alignement comme le problème de recherche the alignment as the maximum score em- bedding problem. Based on subgraph isomorphism, their embedding allows to stretch path or shorten path on both patten and text sides, which leads to homeomorphic embedding. Authors proposed a naive backtracking algorithm to exhaustively search for all simple paths and to obtain an optimal embedding. The algorithm takes exponential runtime even if the pattern is a tree. In order to reduce the search space, authors further propose to exploit local diversity which leads to the limited occurrences of consecutive gaps.

Suivant l'observation que les étiquettes des nœuds (c'est-à-dire les *EC* numbers) pour un graphe requête (ou cible) typique sont souvent très différentes ("local diversity"), les auteurs proposent donc naturellement de guider la recherche en utilisant la similarité de ces étiquettes.

Cette heuristique de guidage est paramétrée par une valeur $f \in [0, 1]$, le réglage $f = 0$ correspond à une recherche exhaustive alors que le réglage $f = 1$ sera plus rapide, mais n'offrira pas de garantie d'optimalité.

2.3.2 Approches d'alignement de réseaux entiers

Dans cette seconde sous-partie, nous allons détailler des approches qui effectuent des alignements entre réseaux entiers (2 à 2 ou multiples).

2.3.2.1 Tohsato *et al*

Ce premier algorithme d'alignement multiple de réseaux métaboliques s'applique uniquement à des graphes linéaires d'enzymes (le degré maximal des nœuds est 2).

L'alignement de deux réseaux est effectué par une adaptation d'un outil classique d'alignement global de deux séquences par programmation dynamique (Needleman et Wunsch [NW70]) aux réseaux métaboliques.

L'algorithme [TMH00] est ensuite étendu au cas de l'alignement multiple par

une approche progressive : deux graphes linéaires sont alignés, formant un "pattern" (chaque nœud est identifié par un EC number partiel : par exemple les enzymes 1.2.3.4 et 1.2.3.5 seront alignées en 1.2.3), qui pourra ensuite être aligné avec un troisième graphe et ainsi de suite. En pratique, l'algorithme est glouton, il calcule pour chaque paire de réseaux l'alignement 2 à 2 correspondant, et choisit celui qui a un score maximum : la paire de réseaux est remplacée par le pattern d'alignement, et l'algorithme est relancé.

2.3.2.2 M-PAL

M-PAL [LdRdGR08] est une méthode d'alignement local de deux réseaux métaboliques qui travaille sur les graphes biochimiques (réactions et composés) sans restriction de topologie (en particulier les cycles sont autorisés). Une caractéristique de cette méthode est qu'elle permet l'alignement de réactions dès lors qu'elles ont au moins un substrat et un produit en commun.

La méthode est basée sur la construction préalable de "building blocks". Un "building block" consiste en l'association de 1 ou 2 réactions du premier réseau à 1 ou 2 réaction du second suivant l'une des six règles suivantes (cf figure 2.8) :

(a) cas identité (empilement de 1 réaction sur 1 réaction) : les deux réactions sont strictement identiques ; les enzymes sont similaires (les 2 premiers niveaux EC sont identiques), les substrats et les produits sont identiques,

(b) cas direct (empilement de 1 réaction sur 1 réaction) : les enzymes sont similaires, il y a au moins un substrat et un produit en commun,

(c) cas de mismatch d'enzyme (empilement de 1 réaction sur 1 réaction) : les enzymes ne sont pas similaires mais il y a au moins un substrat et un produit en commun,

(d) cas direct avec gap (empilement de 1 réaction sur 2 réactions) : comme le cas direct avec sur un des réseaux une réaction supplémentaire,

(e) cas de mismatch d'enzyme avec gap (empilement de 1 réaction sur 2 réactions) : comme le cas de mismatch d'enzymes avec sur un des réseaux une réaction supplémentaire,

(f) cas de match d'enzymes réordonné (empilement de 2 réactions sur 2 réactions) : deux réactions consécutives A suivie de B sont empilées avec deux réactions consécutives B' suivie de A' où l'enzyme catalysant A (resp. B) est similaire à celle catalysant A' (resp. B').

L'algorithme commence par construire ces "building blocks", puis les assemble en voies courtes (quatre "building blocks") sans cycles. Les auteurs imposent de plus qu'un assemblage comporte au moins 3 building blocks directs ou identiques.

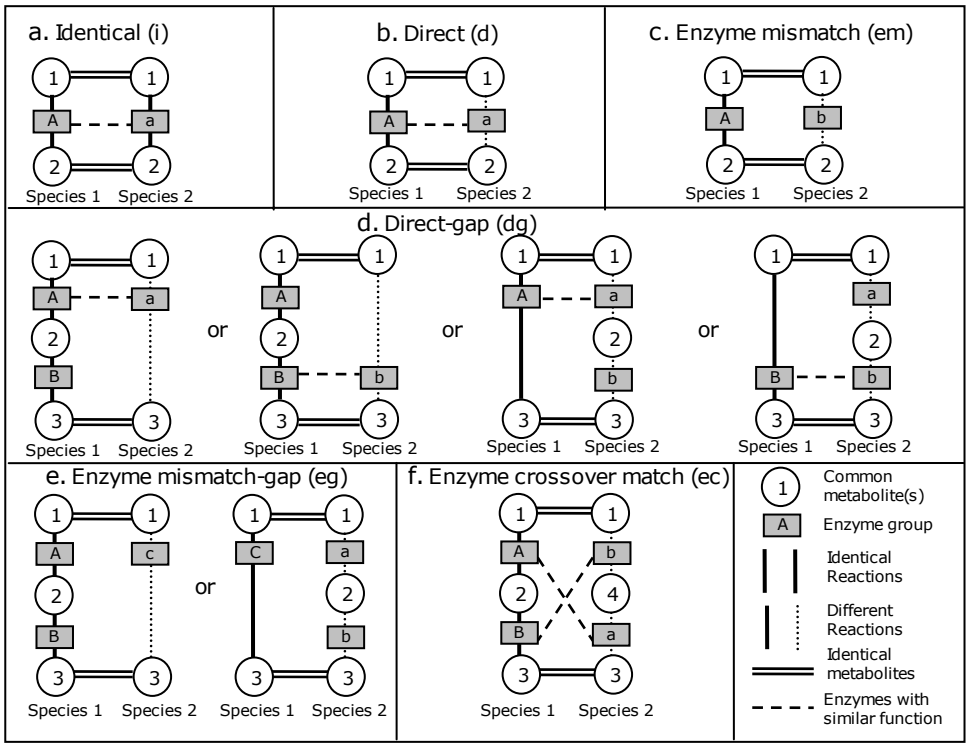


FIGURE 2.8 – Figure extraite de M-PAL [LdRdGR08] détaillant les différents cas de "building blocks". Les nœuds ronds sont des métabolites, les nœuds rectangulaires sont des enzymes et les réactions apparaissent sous la forme d'arêtes verticales. Dans le cas d'identité (a), les réactions sont identiques. Dans tous les autres cas elles sont différentes (traits pleins d'un côté, traits pointillés de l'autre). Dans les cas de match (a, b, d et f) les enzymes sont similaires : il y a une arête reliant les nœuds rectangulaires, alors que dans les cas de mismatch (c, e) ce n'est pas le cas. Dans les cas avec gap (dg et eg) il y a une réaction supplémentaire chez une des deux espèces. Enfin dans le cas de match d'enzymes réordonné (f) on constate que les deux réactions ne sont pas dans le même ordre sur les deux espèces.

Chacun des résultats reçoit ensuite un score qui se base sur la similarité entre enzymes (à la fois du point de vue de la similarité de séquence et du point de vue de la similarité fonctionnelle par les *EC* numbers).

2.3.2.3 MetaRank

Cette approche [AKdCL09] développée par Ay *et al* travaille également sur les graphes biochimiques sans restriction de topologie.

Le principe de l'algorithme est similaire à celui d'IsoRank dont nous avons déjà parlé dans la section traitant des méthodes d'alignement de réseaux PPI. L'algorithme n'étant pas nommé par ses auteurs, nous avons pris la liberté de le baptiser MetaRank.

Comme pour IsoRank, l'idée générale est de se ramener à un calcul de valeurs / vecteurs propres d'une matrice de dissimilarité prenant en compte la topologie des réseaux. Rappelons qu'un vecteur propre de cette matrice correspond à un appariement ("mapping") des nœuds des deux réseaux, pondéré par la valeur propre correspondante.

Les auteurs commencent par résoudre séparément les trois problèmes aux valeurs propres associés aux trois matrices de dissimilarité calculées respectivement sur les réactions, les enzymes et les composés chimiques.

Pour les enzymes, la similarité est mesurée sur la base des *EC* numbers (S_{EC}). Pour les composés, la similarité est basée sur la structure chimique. Pour les réactions, c'est une combinaison linéaire de similarité des enzymes d'une part, et des composés d'autre part.

Les auteurs partent ensuite des solutions du problème des réactions, c'est-à-dire d'un graphe biparti pondéré entre les réactions, dont ils extraient un mapping de score maximal. Ce mapping est ensuite utilisé pour retirer des arêtes des deux autres graphes bipartis pondérés (composés et enzymes) qui sont incompatibles. Ils extraient enfin les mappings de score maximal de ces deux graphes bipartis.

2.4 Données génomiques

Dans cette partie, nous nous limiterons à l'alignement de données génomiques à l'échelle de génomes complets. La taille de ces génomes peut varier de quelques mégabases dans le cas de procaryotes à plusieurs gigabases dans le cas d'eucaryotes. Les unités que nous chercherons à aligner ne seront donc pas les nucléotides mais les gènes. Notons tout de suite que les opérations d'édition classiquement utilisées

Méthode	relation de correspondance	nombre de réseaux	description	référence
MetaPathwayHunter	*EC*	deux	calcul des sous-graphes homéomorphes	[PRYLZU05a]
Wernicke *et al*	*EC*	deux	calcul des sous-graphes homéomorphes	[WR07]
Tohsato *et al*	*EC*	multiples	alignement global progressif	[TN07]
M-Pal	*EC* $\mid \cap_{subtrats/produits} \mid$	deux	calcul de voies courtes sans cycles	[LdRdGR08]
MetaRank	*EC* SIMCOMP	deux	résolution d'équations aux valeurs propres	[AKdCL09]

Table 2.2 – Différentes approches d'alignement de réseaux métaboliques, avec leur choix de relation de correspondance, le nombre de réseaux traités et une description succinte de la méthode. Concernant la relation de correspondance, par exemple dans le cas de M-Pal, elle est basée à la fois sur les *EC* numbers et sur le cardinal de l'intersection des ensembles de produits/substrats alors que dans l'approche MetaRank, c'est une combinaison d'*EC* numbers et d'un indice indiquant la similarité des structures chimiques ($SIMCOMP$).

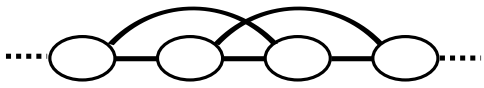


FIGURE 2.9 – Exemple de graphe représentant un génome.

dans l'alignement multiple de séquences ne sont plus valides ici : en plus des mutations ponctuelles (insertions, délétions et substitutions), le génome peut être affecté par des réarrangements de grande ampleur, pouvant concerner d'une centaine de bases à un chromosome entier : il peut s'agir d'inversions, translocations, transpositions, fusions, fissions, duplications ou délétions. Un grand nombre d'approches maintiennent une contrainte de conservation d'ordre entre les génomes, ce qui permet d'utiliser des méthodes issues de l'alignement de séquences, alors que d'autres approches lèvent cette contrainte pour se rapprocher de la réalité biologique.

Un génome complet sera représenté par un graphe dont les nœuds sont des gènes, et où un gène g sera lié par une arête à ses Δ plus proches voisins dans chaque direction sur le génome [8]. Si $\Delta = 1$, les arêtes relieront donc les gènes contigus sur le génome. Cette structure particulière du multigraphe d'alignement permet un bon nombre de simplifications algorithmiques.

Bien entendu, comme les nœuds des graphes primaires sont des gènes, il est naturel d'utiliser pour mesure de similarité entre nœuds un indice basé sur la similarité des séquences (nucléiques ou, plus souvent, protéiques [9]) et la relation de correspondance en sera souvent déduite par seuillage (S_{seuil}).

Fréquemment, les auteurs font usage des termes d'homologie, orthologie et paralogie pour désigner cette relation de correspondance. Il s'agit souvent d'un abus de langage et il apparaît utile de rappeler ici la définition précise de ces termes.

On dit de deux gènes qu'ils sont **homologues** lorsqu'ils dérivent d'un gène ancestral commun. Suivant la définition proposée par Fitch [Fit00], on distingue alors deux cas, suivant que l'évènement phylogénétique le plus récent séparant les deux gènes est une spéciation (apparition des deux espèces) - on dit alors que les deux gènes sont **orthologues** - ou une duplication (apparition de deux copies du gène) - on dit alors que les deux gènes sont **paralogues**.

L'exemple de la figure 2.10 permet de mettre en évidence deux propriétés im-

8. Il s'agit donc formellement d'un graphe d'intervalle. Le plus souvent il s'agira d'un graphe linéaire.

9. En raison de sa rapidité, la plupart des approches auront, une fois de plus, recours à BLAST pour calculer cette similarité.

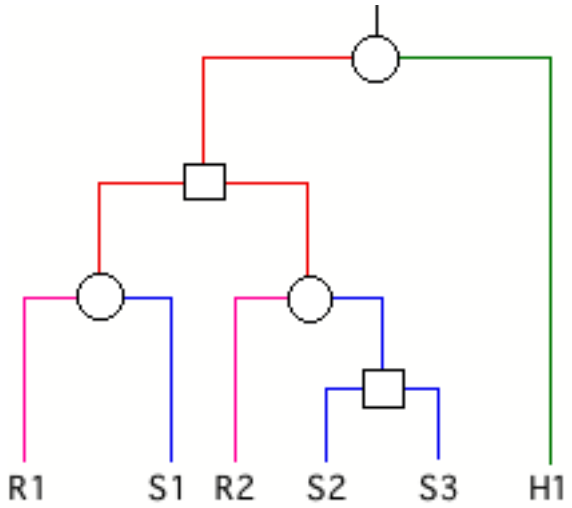


FIGURE 2.10 – Cette figure (adaptée de [Fit00]) représente un arbre de gènes. Les cercles représentent un évènement de spéciation (la couleur des branches correspond à une même espèce) et les carrés représentent un évènement de duplication génique. les gènes $R1$ et $S1$, par exemple, sont orthologues (de même que $R2$ et $S2$ ou $R2$ et $S3$). En revanche $S2$ et $S3$ sont paralogues (de même que $R1$ et $S2$ ou $R1$ et $S3$ par exemple).

portantes des relations d'orthologie/paralogie :

- elles ne sont généralement pas transitives (par exemple $S2$ est orthologue à $R2$, $R2$ est orthologue à $S3$ mais $S2$ et $S3$ ne sont pas orthologues),
- elles ne sont généralement pas bijectives : c'est-à-dire qu'un gène, dans une espèce peut présenter plusieurs orthologues dans une autre espèce (par exemple $R2$ est orthologue à $S2$ et $S3$ et $H1$ est orthologue à n'importe quel Ri et Si).

Enfin, il convient de remarquer que les définitions précédentes sont de nature essentiellement phylogénétique et ne font aucune hypothèse quant à la similarité des séquences ou la fonction des gènes concernés (même si de telles hypothèses doivent parfois être posées pour parvenir à reconstruire l'histoire évolutive d'un gène ou d'une famille de gènes). Ainsi, l'emploi du terme homologue pour indiquer que deux gènes présentent des séquences similaires doit être considéré comme une impropriété. En revanche, on peut raisonnablement faire l'hypothèse que si les séquences sont similaires il est probable que les gènes soient homologues. De même, le fait que deux gènes orthologues devraient présenter la même fonction doit être, le plus souvent, considéré comme une hypothèse de travail.

Suivant les approches, les objectifs seront différents : certaines méthodes cherchent à faire de la génomique comparative, d'autres recherchent les synténies, ou encore des scénarios de réarrangements chromosomiques.

Prenons quelques instants pour rappeler ce que nous entendons par "synténie". Dans le domaine des eucaryotes, on dit de deux gènes qu'ils sont en synténie s'ils sont portés par le même chromosome. Cette définition n'est utile que dans le cadre d'organismes multichromosomiques (typiquement des eucaryotes). La plupart des bactéries n'ayant qu'un chromosome, tous les gènes sont par définition en synténie. Par abus de langage, on parle de "synténies bactériennes" ou "microsynténies" lorsque les gènes restent, au cours de l'évolution, proches sur le chromosome. De manière plus pratique, la recherche de "blocs de synténie" entre n espèces se ramène donc à celle des ensembles de gènes homologues dont la proximité sur les chromosomes est conservée.

Nous avons pris le parti d'exclure les méthodes qui opèrent sur une représentation à l'échelle du nucléotide, mais qui reconstruisent immédiatement des zones strictement identiques ensuite utilisées pour retrouver des blocs de synténie. Parmi ces méthodes, que nous ne détaillerons donc pas, nous pouvons citer notamment Mauve [DMBP04], Mummer [DKF+99], Glass [BPM+00], SSAHA [NCM01], AVID [BDP03], WABA [KZ00] ou encore LAGAN [BDC+03]. Bien que certaines de ces méthodes puissent être transposées à l'échelle de gènes, il ne s'agit pas de leur objectif premier.

Pour des raisons analogues, nous avons également exclu de cet état de l'art les travaux portant sur l'alignement de marqueurs génétiques, tels que Lineup [HMGB03] ou MCPaGe [LHXH08], qui présentent des spécificités (notamment en terme de résolution) rendant difficile la comparaison avec les méthodes travaillant sur des génomes complètement séquencés que nous présentons.

À la fin de cette section, nous rassemblons les différentes approches dans une même table : la table 2.3.

2.4.1 GRIMM-Synteny et Cinteny : méthodes basées sur le tri par inversions

Le travail fondateur de Hannenhali et Pevzner ([HP99], [Ber01]) sur le tri par inversion est à la base de nombreuses méthodes de reconstruction de scénarios évolutifs basés sur les réarrangements chromosomiques. Rappelons rapidement le problème théorique : étant données deux permutations sur un même ensemble, l'objectif est de calculer le nombre minimum d'inversions permettant de passer d'une permutation à l'autre.

Parmi ces méthodes, les plus importantes dans le contexte de la recherche de synténies sont GRIMM [Tes02] et Cinteny [SM07].

Dans ces approches, les génomes sont donc représentés sous la forme de permutations signées, c'est-à-dire d'un ou de plusieurs vecteurs de nombres positifs ou négatifs, chacun représentant un gène, le signe représentant son orientation sur le chromosome. L'objectif est de calculer une distance d'inversion, c'est-à-dire pour deux génomes G_1 et G_2, calculer le nombre minimal d'inversions permettant de transformer G_1 en G_2.

Bien que l'objectif initial soit plutôt de calculer des scénarios évolutifs de réarrangements chromosomiques, la méthode GRIMM-Synteny peut être utilisée pour calculer des blocs de synténie.

En effet, afin de séparer les micro-réarrangements des macro-réarrangements et de pouvoir ensuite appliquer de façon efficace l'algorithme de Hannenhali et Pevzner, les auteurs proposent une stratégie simple de génération de blocs de synténie.

L'idée est simple : deux paires de gènes homologues sont reliées si la distance de Manhattan [10] qui les sépare est inférieure à un seuil fixé, et les blocs de synténie

10. Soit (g_1, g_2) et (g'_1, g'_2) deux paires de gènes homologues, notons r_i le rang de g_i sur son génome. La distance de Manhattan entre (g_1, g_2) et (g'_1, g'_2) est $d_M((g_1, g_2), (g'_1, g'_2)) = |r'_1 - r_1| + |r'_2 - r_2|$.

sont définis comme les composantes connexes (de taille supérieure à un seuil fixé) dans le multigraphe d'alignement ainsi construit.

Cinteny utilise une approche très similaire, mais met en œuvre une structure de données *ad-hoc* appelée TST (pour ternary search tree) permettant une implémentation plus aisée. Cette structure est un arbre d'indexation des groupes d'homologie dans lequel les feuilles contiennent également des pointeurs vers les gènes voisins.

Ces approches restent néanmoins limitées à deux génomes.

2.4.2 Uno et Tagiura, Didier, Hebert et Stoye

Comme dans les approches précédentes, les génomes sont représentés par des permutations (signées ou non) d'entiers. Dans l'article fondateur, celui de Uno et Tagiura [UY00], le problème est formalisé comme la recherche des intervalles en commun entre deux génomes, c'est-à-dire des paires d'intervalles formés du même ensemble d'éléments, un provenant de chacune des deux permutations.

Les auteurs présentent un algorithme optimal en temps $O(n + K)$ et espace $O(n)$ permettant de trouver tous les $K \leq \mathrm{C}_n^2$ intervalles en commun entre deux permutations de n éléments.

Hebert et Stoye [HS01] proposent ensuite une généralisation à la recherche d'intervalles en commun à k génomes, l'algorithme donné est optimal en temps $O(kn + K)$ et espace $O(n)$.

Didier [Did03] revient à l'alignement de deux génomes, mais étend le problème au cas d'une relation many-to-many (on sort dès lors du domaine proprement dit des permutations) : en se limitant à n^2 intervalles en commun au maximum entre deux séquences de n éléments avec duplications, l'auteur parvient à exhiber un algorithme en $O(n^2 log(n))$.

Ces approches restent néanmoins limitées par la représentation sous forme de permutation, qui présuppose que la relation d'homologie est une relation d'équivalence et qui ne permet pas de gérer les évènements de délétion / insertion (gaps).

2.4.3 TEAMS : GeneTeams, HomologyTeams, DomainTeams, Mcgs, MCPage, MCMuSeC

Bergeron *et al* [BCR02] ont proposé en 2002 une nouvelle définition formelle adaptée à la recherche de synténies sur $n \geq 2$ génomes.
Notons tout de suite que cette définition fait l'hypothèse que la relation d'homologie entre les gènes est transitive (il s'agit donc d'une relation d'équivalence). Les gènes sont donc réécrits comme les étiquettes des classes de cette relation.

Informellement les *gene teams* sont des ensembles de gènes tels que sur chacun des génomes aucune paire de gènes de la team n'est éloignée de plus de δ gènes.

L'article initial de Bergeron *et al* fournit un algorithme Diviser pour Régner, qui permet de retrouver tous les *gene teams* formés par n gènes sur m chromosomes linéaires sous l'hypothèse que la relation d'homologie est bijective (c'est-à-dire qu'une classe n'apparaît qu'une fois sur chaque chromosome), les auteurs proposent également une optimisation de cet algorithme basée sur une approche de partitionnement similaire à l'algorithme de Hopcroft.

He et Goldwasser en 2004 [HG05] ont ensuite étendu la notion de *gene teams* aux *homology teams* en autorisant les gènes à être présents en plusieurs exemplaires sur chaque génome. Il faut malgré tout se limiter à deux génomes pour conserver une complexité polynomiale.

Pasek *et al* en 2005 [PBR+05] repartent de cet algorithme mais proposent de découper chaque gène en domaines protéiques (par exemple en utilisant PFAM [FTM+08]). Ils montrent que bien que théoriquement exponentielle (en nombre de solutions), cette approche est en pratique très efficace et permet de traiter plus de deux génomes.

Il est à noter que le découpage en domaines permet naturellement de représenter la relation de correspondance par une relation d'équivalence, ce qui ne serait pas le cas au niveau des gènes.

L'algorithme Mcgs [KCY05], présenté par Kim *et al* la même année, relâche la contrainte de proximité sur un certain nombre de génomes en conservant une complexité raisonnable grâce à une heuristique.

Ling *et al* en 2008 a fourni dans MCPage [LHXH08] une extension supplémentaire du modèle, permettant de traiter aussi les unités chevauchantes (par exemple des "sequence anchors" ou marqueurs).

Enfin en 2009, Ling *et al* [LHX09] proposent une amélioration de l'algorithme Mcgs, combinant l'heuristique de la première version avec une méthode de filtrage.

2.4.4 DAGchainer

Il s'agit ici [HDWS04] d'une méthode qui recherche des chaînes de gènes dont l'ordre est strictement conservé dans deux chromosomes donnés en entrée (pas d'inversion).

L'idée est de construire un *DAG* (Directed Acyclic Graph) dont les nœuds sont des paires de gènes homologues (données en entrée) et dont les arêtes relient les paires qui peuvent être chaînées (deux paires (i, j) et (k, l) peuvent être chaînées si $i < j$ et $k < l$), puis d'extraire de ce graphe les chemins de plus fort score en utilisant une approche par programmation dynamique.

Comme pour l'alignement de séquences, le score fait intervenir une pénalité de substitution (basée sur la similarité de séquence) et une pénalité de gap.

2.4.5 FISH

FISH [CCV03] est un acronyme pour Fast Identification of Segmental Homology. Ce programme est conçu pour retrouver des blocs de synténie entre deux chromosomes en présence de nombreux réarrangements.

L'algorithme construit une matrice booléenne, contenant des '1' lorsque que les gènes correspondants sont homologues (la relation d'homologie est supposée donnée en entrée), et des '0' sinon. Les blocs de synténie apparaissent donc sous la forme de cases '1' proches dans la matrice qui suivent approximativement une diagonale. Ces blocs sont appelés "k-clumps".
Les auteurs utilisent un algorithme classique de programmation dynamique pour déterminer tous les k-clumps de taille maximale (la taille d'un k-clump est simplement égale au nombre de 1 qu'il contient) : la figure 2.11 donne l'idée générale de construction des k-clumps.

Les auteurs définissent également un modèle statistique permettant de calculer le nombre attendu de clumps d'une taille donnée et donc d'associer un score (une P-value) aux clumps.

A priori, l'utilisation de la programmation dynamique autorise les gaps mais pas les inversions. L'introduction d'un voisinage autour de chaque gène (clumps) permet néanmoins la prise en compte de petites inversions locales.

2.4.6 ADHoRe, i-ADHoRe

ADHoRe [VSS+02] (Automatic Detection of Homologuous Regions) recherche

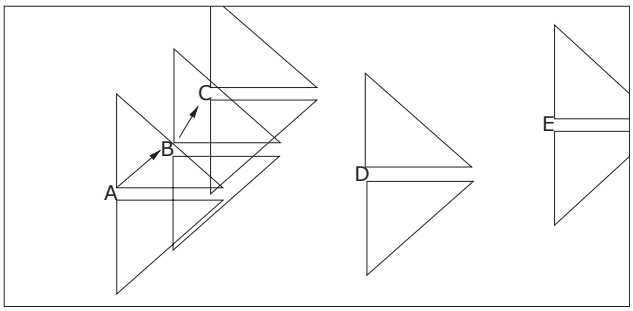


FIGURE 2.11 – (A, B, C) est un exemple de 3-clump : le voisinage de A pour la distance de Manhattan contient B, le voisinage de B contient C, alors que les voisinages de D et E ne contiennent pas de points. Figure extraite de FISH [CCV03].

des zones génomiques dont l'ordre des gènes est conservé entre deux génomes. Comme dans FISH, la relation d'homologie est donnée au départ et représentée par une matrice booléenne. L'algorithme suit trois étapes :

1. preprocessing : fusion des duplications en tandem et élimination des couples de gènes trop éloignés (pour accélérer la procédure)
2. clustering des couples de gènes (au moins 3 couples de gènes par cluster) en utilisant l'indice de dissimilarité suivant :
   $ds((g_1, g_2), (g'_1, g'_2)) = 2max(|r_1 - r'_1|, |r_2 - r'_2|) - min(|r_1 - r'_1|, |r_2 - r'_2|)$ où r est le rang du gène g. Cet indice permettra donc de regrouper les gènes proches sur le chromosome.
3. postprocessing : les clusters sont ensuite évalués statistiquement et regroupés suivant une heuristique dont on trouvera le détail dans [VSS+02].

En 2004, les auteurs ont proposé une extension de cette méthode à l'alignement de multiple génomes, I-ADHORE [SVSVdP04].

I-ADHORE commence par utiliser ADHORE sur toutes les paires de génomes 2 à 2. Cela fournit une première série de blocs appelés "multiplicons de niveau 2". Cette série initialise un ensemble $\mathcal{M}$ de multiplicons.

L'algorithme va ensuite essayer d'ajouter des segments supplémentaires à ces premiers multiplicons. Pour cela, il commence par le plus grand multiplicon précédemment trouvé, il le représente sous forme d'un "profil" (c'est une suite de positions alignées, dans le cas de deux génomes ces positions sont occupées soit par deux gènes homologues, soit par un seul gène et un gap). Ce profil est ensuite confronté à l'ensemble des génomes - y compris ceux dont il est issu - en utilisant

une variante de ADHoRe. Cela implique qu'il faut maintenant aligner un profil avec des gènes. On considère qu'il y a un match entre un gène et une position donnée d'un profil lorsque cette position contient au moins un gène qui est un match.

Cela implique notamment qu'à la fin de l'exécution, tous les gènes d'une position donnée d'un multiplicon de niveau k forment une composante connexe de la relation d'homologie.

Les résultats de cet alignement sont des multiplicons de niveau 3 (il peut s'agir d'extensions d'une partie seulement du multiplicon initial). Ces nouveaux multiplicons sont ensuite remis dans l'ensemble $\mathcal{M}$. L'algorithme itère au niveau 3 (c'est-à-dire reprend le plus grand multiplicon de niveau 3), puis lorsque tous les multiplicons à un niveau donné sont épuisés, l'algorithme itère au niveau inférieur.

Cet algorithme a été amélioré en 2007 : dans la nouvelle version - i-ADHoRe 2.0 [SJSVdP08]- les profils sont remis en question à chaque étape, un algorithme glouton recalcule l'alignement de tous les segments, ce qui permet de corriger des décisions erronées qui auraient pu être prises au début de l'exécution.

2.4.7 Blockfinder, Syntenator, Cyntenator

BlockFinder [RD07] recherche des synténies sur $n \geq 2$ génomes. Il commence par calculer de façon heuristique des cliques de la relation de similarité de séquence.

Il construit ensuite un $DAG\ G = (V, E)$ dont les nœuds sont ces cliques (empilements) et dans lequel un arc relie un nœud $(g_1, g_2, \ldots g_n)$ à un nœud $(g'_1, g'_2, \ldots g'_n)$ s'il existe m génomes sur lesquels g_i précède g'_i (ou bien le suit, si l'orientation est différente de l'orientation initiale). [11]

Dans ce "clique order graph", les blocs de synténie sont enfin définis comme les chemins les plus longs.

Une limitation importante de cette approche réside dans l'utilisation de cliques de la relation d'homologie. Les associations qui ne forment pas une clique complète ne sont pas prises en compte.

Afin de relâcher cette contrainte forte, les auteurs proposent une seconde approche, Syntenator [RD08], qui combine une représentation des génomes (puis des alignements eux-mêmes) comme des graphes partiellement ordonnés (POGs)

11. Plus formellement, en notant $ori(g)$ l'orientation d'un gène g et $\prec$ la relation de précédence sur un génome : $((g_1, g_2, \ldots g_n), (g'_1, g'_2, \ldots g'_n)) \in E \Leftrightarrow \exists \{i_1, \ldots i_m\} \subseteq \{1, \ldots n\}$ avec $m > 1$ tels que $\forall i, j \in \{i_1, \ldots i_m\}\ g_i \prec g'_i \Rightarrow (g_j \prec g'_j$ et $ori(g_i) = ori(g_j)$ et $ori(g'_i) = ori(g'_j))$ ou $(g'_j \prec g_j$ et $ori(g_i) \neq ori(g_j)$ et $ori(g'_i) \neq ori(g'_j))$.

et un alignement de ces graphes par programmation dynamique.

Etant donnés deux génomes (c'est-à-dire deux suites de gènes), la procédure commence par rechercher les alignements locaux optimaux en utilisant un algorithme de programmation dynamique (du type Smith Waterman [SW81]). La fonction de score est bien sûr basée sur la similarité des séquences. Ces alignements sont resumés sous la forme d'un graphe partiellement ordonné (POG) dont les nœuds sont soit un couple de gènes (orthologues), soit un seul gène de l'un ou l'autre génome (insertion/délétion), un arc entre deux nœuds traduit que sur l'un au moins des génomes les gènes se suivent [12]. La figure 2.12 donne une vue générale de la construction du POG pour 2 génomes. Ce POG peut ensuite être lui-même aligné à un nouveau génome ou à un autre POG par la même procédure. Il est à noter que l'alignement de deux POG peut produire un graphe contenant des cycles (cf [RD08]), dans ce cas, les auteurs se ramènent à un graphe sans cycles par inversion de certains arcs.

En 2010, les auteurs ont finalement proposé une nouvelle version de cette méthode, appelée Cyntenator [RD10], qui exploite la phylogénie dans plusieurs de ses aspects. L'ordonnancement des génomes à aligner est maintenant réalisé suivant un arbre phylogénétique, et les pénalités de gap ou mismatch prennent en compte la distance phylogénétique entre les espèces. Le but est de pénaliser plus fortement la perte d'une paire de gènes homologues lorsque les espèces sont proches.

2.4.8 OrthoCluster

OrthoCluster [ZPV+08] adopte une approche différente des précédentes, inspirée de la "fouille de données". L'idée générale est de comparer des ensembles de gènes S_1 et S_2 contigus sur deux génomes (G_1 et G_2) suivant une mesure de similarité entre ensembles $s(S_1, S_2) = min(\frac{|S_1/S_2|}{|S_1|}, \frac{|S_2/S_1|}{|S_2|})$ où $|S_i/S_j|$ est le nombre de gènes de S_i qui présentent au moins un orthologue dans S_j

Pour ce faire, l'approche proposée consiste, pour chaque gène g_1 de G_1, à considérer l'ensemble S_1 des gènes adjacents à une distance maximale d_{max} donnée (fenêtre glissante de taille $2d_{max} + 1$). S_1 est ensuite comparé à tous les ensembles S_2 associés à toutes les fenêtres glissantes sur G_2.

Cette approche brutale est en réalité implémentée de façon efficace en utilisant une structure de données particulière ("set enumeration tree") permettant de retrouver rapidement tous les ensembles contenant un gène donné. Par ailleurs,

12. En pratique, les alignement doivent être réalisés suivant les deux orientations possibles des chromosomes.

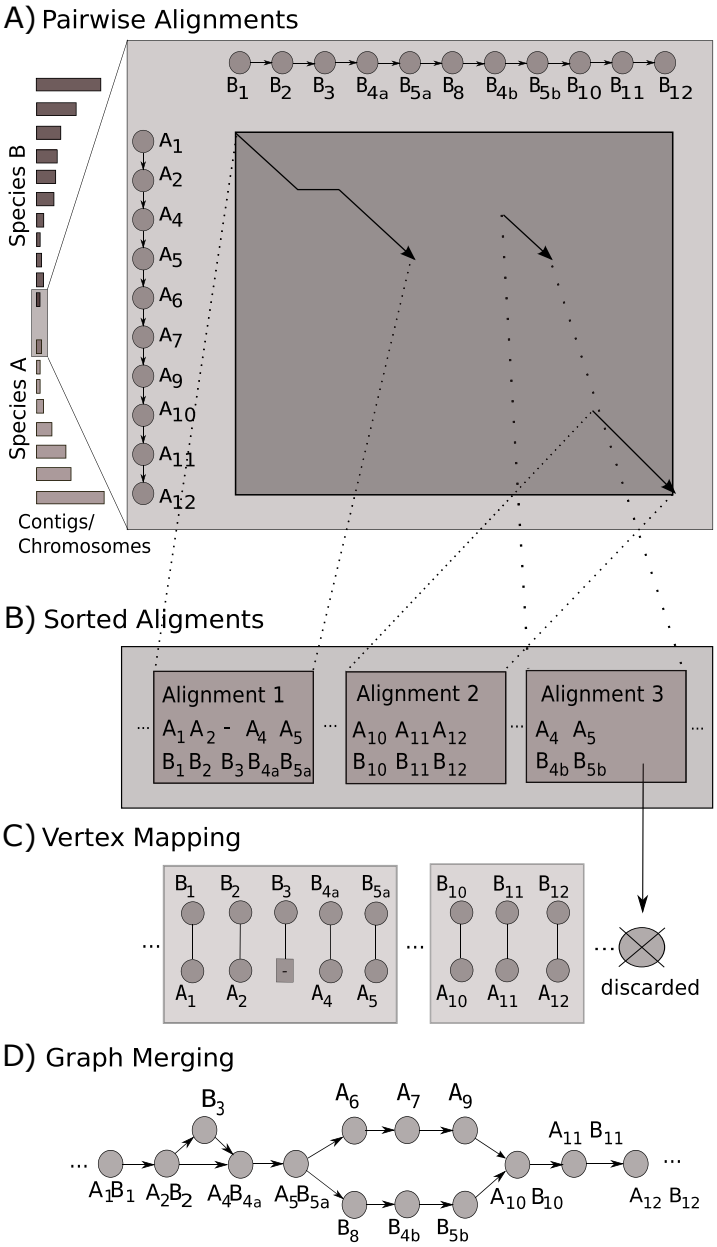


FIGURE 2.12 – Exemple de construction de POG pour deux génomes. L'algorithme calcule d'abord des alignements locaux optimaux, les trie de façon à éliminer les alignements incompatibles, puis construit le graphe partiellement ordonné associé, en fusionnant les nœuds alignés (par exemple (A_1, B_1)) et en conservant les nœuds isolés (par exemple B_3). Figure extraite de SYNTENATOR [RD08].

plusieurs conditions d'élagage de l'espace de recherche sont proposées.

ORTHOCLUSTER permet donc de prendre en compte des inversions et des gaps puisqu'il travaille sur des ensembles. Il peut également prendre en compte l'orientation des gènes (en modifiant la relation de similarité).

L'algorithme semble assez efficace en pratique pour 2 génomes. L'extension à plus de 2 génomes, bien que qualifiée de "straightforward" par les auteurs, nous semble néanmoins délicate, à la fois en terme d'efficacité mais surtout de redondance des résultats.

2.4.9 ʼUber-Opérons

L'approche décrite par Che *et al* [CLM+06] a pour objectif d'identifier les ʼuber-opérons présents sur un génome procaryote G. Un opéron est informellement un groupe de génes adjacents co-transcrits qui coopèrent généralement dans une même fonction physiologique. Un ʼuber-opéron est défini comme un groupe d'opérons sur un même génome G qui sont associés fonctionnellement ou évolutivement. Ce regroupement fonctionnel ou évolutif est détecté en utilisant les opérons d'un second génome G' (dit de référence).

Cette méthode sort donc *stricto sensu* de notre cadre de travail : il ne s'agit plus d'aligner localement des données génomiques, mais de regrouper des alignements locaux déjà identifiés. Nous faisons le choix de la présenter rapidement parce qu'elle nous semble malgré tout suffisamment intéressante dans le domaine pour devoir apparaître dans cet état de l'art.

Les données utilisées en entrée du programme sont :
- les ensembles d'opérons sur les deux génomes, obtenus par des programmes de prédiction spécialisés [PHAA05][CSD+04],
- une relation de correspondance entre gènes des deux génomes (obtenue par seuillage sur l'E-value de BLAST).

A partir de ces données, les auteurs vont chercher un appariement ("matching") entre les deux génomes qui maximise un certain score.

Ce score est défini de la façon suivante : pour un "matching" donné, les auteurs considèrent le multigraphe M dont les nœuds sont les opérons et les arêtes sont données par le matching des gènes. Le score est le nombre de composantes connexes de ce multigraphe. Intuitivement, trouver un matching qui maximise ce score va revenir à partitionner l'ensemble des opérons des deux génomes en des groupes d'opérons fortement connectés.

Le problème étant NP-complet, Che *et al* ont recours à une heuristique glou-

Méthode	granularité	nb de génomes	correspondance	opérations d'édition			algorithmique	objectif	référence
				INV / TRANS	DUPL	DEL			
GRIMM	gènes	multiple	one-to-one	oui	oui	oui	minimiser la distance d'inversion	scénarios évolutifs	[Tes02]
Cinteny	gènes	multiple	many-to-many	oui	oui	oui	idem	scénarios évolutifs	[SM07]
Uno et Tagiura	gènes	deux	one-to-one	oui	non	non	recherche d'intervalles en commun	synténies	[UY00]
Heber et Stoye	gènes	multiple	one-to-one	oui	non	non	recherche d'intervalles en commun	synténies	[HS01]
Didier	gènes	multiple	many-to-many	oui	oui	non	recherche d'intervalles en commun	synténies	[Did03]
Gene Teams	gènes	multiple	équivalence	oui	non	oui	diviser pour régner	synténies	[BCR02]
Homology Teams	gènes	deux	équivalence	oui	oui	oui	idem	synténies	[HG05]
Domain Teams	domaines	multiple	équivalence	oui	oui	oui	idem	synténies	[PBR+05]
Mcgs	gènes	multiple	équivalence	oui	oui	oui	idem	synténies	[KCY05]
MCPaGe	gènes	deux	équivalence	oui	oui	oui	idem	synténies	[LHXH08]
MCMuSeC	gènes	multiple	équivalence	oui	oui	oui	idem	synténies	[LHX09]
DAGchainer	gènes	deux	many-to-many	non	non	oui	programmation dynamique	synténies	[HDWS04]
FISH	gènes	deux	many-to-many	locales	non	oui	programmation dynamique	synténies	[CCV03]
ADHoRe	gènes	deux	many-to-many	non	tandem	oui	clustering	régions homologues	[VSS+02]
i-ADHoRe	gènes	multiple	many-to-many	non	tandem	oui	heuristique gloutonne	régions homologues	[SVSVdP04]
BlockFinder	gènes	multiple	many-to-many	non	oui	oui	chemins max dans un DAG	synténies	[RD07]
Syntenator	gènes	multiple	many-to-many	non	oui	oui	programmation dynamique sur POG	synténies	[RD08]
Cyntenator	gènes	multiple	many-to-many	non	oui	oui	idem + phylogénie	synténies	[RD10]
OrthoCluster	gènes	multiple	many-to-many	oui	oui	oui	approche brutale avec fenêtre glissante	synténies	[ZPV+08]

Table 2.3 – Méthodes d'alignement de génomes. La colonne *granularité* précise le type de nœuds alignés : gènes ou domaines protéiques. La colonne *nb de génomes* indique si l'approche aligne seulement deux génomes ou s'il s'agit d'alignement multiple. *correspondance* donne le type de relation utilisée, *one-to-one* signifie qu'elle est bijective, *équivalence* signifie que c'est une relation d'équivalence et enfin *many-to-many* correspond à une relation quelconque. Dans les colonnes récapitulant les *opérations d'édition* (inversion, translocation, duplication et délétion), un *oui* signifie que l'approche parvient à retrouver un alignement en présence de ce type de perturbation, non signifie qu'elle n'y parviendra pas. Dans les dernières colonnes nous donnons brièvement un aperçu de la méthode et de son objectif.

tonne :

1. sélectionner la paire d'opérons qui réalise le matching le plus important,
2. les fusionner en un même groupe d'opérons, et recalculer le matching optimal,
3. recommencer tant qu'il reste des opérons.

Chacun des groupes d'opérons rendus en résultat par l'algorithme sera finalement interprété comme un ´uber-opéron sur le génome G et un ´uber-opéron sur le génome G'.

2.5 Approches à données hétérogènes

Nous allons présenter dans cette section les approches développées par Ogata *et al* et par Boyer *et al*. Il s'agit d'approches génériques qui peuvent s'appliquer à plusieurs types de données simultanément et qui sont à la base du travail développé dans cette thèse

2.5.1 Ogata

L'intérêt de l'article d'Ogata [OFGK00] est avant tout historique, puisqu'il s'agit d'une des premières tentatives de formalisation de la question générale de l'alignement de réseaux biologiques.

La méthode développée vise initialement la recherche de clusters d'enzymes impliquées dans les mêmes voies métaboliques et codées par des gènes colocalisés sur le génome (par exemple des opérons).
Pour retrouver ces clusters (appelés $FREC$ pour Functionally Related Enzyme Clusters), l'approche dispose d'un réseau métabolique représenté par un graphe de réactions $G = (V, E)$, et d'un graphe représentant la disposition des gènes sur le génome, $G' = (V', E')$, mais la méthode est présentée dans le cadre plus général de deux graphes quelconques.

Une réaction est mise en correspondance avec un gène sur la base de l'égalité des EC numbers. Les empilements sont donc formés d'une réaction et d'un gène.

Etant donnés deux empilements $v_1 = (R_1, g_1)$ et $v_2 = (R_2, g_2)$ avec R_1, $R_2 \in V$ et g_1, $g_2 \in V'$, les auteurs définissent deux distances $d(v_1, v_2)$ et $d'(v_1, v_2)$, une dans chaque graphe, correspondant respectivement à la longueur du chemin le plus court entre les réactions R_1 et R_2 dans G et à la longueur du chemin le plus court entre les gènes g_1 et g_2 dans G'.

L'algorithme va procéder ensuite à un clustering des empilements en autorisant des gaps. On initialise les clusters à des singletons d'empilement, puis on

les fusionne progressivement suivant l'heuristique suivante : deux clusters C_1, C_2 seront fusionnés s'il existe une paire d'empilements $u_1 \in C_1$, $u_2 \in C_2$ telle que $d(u_1, u_2) \leq gap$ et une paire d'empilements $v_1 \in C_1$, $v_2 \in C_2$ telle que $d'(v_1, v_2) \leq gap'$.

La figure 2.13 donne une représentation schématique du fonctionnement de l'algorithme.

Cet article d'Ogata jette les bases de l'alignement de réseaux. Ainsi, les auteurs identifient bien la notion d'empilement (limitée ici à deux nœuds) ainsi que la contrainte de localité (adjacence) sur chacun des réseaux primaires.

En revanche, il manque encore un ingrédient important : le multigraphe d'alignement, qui sera introduit dans l'article de Boyer *et al* [BML+05] et qui permettra de définir précisément (et non de manière procédurale) les alignements recherchés et, surtout, de remplacer l'étape heuristique de clustering par un algorithme exact.

2.5.2 C_3Part

L'approche de Boyer *et al* [BML+05] reprend l'idée d'Ogata d'une recherche de voisinage communs à plusieurs graphes, et la formalise comme le problème général de la recherche de composantes connexes communes à plusieurs graphes.

L'article introduit la notion d'une représentation fusionnée des réseaux primaires : le multigraphe d'alignement, et définit les alignements locaux recherchés comme les composantes connexes communes (CCC) de ce multigraphe.

Les auteurs remarquent que les composantes connexes communes forment une partition du multigraphe d'alignement, et proposent un algorithme exact de partitionnement de ce graphe. En pratique, l'algorithme (appelé C_3Part) procède par raffinement, la procédure est la suivante :

1. commencer avec une seule classe contenant tous les nœuds,
2. calculer $\bigcap_i (CC_i)$, l'intersection de toutes les composantes connexes pour tous les ensembles d'arêtes, ce qui aboutit à une nouvelle partition dans laquelle chaque classe est une CCC potentielle,
3. itérer l'étape 2 sur chacune des classes jusqu'à ce que la partition soit stable.

L'apport important par rapport au travail d'Ogata est que l'algorithme est exact et non plus heuristique. Les auteurs donnent d'ailleurs dans l'article quelques exemples de graphes qui sont manqués par l'heuristique d'Ogata.

C'est sur cette méthode que notre travail de thèse va s'appuyer, nous allons dans un premier temps présenter une définition formelle du problème de l'aligne-

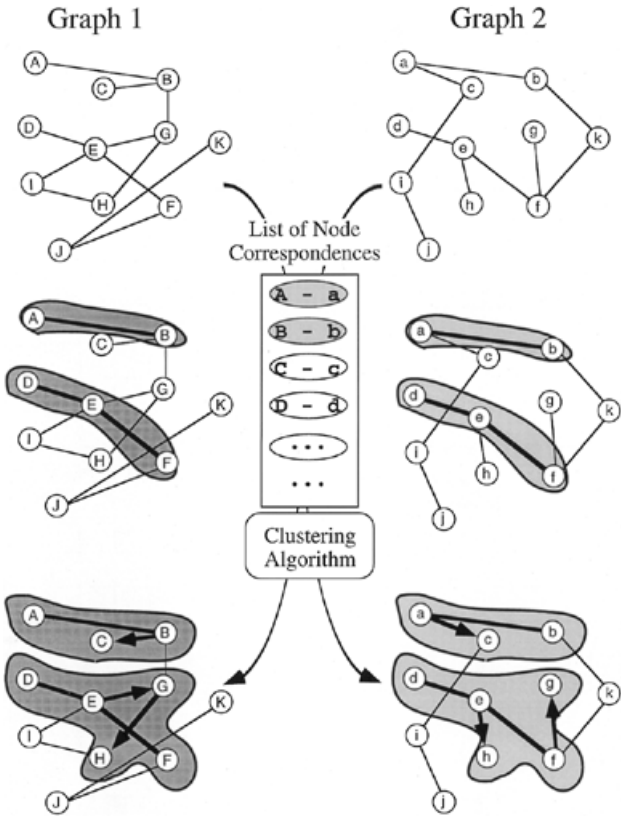


FIGURE 2.13 – Représentation schématique de l'algorithme d'alignement local développée dans [OFGK00]. La méthode prend en entrée une liste de correspondances entre réactions et gènes, chacun des empilements correspondant est initialement dans un cluster séparé. Au cours de l'algorithme, deux clusters sont fusionnés s'ils sont connectés sur chacun des deux graphes. Sur la figure nous constatons ainsi que le premier clustering fusionne $\{(A, a)\}$ et $\{(B, b)\}$, puis que le cluster obtenu est ensuite fusionné avec $\{(C, c)\}$. Le cluster résultant, $\{(A, a), (B, b), (C, c)\}$ ne peut plus être fusionné avec aucun des clusters restants ($\{(I, i)\}$, $\{(J, j)\}$, $\{(K, k)\}$ et $\{(D, d), (E, e), (F, f), (G, g), (H, h)\}$), il s'agit donc d'un $FREC$.

ment local de réseaux biologiques, puis nous donnerons une amélioration algorithmique de C_3Part qui permettra d'utiliser l'algorithme dans le cas d'un grand nombre de réseaux, et enfin nous étendrons à la fois le formalisme et l'algorithme au cas plus général de l'alignement local partiel : au lieu d'imposer un empilement systématique de n nœuds, nous autoriserons un certain nombre de nœuds manquants, pour s'approcher de la réalité biologique et tenir compte d'erreurs.

Chapitre 3

Alignement local multiple

Dans cette partie, nous allons proposer une formalisation du problème général de l'alignement local multiple de n réseaux.

Nous commencerons par donner la représentation des réseaux et de la relation de correspondance sous la forme d'un **graphe de données stratifié**. Nous définirons ensuite les alignements locaux comme des sous-graphes maximaux du graphe de données stratifié vérifiant deux contraintes :

- contrainte 'verticale' de correspondance entre n nœuds : **contrainte d'empilement**
- contrainte 'horizontale' portant sur les réseaux primaires : **contrainte de localité**.

Informellement, la contrainte d'empilement garantit que chaque nœud de l'alignement est bien aligné avec $(n-1)$ autres nœuds, pris dans chacun des autres réseaux. La contrainte de localité, comme son nom l'indique, assure que l'alignement est bien local : sur chacun des réseaux la partie alignée devra être connexe.

Enfin, nous expliciterons la représentation compactée du graphe de données stratifié sous la forme du **multigraphe d'alignement** que nous avons déjà évoqué dans la Section 2. Nous donnerons la définition des **connectons** dans ce multigraphe, et nous montrerons qu'ils correspondent exactement aux alignements locaux précédemment définis.

3.1 Graphe de données stratifié

Nous avons besoin d'un formalisme suffisamment général pour recouvrir chacun des différents cas trouvés dans la littérature, que ce soient les alignements multiples

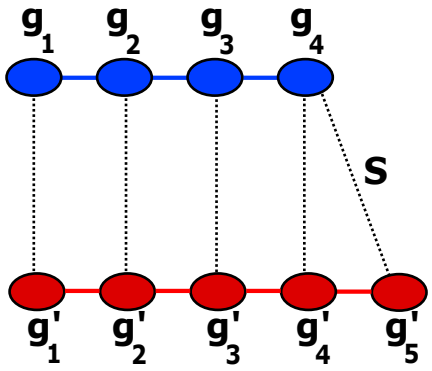


FIGURE 3.1 – Exemple simple de graphe de données stratifié sur deux graphes primaires représentant des génomes. Les nœuds et arêtes des graphes primaires sont représentés en couleurs, bleu pour le premier graphe et rouge pour le second. La relation de correspondance S est représentée par les arêtes pointillées.

de génomes, PPI, réseaux métaboliques, ou les alignements à données hétérogènes, c'est-à-dire, par exemple, l'alignement de génomes et d'un réseau métabolique.
Le formalisme que nous allons détailler ici est celui du *graphe de données stratifié*. Il a été inspiré entre autres par les travaux de Kalaev *et al* en 2008.

Informellement, un graphe de données stratifié va être formé de :
- n graphes primaires non-orientés (chacun formant une strate),
- une relation de correspondance S entre nœuds de différents graphes primaires.

La figure 3.1 présente un exemple de graphe de données stratifié avec les graphes primaires associés et la relation de correspondance.

Plus formellement, nous avons :

Définition 3.1. *Etant donné un ensemble de n graphes $G_i = (V_i, E_i)$, $i \in [|1, n|]$ et une relation de correspondance S entre éléments d'ensembles distincts V_i, le* graphe de données stratifié *est le graphe $D = (V, E)$ où*
- $V = \bigcup_i V_i$
- $E = E_P \cup E_S = (\bigcup_i E_i) \cup \{(u, v) \in V_i \times V_{j \neq i} / u\ S\ v\}$

Il y a donc deux types distincts d'arêtes dans E : les arêtes de E_P (P comme "primaires") correspondent aux ensembles initiaux $\bigcup_i E_i$ (nous y ferons référence dans la suite sous le nom *arêtes intra-strate*) alors que les arêtes de E_S connectent des noeuds de différentes strates $\{(u, v) \in V_i \times V_{j \neq i} / u\ S\ v\}$(nous y ferons référence dans la suite sous le nom *arêtes inter-strates*). Nous donnons dans la Figure 3.2

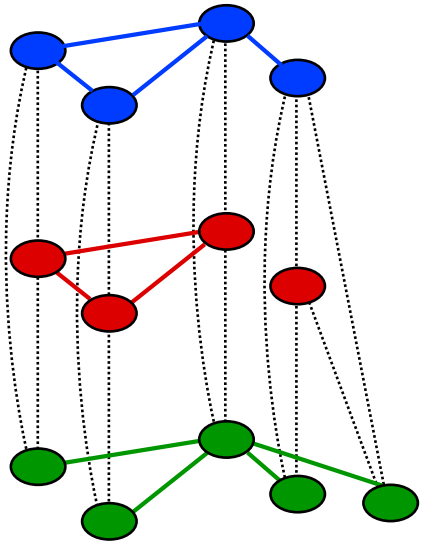


FIGURE 3.2 – Exemple de graphe de données stratifié possédant trois strates, correspondant à trois réseaux. Les arêtes intra-strates sont représentées par des traits pleins en couleur alors que les arêtes inter-strates - c'est-à-dire la relation de correspondance S - sont représentées par des lignes pointillées.

un second exemple de graphe de données stratifié à trois strates.

Les arêtes intra-strates - E_P - traduisent le voisinage, c'est-à-dire la localité sur un réseau primaire, alors que les arêtes inter-strates - E_S - rendent compte des correspondances entre nœuds de différents réseaux.

Intuitivement, les alignements locaux sont des sous-graphes de ce graphe de données stratifié satisfaisant certaines contraintes. Dans la Figure 3.3, nous donnons un exemple d'alignement local.

3.2 Relation de correspondance multiple et contrainte d'empilement

La première contrainte qui intervient dans la définition d'un alignement local est la contrainte d'empilement.

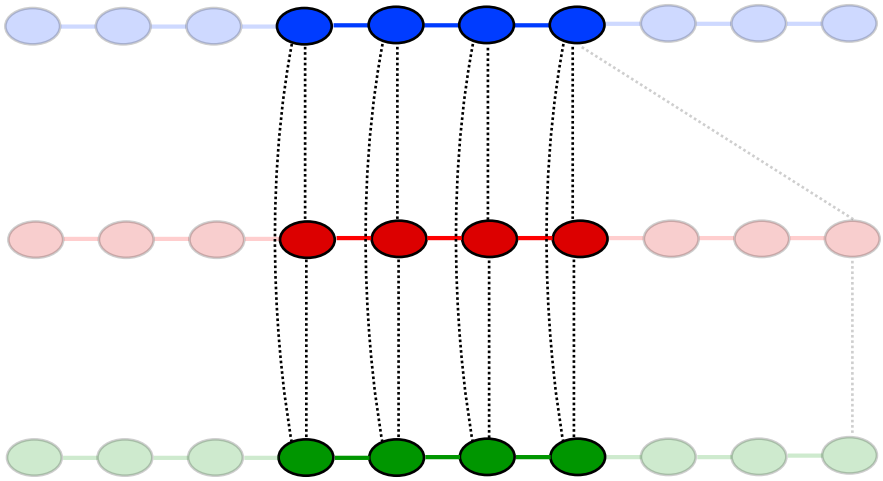


FIGURE 3.3 – Exemple d'alignement local. Les nœuds et arêtes de l'alignement apparaissent en foncé.

Cette contrainte s'exprime dans le graphe de données par les arêtes inter-strates, c'est-à-dire les arêtes de l'ensemble $E_S = \{(u, v) \in V_i \times V_{j \neq i} / u \; S \; v\}$.

En pratique, nous avons donc besoin d'une relation de correspondance binaire S entre nœuds de deux réseaux différents. Voici quelques exemples de choix suivant les différents types de réseaux primaires à aligner.

- génome / génome : deux gènes se correspondent si leur similarité de séquence est supérieure à un certain seuil,
- PPI / PPI : deux protéines se correspondent si leur similarité de séquence est supérieure à un certain seuil,
- métabolisme / métabolisme : deux réactions se correspondent si les k premiers chiffres de leurs numéros EC sont identiques,
- métabolisme / PPI : une réaction correspond à une protéine si la protéine catalyse la réaction,
- métabolisme / génome : une réaction correspond à un gène si le gène code pour une enzyme qui catalyse la réaction,
- PPI / génome : une protéine correspond à un gène s'il code pour cette protéine.

Notons tout de suite que rien n'impose que la relation de correspondance soit

- transitive : on peut avoir $u \; S \; v$ et $v \; S \; w$ sans avoir $u \; S \; w$,
- fonctionnelle (resp. injective) : on peut avoir $u \; S \; v$ et $u \; S \; w$ (resp. $v \; S \; u$ et

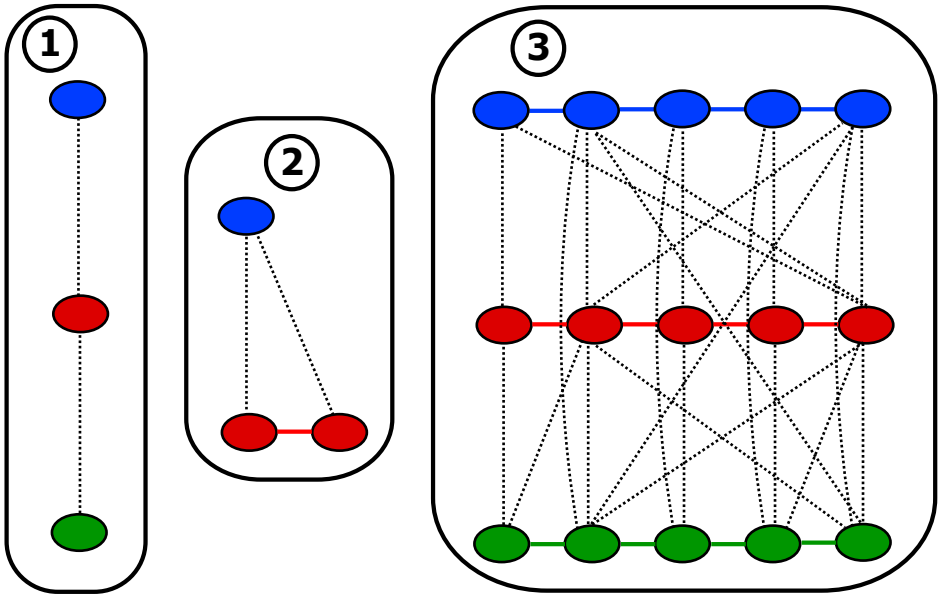


FIGURE 3.4 – Exemples de graphes de données stratifiés sur des réseaux primaires représentant des génomes. La relation de correspondance apparaît en pointillés. 1) Exemple de relation de correspondance non-transitive. 2) Exemple de relation de correspondance non-fonctionnelle. 3) Exemple de relation de correspondance dégénérée.

$w\ S\ u$) avec $u \in V_i$, $v \in V_j$ et $w \in V_j$ où $j \neq i$.
Dans la suite, une relation qui n'est ni fonctionnelle ni transitive sera dite "dégénérée". Dans la Figure 3.4, nous donnons des exemples de graphes de données : un premier avec une relation de correspondance non-transitive, un second avec une relation de correspondance non-fonctionnelle et un dernier avec une relation de correspondance dégénérée.

Une fois ce choix de construction effectué, il nous reste à définir quand n nœuds se correspondent. En effet, entre n nœuds pris chacun dans un réseau différent, il peut y avoir entre 0 et $\frac{n(n-1)}{2}$ arêtes inter-strates. Dans quels cas allons-nous considérer que ces n nœuds sont "aligné" ?
Cela revient à définir une n-correspondance au sens donné dans la définition 3.2.

Définition 3.2. *Etant donné le graphe de données stratifié* $D = (\bigcup_i V_i, E_P \cup E_S)$, *nous appellerons* n-correspondance *un sous-ensemble du produit cartésien* $V_1 \times V_2 \times \ldots \times V_n$. *Les éléments de la* n*-correspondance seront appelés des* empilements.

On notera dans la suite $\mathcal{R}(V_1 \times V_2 \times \ldots \times V_n) \subseteq (V_1 \times V_2 \times \ldots \times V_n)$ une

n-correspondance.

La définition la plus faible des empilements est la contrainte de connexité, mais on peut imaginer des contraintes de plus en plus fortes, jusqu'à forcer la présence de chacune des $\frac{n(n-1)}{2}$ arêtes.
Voyons des exemples de définitions d'empilements.

Définition 3.3. ***Empilement-Connexe***
Etant donné le graphe de données stratifié $D = (\bigcup_i V_i,\ E_P \cup E_S)$*, un n-uplet* $(v_1, \dots v_n) \in (V_1 \times V_2 \times \dots \times V_n)$ *est un* empilement-connexe *si et seulement si le sous-graphe induit par* $(v_1, \dots v_n)$ *est connexe.*

Définition 3.4. ***Empilement-Etoile***
Etant donné le graphe de données stratifié $D = (\bigcup_i V_i,\ E_P \cup E_S)$*, un n-uplet* $(v_1, \dots v_n) \in (V_1 \times V_2 \times \dots \times V_n)$ *est un* empilement-étoile *si et seulement si* $\forall i > 1,\ (v_1, v_i) \in E_S$.

Définition 3.5. ***Empilement-Clique***
Etant donné le graphe de données stratifié $D = (\bigcup_i V_i,\ E_P \cup E_S)$*, un n-uplet* $(v_1, \dots v_n) \in (V_1 \times V_2 \times \dots \times V_n)$ *est un* empilement-clique *si et seulement si* $\forall (i, j),\ (v_i, v_j) \in E_S$.

La définition des empilements-cliques (définition 3.5) impose une correspondance forte, chacun des nœuds doit être en correspondance avec les $n - 1$ autres nœuds de l'empilement. Cela implique que suivant les données et le type de relation de correspondance choisie, on peut éventuellement rater des alignements intéressants.
La définition des empilements-étoiles (définition 3.4) force une correspondance à un réseau central (le 1^{er}) pris comme référence.
Enfin, la définition des empilements-connexes (définition 3.3) est la plus faible, elle autorise le plus grand nombre d'empilements, dont malheureusement une partie peut correspondre à des empilements inintéressants d'un point de vue biologique.

Des variantes sont possibles, on peut par exemple ajouter à la définition des empilements-connexes une notion de densité (définition 3.6) en forçant que chaque nœud de l'empilement soit en correspondance avec au moins $\gamma \times (n - 1)$ autres nœuds de l'empilement où $0 < \gamma \leq 1$. Les sous-graphes correspondants sont appelés des γ-quasi-cliques. Formellement, un graphe $G = (V, E)$ est une γ-quasi-clique pour $0 < \gamma \leq 1$ si et seulement si $\forall v \in V$, $degré(v) \geq \gamma \times (|V| - 1)$.

Définition 3.6. ***Empilement-γ-Dense***
Etant donné le graphe de données stratifié $D = (\bigcup_i V_i,\ E_P \cup E_S)$*, un n-uplet*

$(v_1, \dots v_n) \in (V_1 \times V_2 \times \dots \times V_n)$ *est un* empilement-γ-dense *si et seulement si le sous-graphe induit par* $(v_1, \dots v_n)$ *est connexe et est une* γ*-quasi-clique.*

On peut aussi définir deux autres types d'empilements qui apparaissent dans la littérature, les empilements-chemins (définition 3.7), et les empilements-arbres (définition 5.3).

Définition 3.7. ***Empilement-Chemin***
Etant donné le graphe de données stratifié $D = (\bigcup_i V_i,\ E_P \cup E_S)$*, un n-uplet* $(v_1, \dots v_n) \in (V_1 \times V_2 \times \dots \times V_n)$ *est un* empilement-chemin *pour une permutation* p *des indices donnée si et seulement si* $\forall i \in [|1, n-1|],\ (v_{p(i)}, v_{p(i+1)}) \in E_S$*.* [1]

Pour les empilements-arbres, redonnons rapidement la définition des clades vue dans la Section 2.2.2.2. On appelle clade l'ensemble des indices des feuilles associées à un nœud interne de l'arbre. Soit $\mathcal{C}(T)$ l'ensemble des clades de T, notons $(v_1, \dots v_n)_I$ l'ensemble $\{v_i,\ i \in I\}$.

Définition 3.8. ***Empilement-Arbre***
Etant donné le graphe de données stratifié $D = (\bigcup_i V_i,\ E_P \cup E_S)$*, un n-uplet* $(v_1, \dots v_n) \in (V_1 \times V_2 \times \dots \times V_n)$ *est un* empilement-arbre *pour un arbre donné* T *si et seulement si pour tout clade* $I \in \mathcal{C}(T)$*,* $(v_1, \dots v_n)_I$ *admet un chemin pour* E_S*.*

Le choix de la meilleure définition des empilements dépend entièrement du type de données (notamment de la relation de correspondance) et du type d'objet biologique recherché. Si l'on a affaire à une relation de correspondance dégénérée (par exemple si l'on cherche à retrouver des synténies entre des génomes d'espèces voisines), utiliser des empilements-cliques sera préférable ; si, par contre, le graphe de données stratifié comporte assez peu d'arêtes, il vaudra mieux utiliser des empilements-connexes. Les empilements-γ-denses sont un cas intermédiaire entre la clique ($\gamma = 1$) et la composante connexe ($\gamma = \frac{1}{n-1}$).

Les autres types d'empilements correspondent à des cas très spécifiques. Typiquement les empilements-étoiles seront utilisés lorsqu'on possède un réseau de référence sur lequel on veut aligner le reste des réseaux, dans le cas de données génomiques, l'objectif peut être l'annotation fonctionnelle : le réseau central est déjà annoté et on cherche à proposer des annotations pour le reste des réseaux.

Pour les empilements-arbres, ils seront potentiellement utiles si l'on aligne des réseaux du même type, par exemple des réseaux PPI, et que l'on possède un arbre phylogénétique.

1. Notons que dans cette définition la permutation est fixée au départ. On pourrait imaginer une définition plus générale dans laquelle il suffit qu'une permutation existe. L'intérêt biologique d'une telle définition reste néanmoins douteux.

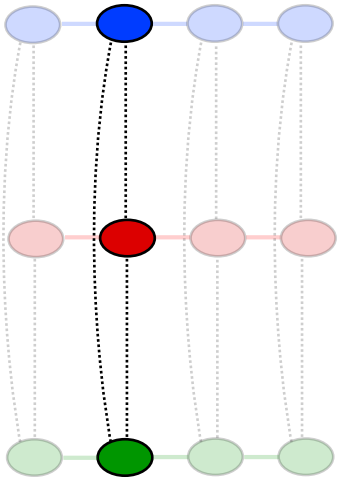


FIGURE 3.5 – Exemple d'un empilement-clique. Les nœuds et arêtes de l'empilement apparaissent en foncé.

Quant aux empilements-chemins, il s'agit simplement d'une approximation des empilements-arbres, utilisée notamment dans NETWORKBLAST-M [KBS08].

Nous pouvons maintenant définir formellement notre contrainte d'empilement.

On dira d'un nœud qu'il est bien empilé dans un graphe de données stratifié s'il appartient à au moins un empilement inclus dans ce graphe (définition 5.12).

Définition 3.9. *Etant donné le graphe de données stratifié* $D = (V = \bigcup_i V_i, E = E_P \cup E_S)$ *et la n-correspondance choisie* $\mathcal{R}(V_1 \times V_2 \times \ldots \times V_n)$*, on dira qu'un nœud* $v_i \in V_i$ *est* bien empilé *dans un sous-graphe* $D' = (V', E')$ *de* D *si et seulement si* $\exists$ *un ensemble de* $(n-1)$ *nœuds* $\{v_j\}_{j \neq i} \subset V'$ *tel que* $(v_1, \ldots, v_n) \in \mathcal{R}(V_1 \times V_2 \times \ldots \times V_n)$.

Cela nous permet enfin de définir la contrainte d'empilement (définition 5.13).

Définition 3.10. *Etant donné le graphe de données stratifié* $D = (V = \bigcup_i V_i, E = E_P \cup E_S)$ *et la n-correspondance choisie* $\mathcal{R}(V_1 \times V_2 \times \ldots \times V_n)$ *, on dira qu'un sous-graphe* $D' = (V', E')$ *de* D *satisfait la contrainte d'empilement si et seulement si chacun des nœuds de* V' *est bien empilé dans* D'.

Informellement, imposer cette contrainte d'empilement revient à imposer que chaque nœud du sous-graphe fasse partie d'un empilement dans le sous-graphe.

Notes d'implémentation.
Dans l'implémentation des algorithmes de cette thèse, qui est décrite en Annexe 8.1, le choix de la contrainte d'empilement est contrôlé par le paramètre **aggregator** qui peut prendre les valeurs suivantes :

- cc : empilement-connexe définition 3.3,
- center : empilement-étoile définition 3.4,
- clique : empilement-clique définition 3.5,
- dense : empilement-γ-dense définition 3.6.

3.3 Voisinage et contrainte de localité

Comme nous l'avons vu précédemment, la notion de voisinage est traduite par les ensembles d'arêtes intra-strates du graphe de données stratifié : $E_P = \bigcup_i E_i$.

Les arêtes intra-strates ont des significations différentes suivant les types de réseaux. Citons par exemple :

- génome : deux gènes sont reliés s'ils sont adjacents sur le génome,
- réseau PPI : deux protéines sont reliées si elles interagissent (suivant une méthode expérimentale donnée),
- graphe métabolique : deux réactions sont reliées s'il existe un composé qui est le substrat d'une réaction et le produit de l'autre.

Nous voulons que sur chaque strate le sous-graphe induit par l'alignement soit un voisinage, ce qui sera pour nous synonyme de composante connexe.

Ce type de sous-graphe sera dit "connexe par strate", la définition formelle est donnée ci-dessous.

Définition 3.11. *Etant donnés n réseaux primaires $G_i = (V_i, E_i)$, $i \in [|1, n|]$ et le graphe de données stratifié associé $D = (V = \bigcup_i V_i,\ E = E_P \cup E_S)$, on dira d'un sous-graphe $D' = (V', E')$ de D qu'il est* connexe par strates *si et seulement si $\forall i \in [|1, n|]$, le graphe $(V' \cap V_i, E' \cap E_i)$ forme une composante connexe.*

Cette définition assure que, sur chaque strate, D' sera bien un voisinage.

Nous pouvons enfin définir formellement la contrainte de localité.

Définition 3.12. *Contrainte de localité.*
Un sous-graphe D' d'un graphe de données stratifié D satisfait la contrainte de localité si et seulement si il est connexe par strates.

Dans la littérature, les algorithmes utilisent souvent des contraintes plus fortes que la simple connexité. Ainsi, dans le cas de l'alignement de réseaux PPI, de nombreuses études ([SSK+05], [KBS08], [FNS+06]) imposent de plus une densité minimale des graphes $(V' \cap V_i, E' \cap E_i)$.

3.4 Définition des alignements locaux

Nous venons de définir la contrainte de localité et la contrainte d'empilement. Il nous reste à ajouter une contrainte de maximalité au sens des nœuds.

Définition 3.13. *Un sous-graphe D' d'un graphe de données stratifié D est un alignement local s'il vérifie la contrainte d'empilement et la contrainte de localité, et si l'on ne peut pas lui ajouter de nœud sans invalider une de ces deux contraintes.*

Dans le paragraphe suivant, nous allons introduire une représentation plus riche que le graphe de données stratifié qui permettra de représenter simultanément la contrainte de localité et la contrainte d'empilement. Ceci permettra de définir plus simplement les alignements locaux et de proposer des algorithmes pratiques pour les calculer.

3.5 Multigraphe d'alignement

Dans un multigraphe d'alignement (MGA), la contrainte d'empilement est représentée par les nœuds et la contrainte de localité par les arêtes. Les nœuds d'un MGA sont simplement les empilements. On remarque déjà que cela induira la plupart du temps une perte d'information par rapport au graphe de données stratifié puisque toute l'information sur les nœuds qui n'appartiennent pas à des empilements est perdue.

La seconde étape consiste à introduire les arêtes qui vont rendre compte du voisinage sur les réseaux primaires.
Etant donnée la contrainte de localité que nous avons choisie (la connexité sur chacun des graphes primaires), il nous faut conserver toute l'information de voisinage entre nos empilements, nous allons donc avoir besoin de n ensembles d'arêtes, chacun d'entre eux retranscrivant le voisinage sur un des réseaux primaires - d'où

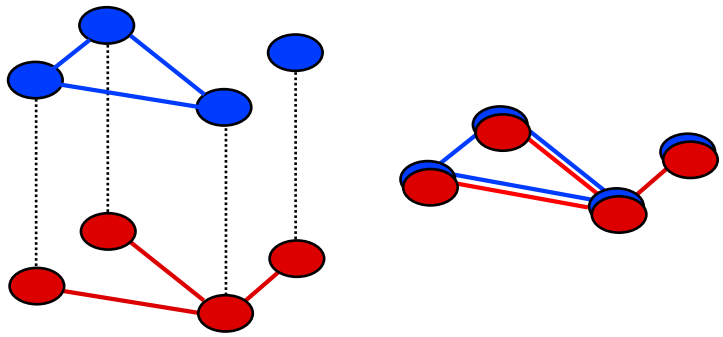


FIGURE 3.6 – Graphe de données stratifié (deux strates) sur la gauche et multigraphe d'alignement associé sur la droite.

l'appelation *multi*graphe.

Plus formellement :

Définition 3.14. *Pour un graphe de données stratifié* $D = (V = \bigcup_i V_i,\ E = E_P \cup E_S)$ *et une n-correspondance* $\mathcal{R}$*, le* multigraphe d'alignement *(MGA) est le multigraphe* $M = (\mathcal{V}, \mathcal{E}_1, \ldots, \mathcal{E}_n)$ *tel que :*

- $\mathcal{V} = \mathcal{R}(V_1 \times V_2 \times \ldots \times V_n)$
- $\forall u = (u_1, u_2, \ldots, u_n) \in \mathcal{V}$ *et* $v = (v_1, v_2, \ldots, v_n) \in \mathcal{V}$,
  $(u, v) \in \mathcal{E}_i \Leftrightarrow (u_i, v_i) \in E_i \vee (v_i = u_i)$

La Figure 3.6 présente, dans le cas simple de l'alignement de deux réseaux, un exemple de multigraphe d'alignement. Dans la figure 3.7, nous voyons un exemple un peu plus complexe, dans le cas d'une relation de correspondance non-fonctionnelle. Et enfin dans la figure 3.8, nous donnons un exemple de multigraphe associé à un graphe de données à trois strates.

La taille de cette représentation est potentiellement exponentielle dans le nombre d'empilements et - surtout - d'arêtes lorsque l'on augmente le nombre de réseaux primaires et que l'on utilise une relation de correspondance plus dégénérée.

Ainsi, dans le pire des cas, pour une relation dégénérée mettant tous les nœuds en correspondance, le nombre d'empilements serait en $O(N^n)$ où N est le nombre de nœuds d'un réseau primaire.

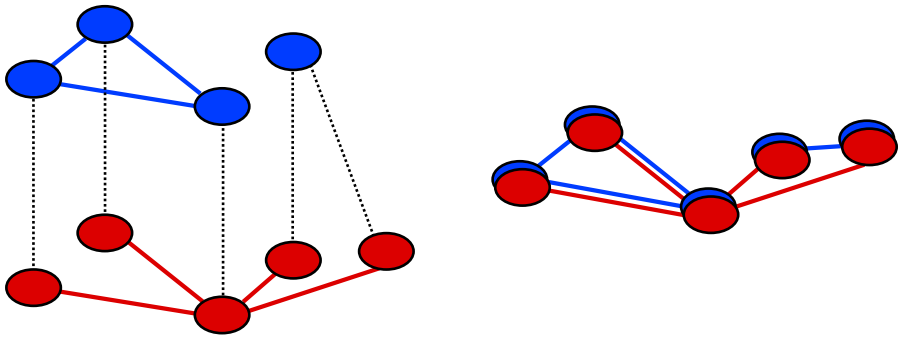


FIGURE 3.7 – Graphe de données stratifié (deux strates) sur la gauche avec une relation de correspondance non-fonctionnelle et multigraphe d'alignement associé sur la droite. On remarque que le nœud à l'extrême droite du réseau bleu apparaît dans deux nœuds du multigraphe, puisqu'il appartient à deux empilements.

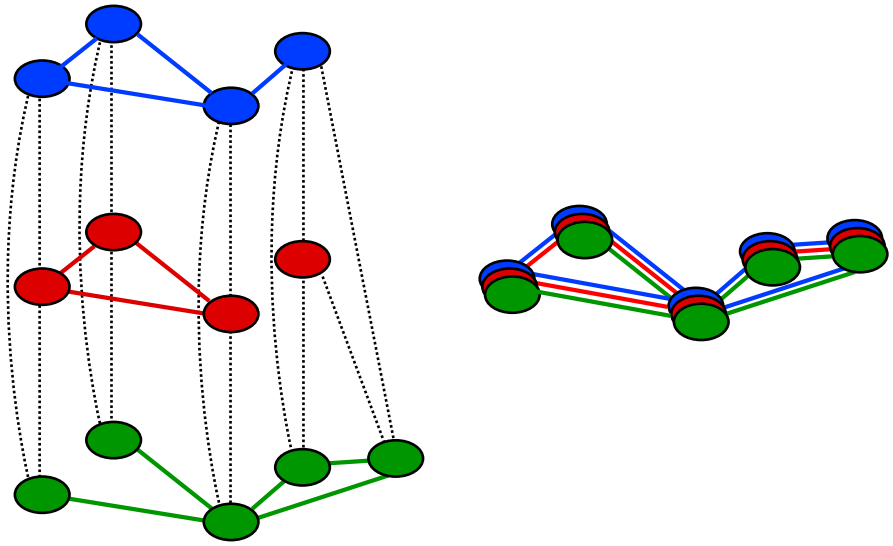


FIGURE 3.8 – Graphe de données stratifié (à trois strates) sur la gauche et multigraphe d'alignement associé sur la droite. Dans cet exemple, la condition d'empilement choisie est que les nœuds forment une clique de la relation S.

3.6 Définition des connectons

Dans cette section, nous allons définir un ensemble de sous-graphes du multigraphe d'alignement, appelés connectons, et nous montrerons ensuite que ces connectons correspondent exactement aux alignement locaux du graphe de données stratifié tels que définis dans la section 3.4.

Définition 3.15. *Un sous-graphe* $(\mathcal{V}', \mathcal{E}'_1, \ldots, \mathcal{E}'_n)$ *du MGA est dit* n-connexe *si et seulement si* $\forall i \in [|1, n|]$, $(\mathcal{V}', \mathcal{E}'_i)$ *est connexe.*

En d'autres termes, un sous-graphe est n-connexe s'il est connexe sur chaque couleur [2] individuellement.

Nous pouvons maintenant définir les connectons.

Définition 3.16. *Un* connecton *est un sous-graphe* n*-connexe maximal du MGA.*

Maximal signifie qu'on ne peut lui ajouter aucun nœud sans briser la n-connexité. La figure 3.9 donne un exemple de MGA avec 3 connectons.

Vérifions maintenant que les connectons sont exactement les alignements locaux du graphe de données stratifié.

Proposition 1. *Les connectons correspondent exactement aux alignements locaux du graphe de données stratifié.*

Démonstration. Nous allons commencer par montrer que les connectons sont des alignements locaux, puis nous prouverons que tout alignement local correspond à un connecton.
Soit un connecton $C = (\mathcal{V}', \mathcal{E}'_1, \ldots, \mathcal{E}'_n)$ du multigraphe $M = (\mathcal{V}, \mathcal{E}_1, \ldots, \mathcal{E}_n)$ et le sous-graphe correspondant D' dans le graphe de données stratifié.
D' vérifie naturellement la contrainte d'empilement puisque les nœuds du multigraphe sont des empilements. D' vérifie aussi la contrainte de localité par construction du multigraphe d'alignement $((u, v) \in \mathcal{E}'_i \Leftrightarrow (u_i, v_i) \in E_i \vee (v_i = u_i))$ et le fait que $\forall i \in [|1, n|]$, $(\mathcal{V}', \mathcal{E}'_i)$ est connexe.
Enfin D' est maximal puisque C est maximal : on ne peut lui ajouter que des nœuds mal empilés, ce qui viendrait contredire la contrainte d'empilement.
Tout connecton correspond donc à un alignement local. Il nous reste à montrer que tout alignement local correspond bien à un connecton.
Soit D' un alignement local dans le graphe de données stratifié.

2. Dans la suite, nous utiliserons le terme "couleur" pour désigner l'ensemble des arêtes associées à une strate.

Il vérifie donc la contrainte d'empilement, ce qui implique qu'il correspond à un sous-ensemble $\mathcal{V}'$ de nœuds du MGA. Soit $(\mathcal{E}'_1, \dots \mathcal{E}'_n)$ les ensembles d'arêtes qui relient les nœuds de $\mathcal{V}'$, D' vérifie aussi la contrainte de localité, ce qui assure par construction du MGA, que $\forall i \in [|1, n|]$, $(\mathcal{V}', \mathcal{E}'_i)$ est une composante connexe. On sait que $C = (\mathcal{V}', \mathcal{E}'_1, \dots, \mathcal{E}'_n)$ est maximal puisque D' est maximal, donc C est un connecton.

Nous donnons maintenant une propriété importante des connectons qui va nous permettre de concevoir des algorithmes efficaces et simples d'énumération des alignements locaux, ce qui n'était pas possible dans le graphe de données stratifié.

Proposition 2. *Les connectons forment une partition des nœuds du multigraphe d'alignement.*

Démonstration. Procédons par l'absurde.
Supposons la propriété fausse, c'est-à-dire qu'il existe au moins deux connectons $C_1 = (\mathcal{U}_1, \mathcal{F}_1, \dots \mathcal{F}_n)$ et $C_2 = (\mathcal{U}_2, \mathcal{F}'_1, \dots \mathcal{F}'_n)$ distincts et d'intersection non-nulle.
Montrons que $C = C_1 \cup C_2$ est n-connexe : pour cela il nous faut montrer que tout couple de nœuds de $(\mathcal{U}_1 \cup \mathcal{U}_2)$ est connecté pour chacun des $\mathcal{F}_i \cup \mathcal{F}'_i$, $i \in [|1, n|]$.
Prenons deux nœuds v,$v' \in (\mathcal{U}_1 \cup \mathcal{U}_2)$, nous avons deux cas de figure.
Si v et v' appartiennent tous les deux à un même connecton C_1 (resp. C_2), alors ils sont connectés pour chacun des $\mathcal{F}_i$ (resp. $\mathcal{F}'_i$), $i \in [|1, n|]$ par définition des connectons.
Si v et v' appartiennent à des connectons différents (par exemple $v \in \mathcal{U}_1$ et $v' \in \mathcal{U}_2$), prenons un nœud u de l'intersection, $u \in \mathcal{U}_1 \cap \mathcal{U}_2$.
Par définition du connecton C_1, v et u sont connectés pour chacun des $\mathcal{F}_i$, $i \in [|1, n|]$. Par définition du connecton C_2, v' et u sont connectés pour chacun des $\mathcal{F}'_i$, $i \in [|1, n|]$. Donc v et v' sont connectés pour chacun des $\mathcal{F}_i \cup \mathcal{F}'_i$, $i \in [|1, n|]$.
C est donc n-connexe.
Or $C_1 \subset C$ ce qui vient contredire la maximalité du connecton C_1.
Les connectons forment donc bien une partition des nœuds du multigraphe d'alignement [3].

Notons tout de suite que cette propriété ne serait pas vérifiée si l'on avait choisi comme contrainte de localité une contrainte plus forte que la connexité (comme nous l'avons vu dans le Chapitre 2, il est courant dans les approches d'alignement de réseaux PPI d'imposer plutôt une contrainte de densité).

3. Notons que, par contre, les alignements locaux ne forment pas une partition des nœuds du graphe de données stratifié, parce que d'une part, dans le cas général certains nœuds sont mal empilés, et d'autre part, un nœud peut appartenir à plusieurs empilements qui peuvent être dans des alignements différents.

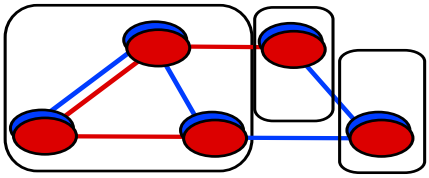


FIGURE 3.9 – MGA avec trois connectons. Chacun des connectons est bien connexe pour chacune des trois strates, et ils sont bien maximaux. On remarque qu'ils forment une partition du MGA.

Cependant, on peut remarquer que notre condition de connexité étant plus faible, les alignements denses sur chacun des réseaux seront inclus dans nos résultats. Il sera donc possible - si nos résultats ne sont pas trop nombreux - de procéder à la sélection d'un sous-ensemble de connectons qui vérifient cette condition supplémentaire de densité sur chacun des réseaux, permettant de résoudre ce problème également de façon exacte.

```
Notes d'implémentation.
Dans l'implémentation, nous offrons la possibilité de fixer une
taille minimale des connectons. Cette taille peut être définie
soit en nombre de nœuds du multigraphe (c'est à dire en nombre
d'empilements), soit en nombre de nœuds par strate.
Les deux paramètres associés sont les suivants :
 minsize : taille min d'un connecton en nombre d'empilements,
 mineltsize : taille min d'un connecton sur chacune des strates.
```

3.7 Interprétation biologique

Maintenant que nous avons une définition formelle de ce qu'est un alignement local de réseaux, retournons un instant vers la biologie pour expliquer sur quelques exemples pratiques les objets biologiques qui seront retrouvés en utilisant cette définition.

Dans le cadre d'un problème d'alignement local de génomes, les connectons calculés correspondent donc à des segments de génomes conservés. S'il s'agit de génomes de procaryotes, ces connectons pourront être interprétés comme des opérons - groupement de gènes adjacents et co-régulés qui réalisent une même fonction physiologique. De manière plus générale, nos connectons correspondent à des synténies

(Section 2.4).

En ce qui concerne l'alignement de réseaux PPI, classiquement les composantes connexes denses d'un réseau PPI sont interprétées comme des complexes protéiques, nos connectons pourront être interprétés de la même manière.

Lorsque notre définition est appliquée à l'alignement de réseaux métaboliques, les connectons pourront être interprétés comme des voies métaboliques (ensemble de réactions reliées par leurs produits - substrats).

Enfin, dans le cas d'alignement de données hétérogènes, par exemple un génome contre un réseau métabolique, un connecton correspondra à un ensemble de gènes adjacents sur le chromosome qui codent pour des enzymes impliquées dans des réaction chimiques connectées (par exemple dans une même voie métabolique).

3.8 Extensions : trous et conservation partielle de la topologie

3.8.1 Trous

On constate immédiatement que la contrainte de localité (connexité) risque d'être non vérifiée très facilement si des nœuds sont manquants. Dans le cas génomique, par exemple, une délétion de gène risque d'invalider la connexité entre deux gènes, et donc l'alignement.
Un des problèmes classiques dans le domaine de l'alignement de séquence est justement la gestion de ces "trous" dans l'alignement ("gaps" en anglais).
De manière générale, il s'agit de maintenir la connexité d'un alignement, malgré la présence d'éléments sans correspondants au sein de l'alignement. Dans notre graphe de données stratifié, la proximité entre nœuds étant représentée par les arêtes intra-strates (E_P), une stratégie simple pour relâcher cette contrainte consiste donc à rajouter des arêtes intra-strates entre des nœuds "pas trop éloignés".
En pratique, nous allons utiliser la δ-fermeture transitive d'un graphe (Définition 3.17).

Définition 3.17. *La δ-fermeture transitive d'un graphe $G = (V, E)$ est le graphe $G' = (V, E')$ où : E' est l'ensemble des couples $(x, y) \in V^2$ tels qu'il existe un chemin entre x et y dans G de longueur $\leq \delta$.*

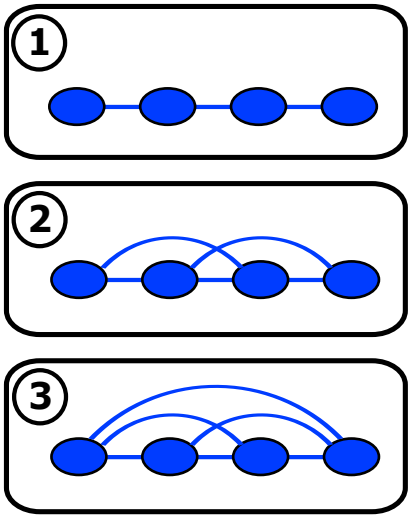


FIGURE 3.10 – Exemple de fermeture transitive d'un graphe. 1) graphe G. 2) la 2-fermeture transitive de G. 3) la 3-fermeture transitive de G. Pour $\delta > 3$, la δ-fermeture transitive de notre graphe restera identique à sa fermeture transitive.

Pour $\delta = \infty$, E' est l'ensemble des couples $(x, y) \in V^2$ tels qu'il existe un chemin entre x et y dans G, c'est à dire la fermeture transitive de G

En calculant la δ-fermeture transitive de chacune de nos strates, nous allons donc pouvoir autoriser des trous comportant $(\delta - 1)$ nœuds.
A noter qu'il est possible de choisir des valeurs de δ différentes pour chacun des réseaux primaires, ce qui fait sens notamment dans le cas de données hétérogènes.

La figure 3.10 donne un exemple de k-fermetures transitives pour $k = 2$ et $k = 3$. Ensuite, dans la Figure 3.11, nous en voyons la conséquence sur les alignements : en remplaçant dans notre graphe de données stratifié un réseau par sa 2-fermeture transitive, nous autorisons les trous de taille 1, ce qui revient à relâcher la contrainte de localité.

Notes d'implémentation.
Le paramètre correspondant à δ est appelé **deltagap**.
L'utilisateur peut définir une valeur de **deltagap** différente sur chaque réseau primaire.

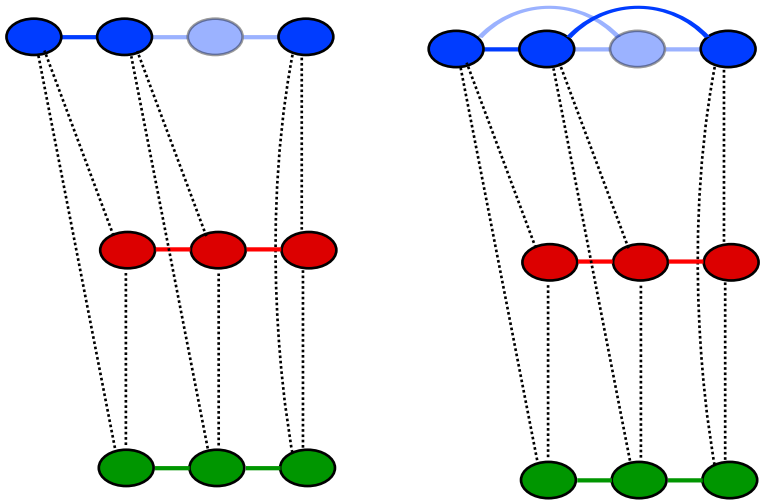


FIGURE 3.11 – Deux exemples d'alignements au sein d'un graphe de données stratifié. Sur la gauche - sans fermeture transitive - les nœuds foncés ne forment pas un alignement puisqu'ils ne sont pas connectés sur le premier réseau. Par contre sur la droite nous avons utilisé la 2-fermeture transitive de la première strate, ce qui nous permet d'autoriser un trou de taille 1 sur ce réseau et donc de retrouver l'alignement.

3.8.2 Conservation partielle de la topologie

Les deux cas classiques et opposés rencontrés dans la littérature sont les suivants : à un bout du spectre, on a les alignements qui forcent une conservation stricte de la topologie dans chacun des réseaux, à l'autre bout, on a les alignements qui autorisent tous les réarrangements de nœuds dans chacun des réseaux, à condition qu'ils restent connectés.

Entre ces deux cas extrêmes, il peut être intéressant de trouver des situations intermédiaires, c'est-à-dire de définir une mesure du degré de réarrangement autorisé localement dans un connecton. Cela permettrait, par exemple dans le cas génomique, de n'autoriser des permutations que si les gènes concernés restent proches.

Lorsqu'on cherche une topologie commune à plusieurs graphes $G_1, G_2, \dots G_n$ définis sur le même ensemble de nœuds V, une idée naturelle est de chercher un graphe $G = (V, E)$ qui soit inclus dans chacun des graphes G_i, c'est-à-dire qui en constitue un "squelette commun". Cela nous amène à une première définition de connectons à topologie conservée.

Définition 3.18. *Un sous-graphe $G' = (\mathcal{V}', \mathcal{E}'_1, \dots, \mathcal{E}'_n)$ du MGA est un* connecton à topologie conservée *si et seulement si G' est un sous-graphe n-connexe maximal tel que $\exists G = (\mathcal{V}', \mathcal{E})$ connexe tel que $\forall i$, $\mathcal{E} \subseteq \mathcal{E}'_i$.*

La conservation de la topologie est assurée par l'inclusion de ce squelette G dans chacun des ensembles d'arêtes. Si l'on veut autoriser une conservation seulement partielle de la topologie, il faut donc affaiblir cette condition d'inclusion. C'est ce que nous allons faire en pratique en imposant seulement l'inclusion de $\mathcal{E}$ dans des ensembles plus grands, les ensembles $\mathcal{E}'_i$ auxquels on aura ajouté des arêtes via une k-fermeture transitive (Définition 3.17).

Cela nous amène à la définition suivante de connectons vérifiant une condition de conservation partielle de la topologie.

Définition 3.19. *Un sous-graphe $G' = (\mathcal{V}', \mathcal{E}'_1, \dots, \mathcal{E}'_n)$ du MGA est un* connecton à topologie k-conservée *si et seulement si G' est un sous-graphe n-connexe maximal tel que $\exists G = (\mathcal{V}', E)$ connexe tq $\forall i$, $\mathcal{E} \subseteq \bar{\mathcal{E}}'^{k}_i$, où $\bar{\mathcal{E}}'^{k}_i$ est l'ensemble d'arêtes de la k-fermeture transitive du graphe $(\mathcal{V}', \mathcal{E}'_i)$.*

On constate bien que pour une valeur $k = 0$, on retrouve la définition de connectons à topologie conservée (définition 3.18), alors que pour une valeur $k = \infty$, on

retrouve la définition originelle des connectons, sans aucune contrainte de conservation de topologie. Entre ces deux valeurs de k, nous allons pouvoir retrouver des connectons dans lesquels la topologie sera conservée partiellement, ce qui nous permettra, par exemple, de borner la magnitude des réarrangements dans le cas génomique.

Une propriété importante des connectons à topologie k-conservée est la suivante.

Proposition 3. *Les connectons à topologie k-conservée sont exactement les sous-graphes maximaux $G' = (\mathcal{V}', \mathcal{E}'_1, \ldots, \mathcal{E}'_n)$ du MGA tels que $(\mathcal{V}', \bigcap_{j=1}^{n} \overline{\mathcal{E}'_j}^{k})$ est connexe.*

Démontrons cette propriété.

Démonstration. Nous allons commencer par montrer que pour tout connecton à topologie k-conservée $(\mathcal{V}', \mathcal{E}'_1, \ldots, \mathcal{E}'_n)$, $(\mathcal{V}', \bigcap_{j=1}^{n} \overline{\mathcal{E}'_j}^{k})$ est connexe. Nous montrerons ensuite l'implication inverse.
Prenons un connecton à topologie k-conservée $(\mathcal{V}', \mathcal{E}'_1, \ldots, \mathcal{E}'_n)$. Par définition, $\exists G = (\mathcal{V}', \mathcal{E})$ connexe tq $\forall i$, $\mathcal{E} \subseteq \overline{\mathcal{E}'_i}^{k}$.
$\mathcal{E}$ est inclus dans chacun des $\overline{\mathcal{E}'_i}^{k}$, donc dans leur intersection $\bigcap_{j=1}^{n} \overline{\mathcal{E}'_j}^{k}$.
Cela implique donc que G est inclus dans le graphe $(\mathcal{V}', \bigcap_{j=1}^{n} \overline{\mathcal{E}'_j}^{k})$.
G étant connexe, on obtient directement la connexité de $(\mathcal{V}', \bigcap_{j=1}^{n} \overline{\mathcal{E}'_j}^{k})$.
Montrons maintenant l'implication réciproque.
Prenons un connecton $(\mathcal{V}', \mathcal{E}'_1, \ldots, \mathcal{E}'_n)$ tel que $(\mathcal{V}', \bigcap_{j=1}^{n} \overline{\mathcal{E}'_j}^{k})$ est connexe.
Pour prouver que ce connecton est une connecton à topologie k-conservée, il suffit d'exhiber un graphe possédant les propriétés voulues.
Choisissons comme graphe G le graphe $(\mathcal{V}', \bigcap_{j=1}^{n} \overline{\mathcal{E}'_j}^{k})$.
G est bien connexe, et vérifie la condition supplémentaire d'inclusion dans chacune des fermetures transitives.
Notre propriété est bien correcte.

Cette propriété sera utilisée dans l'implémentation.

Notes d'implémentation.
Le paramètre correspondant à k dans l'implémentation est appelé **deltashuf**.
L'utilisateur peut définir une valeur de **deltashuf** différente sur chaque réseau primaire.

Chapitre 4

Algorithme de construction et partitionnement à la volée du MGA

Nous avons commencé par définir les alignements locaux maximaux du graphe de données stratifié, puis nous avons défini des objets qui leur correspondent exactement : les connectons du multigraphe d'alignement.

Il nous reste à expliquer comment ces connectons sont calculés d'un point de vue algorithmique. C'est ce que nous allons faire dans ce chapitre.

Nous allons commencer par expliquer pourquoi l'approche naturelle, de construction puis de partitionnement du multigraphe, échoue en pratique dans le cas d'une relation très dégénérée ou de nombreux réseaux, puis nous détaillerons l'approche que nous avons développée.

4.1 Algorithmes de partitionnement du MGA

Une première approche naturelle, étant donnée la formalisation choisie (Chapitre 3), consiste à :

– construire le multigraphe d'alignement,

– le partitionner.

Le processus de partitionnement n'est pas trivial, remarquons tout d'abord qu'il ne correspond pas simplement à l'intersection des composantes connexes de chaque graphe primaire. Ceci est simplement du au fait qu'une partie d'une composante connexe n'est pas nécessairement connexe. Ainsi, dans la Figure 4.1, si

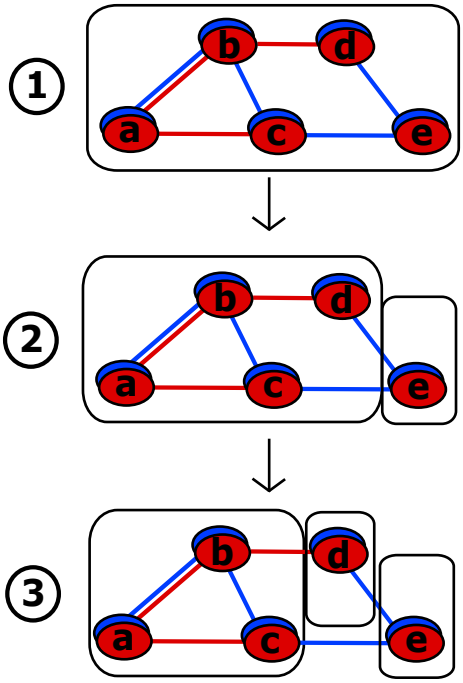


FIGURE 4.1 – 1) Multigraphe d'alignement. 2) Partition du multigraphe suivant l'intersection des composantes connexes bleues ($\{a, b, c, d, e\}$) et rouges ($\{a, b, c, d\}\{e\}$). On remarque que $\{a, b, c, d\}$ n'est toujours pas un connecton, puisque le nœud d est déconnecté pour la couleur bleue. 3) Partition du multigraphe en connectons.

nous calculons l'intersection des composantes connexes rouges et bleues, nous obtenons $\{a, b, c, d\}$ et $\{e\}$, et on constate que $\{a, b, c, d\}$ n'est pas connexe pour les arêtes bleues.

Plusieurs algorithmes exacts ont été proposés pour procéder au partitionnement du multigraphe d'alignement.

L'algorithme utilisé par Boyer *et al* (C3PART [BML+05]) calcule de façon itérative la partition, en commençant avec une seule classe contenant tous les nœuds du multigraphe d'alignement, puis en raffinant cette partition à chaque itération. La procédure de C3PART fonctionne de la façon suivante :

1. commencer avec une classe contenant tous les nœuds,
2. calculer $\bigcap_i (CC_i)$, l'intersection de toutes les composantes connexes pour tous les ensembles d'arêtes, ce qui aboutit à une nouvelle partition dans laquelle

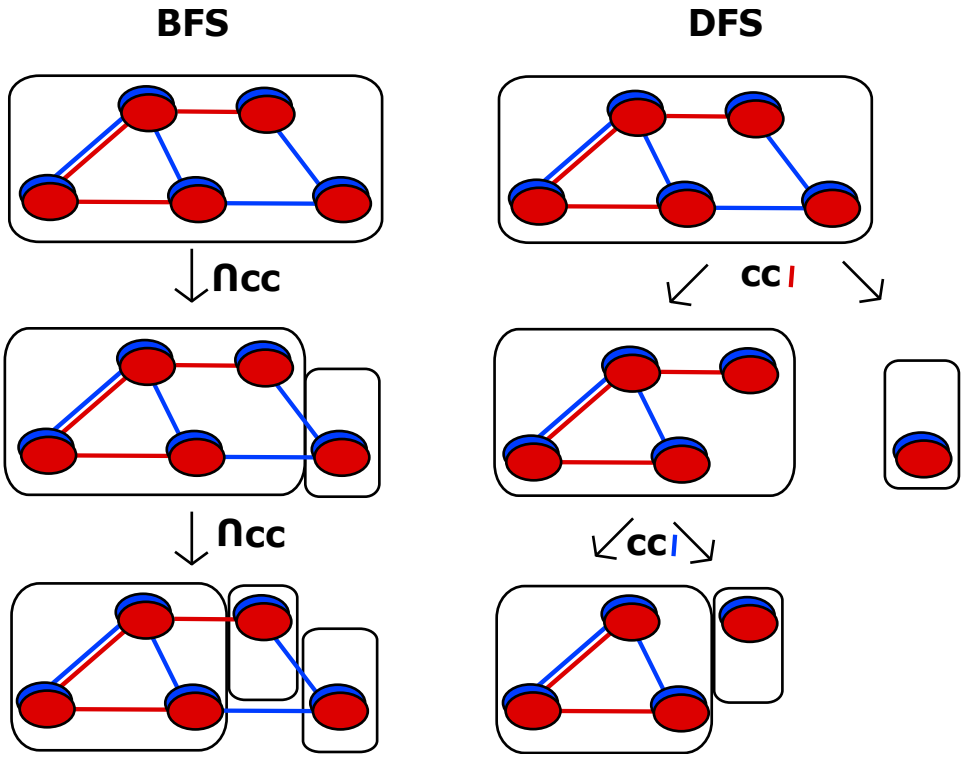


FIGURE 4.2 – Fonctionnement simplifié des algorithmes de partitionnement $C3Part$ original (BFS) et $C3Part$ amélioré (DFS) . $C3Part$ original utilise une approche en largeur d'abord (BFS) et calcule systématiquement l'intersection des composantes connexes sur chaque couleur, $C3Part$ amélioré utilise une approche en profondeur d'abord (DFS) et parcourt les couleurs en boucle, partitionnant dès qu'une couleur l'exige (c'est-à-dire dès que la classe courante n'est pas connexe pour la couleur en question).

chaque classe est un connecton potentiel,

3. itérer l'étape 2 sur chacune des classes jusqu'à ce que la partition soit stable.

Dans l'article original, C3Part utilisait une approche en largeur d'abord, dont nous donnons le pseudocode ci-dessous (Algorithme 1).
Ce pseudocode utilise deux algorithmes connus : un algorithme de calcul de composantes connexes, et un algorithme de calcul d'intersection.

Dans le pire des cas, la complexité en temps est $O((N + M) \times N)$ où N est le nombre de nœuds et M le nombre d'arêtes du multigraphe. Cela découle du fait que chaque itération demande $O(N+M)$ opérations pour calculer les composantes connexes et que, dans le pire des cas, l'algorithme devra faire $O(N)$ itérations (cela correspond au cas où il y a $O(N)$ classes à la fin et une classe a été extraite à chaque itération).

Cette complexité n'est pas optimale ; en 2003, Gai *et al* [HPR04] ont proposé un algorithme plus complexe, combinant la maintenance dynamique d'une "spanning forest" et une approche de partitionnement similaire à l'algorithme de Hopcroft. Cet algorithme a une complexité en $O((N + M \times \log N) \times \log N)$. De plus, dans le cas des graphes d'intervalles - correspondant notamment au cas de l'alignement de génomes -, la complexité devient $O((N + M) \times \log N)$.
En pratique, ni C3Part ni ces algorithmes ne sont utilisables dans le cas de l'alignement d'un grand nombre de réseaux, puisque leur pré-requis est la construction explicite du multigraphe d'alignement, dont la taille en nombre de nœuds comme en nombre d'arêtes augmente de façon exponentielle avec le nombre de réseaux quand la relation de correspondance S est dégénérée.
Si N_0 est le nombre de nœdus d'un graphe primaire et n le nombre de graphes primaires à aligner, on a, dans le pire cas d'une relation totalement dégénérée, $N \sim O(N_0^2)$ et $M \sim O(N_0^{2n})$. Afin de traiter ces cas, nous allons donc appliquer une approche similaire de partitionnement, mais en évitant la construction complète du multigraphe d'alignement.
L'idée générale est de construire le multigraphe à la volée, en commençant avec une composante connexe d'un premier réseau, puis en l'étendant sur un second réseau, en procédant ensuite à un partitionnement sur ces deux premiers réseaux, en étendant les résultats récursivement sur un troisième réseau, puis en partitionnant sur les trois réseaux, et ainsi de suite. Notons que l'approche détaillée ci-dessous sera donc indépendante du choix de l'algorithme de partitionnement en lui-même.

Une première étape est de transformer l'algorithme de C3Part en algorithme de partitionnement en profondeur d'abord, afin d'introduire ensuite la construction

à la volée du multigraphe. Nous avons introduit dans le même temps une boucle sur les couleurs, permettant de partitionner la classe courante au plus tôt, c'est-à-dire dès qu'une couleur l'exige, ce qui permet de réduire les calculs nécessaires (voir figure 4.2 pour un exemple).
Nous donnons le pseudocode de ce C3Part amélioré ci-dessous (algorithme 2). On remarque que lorsqu'on partitionne sur une couleur i, on lance l'appel récursif sur la classe courante C, qui va relancer une boucle sur les couleurs, et va peut-être appeller la procédure de calcul des composantes connexes pour I ce qui est inutile puisque C est issue du partitionnement sur i : on sait qu'elle est connexe pour i.
En pratique, le code est donc légèrement plus complexe que celui présenté ici : l'appel récursif $DFS(NewC)$ transmettra l'information qu'il est issu du partitionnement sur i, ce qui évite l'appel en question.

Algorithm 1: BFS /* C3Part original */.
Global: Multigraphe d'alignement $M = (V', E'_1, \ldots, E'_n)$
Input: Partition P */* *initialisée avec une seule classe : V'* */*
Variables:
Partition $NewP$ */* *initialisée à $\emptyset$* */*
Tableau de partitions CC */* *initialisé à vide* */*.

```
(1)   begin
(2)     /* pour chaque classe de la partition courante P */
(3)     for i = 1 to |P| do
(4)       /* calculer les composantes connexes pour chacun des
          E'_j */
(5)       for j = 1 to n do
(6)         CC[j] ← COMPOSANTES_CONNEXES(P[i], j);
(7)       end for
(8)       /* calculer l'intersection des composantes connexes */
(9)       NewP ← NewP ∪ INTERSECTION(CC);
(10)      end for
(11)    end for
(12)    if (|P| = |NewP|)then
(13)      /* toutes les classes sont stables */
(14)      PRINT(NewP);
(15)    else
(16)      /* certaines classes doivent encore être partitionnées */
(17)      BFS(NewP);
(18)    end if
(19)  end
```

Algorithm 2: DFS /* C3Part amélioré */.
Global: Multigraphe d'alignement $M = (V', E'_1, \ldots, E'_n)$
Input: Ensemble_de_nœuds C /* *classe courante : initialisée à V'* */
Variables: Partition P /* *partition courante sur le ième réseau* */

```
(1)  begin
(2)    /* partitionner sur chaque réseau séparément */
(3)    for i = 1 to n do
(4)      /* partitionner C sur le réseau i */
(5)      P ← COMPOSANTES_CONNEXES(C, i) ;
(6)      if (|P| ≠ 1)then
(7)        /* C a été partitionné sur le réseau i : recurrence*/
(8)        for NewC ∈ P do
(9)          DFS(NewC)
(10)       end for
(11)     return
(12)     end if
(13)   end for
(14)   /* C n'est plus partitionnable : c'est un connecton */
(15)   PRINT(C)
(16) end
```

4.2 Algorithme de construction et partitionnement à la volée du multigraphe d'alignement

Notre objectif est maintenant d'éviter la construction explicite du multigraphe d'alignement, ceci afin de pouvoir traiter de façon toujours exacte des cas d'alignement de nombreux réseaux avec des relations dégénérées.

Pour cela, nous allons nous baser sur l'algorithme DFS qui combine un parcours en profondeur d'abord et une boucle de partitionnement sur les couleurs. On remarque que, dans une branche d'exécution, tant qu'on n'est pas passé au moins une fois par la couleur k, les informations de connexité associées nous sont inutiles. Cela implique qu'on peut se contenter, à un moment donné de l'exécution, d'un multigraphe construit seulement sur les couleurs déjà parcourues.

En pratique, l'algorithme va donc s'appliquer à un "multigraphe courant" construit sur les k premières strates, et va alterner deux phases :

– une phase de partitionnement sur les k premières strates,

– une phase d'expansion sur une nouvelle strate.

La phase de partitionnement peut être assurée par tout algorithme exact précédemment mentionné, c'est-à-dire soit l'algorithme d'intersection simple de Boyer [BML+05] soit celui plus complexe de Gai [HPR04]. Nous noterons $Partitionner_{\{1...k\}}(C)$ l'opération de partitionnement de C sur les k premières couleurs.
Quant à la phase d'expansion, elle est assurée par la procédure *Prolonger* qui sera lancée au moment où l'on atteint une couleur qui n'a pas encore été traitée dans la branche d'exécution. Cette phase - décrite dans la section 4.2.2 - permet d'ajouter une nouvelle couleur $(k+1)$ au multigraphe courant.

En pratique que gagnons-nous par rapport à la construction explicite du multigraphe d'alignement ? Nous évitons premièrement la construction d'un grand nombre d'arêtes, qui relient les parties du multigraphe d'alignement qui sont partitionnées. Le second gain apparaît lorsque nous utilisons un paramètre spécifique : *mineltsize*. Ce paramètre donne un nombre minimal de nœuds par strate dans un connecton, le but étant de conserver un voisinage suffisant sur chaque réseau. Avec ce paramètre, dès que, sur une strate, la contrainte n'est pas vérifiée, on peut élaguer la branche courante, ce qui nous épargne le prolongement d'un grand nombre d'empilements et la construction d'un grand nombre d'arêtes.

Nous allons commencer par donner le pseudocode de l'algorithme global, appellé OTF (pour "on the fly", c'est-à-dire "à la volée") avant d'expliciter la procédure - *Prolonger* - qui permet d'ajouter une strate au multigraphe courant.

4.2.1 Squelette de l'algorithme

Il s'agit d'un algorithme récursif, que nous allons lancer avec comme arguments le multigraphe $(\mathcal{V}_1, \mathcal{E}_1)$ formé de l'ensemble des nœuds et arêtes du premier réseau et l'entier 1 identifiant la strate courante. Le pseudocode est donné ci-dessous (algorithme 3).
Rappelons les deux phases alternées dans l'algorithme :

– phase de partitionnement sur les i premières strates,

– phase d'expansion sur une nouvelle strate.

Si la première phase divise le multigraphe courant en plusieurs classes ($|P| \neq 1$), l'algorithme OTF se lance récursivement sur chacune de ces classes.
Si par contre elle résulte en une unique classe, l'algorithme passe à la seconde phase.

Algorithm 3: OTF.
Global: Graphe de données stratifié $D = (V, E)$ pour les réseaux $G_k = (V_k, E_k)$, $k \in [|1, n|]$
Input: Multigraphe C /* *classe courante : initialisée à* $(\mathcal{V}_1, \mathcal{E}_1)$ */
Entier k /* *strate courante : initialisée à* 1 */
Variables: Partition P /* *partition courante sur les* k *premières strates* */

```
(1)   begin
(2)     /* partitionner sur les k premières strates */
(3)     P ← Partitionner_{1...k}(C) ;
(4)     if (|P| ≠ 1)then
(5)       /* C a été partitionné : recurrence*/
(6)       for NewC ∈ P do
(7)         OTF(NewC, i)
(8)       end for
(9)     else if (k < n)then
(10)      /* nous arrivons sur un réseau qui n'a pas encore été
            traité : il faut construire une strate supplémentaire */
(11)      NC ← Prolonger(C, k + 1) ;
(12)      OTF(NC, k + 1) ;
(13)    else
(14)      /* C est stable */
(15)      PRINT(C)
(16)    end if
(17)  end
```

Celle-ci consiste à ajouter une strate supplémentaire au multigraphe courant, tant que c'est possible (nous expliquons plus en détail cette phase dans la partie suivante). Lorsque ce n'est plus possible, le multigraphe courant est renvoyé comme résultat.

Notons que le partitionnement sur l'ensemble des k couleurs (ligne 3) est nécessaire car l'ajout d'une couleur $i + 1$ ne garantit pas que le multigraphe reste connecté sur ces i couleurs.

En effet, certains empilements de taille i ont pu ne pas être prolongés en empilements de taille $i + 1$, ils disparaissent du nouveau multigraphe qui peut donc se retrouver déconnecté sur les i premières couleurs. Une autre raison est que le multigraphe n'est pas nécessairement connexe sur la dernière couleur. Lorsqu'on va Partitionner sur cette couleur, les autres couleurs peuvent donc se trouver

déconnectées à leur tour. [1]

Nous allons maintenant détailler dans la section suivante la procédure *Prolonger*, qui prolonge le multigraphe courant sur une nouvelle strate.

4.2.2 Le cœur de l'algorithme : la procédure *Prolonger*

La procédure PROLONGER prend en argument le multigraphe courant et l'indice du réseau primaire à ajouter. Nous avons aussi accès aux informations globales, c'est-à-dire au graphe de données stratifié complet.

Le but est de passer d'un multigraphe $C = (\mathcal{V}, \mathcal{E}_1, \ldots, \mathcal{E}_k)$ à un multigraphe $NewC = (\mathcal{V}', \mathcal{E}'_1, \ldots, \mathcal{E}'_{k+1})$. $NewC$ va vérifier les mêmes contraintes que C, et chacun des empilements de $\mathcal{V}'$ correspondra à un empilement de $\mathcal{V}$ prolongé par un nœud de V_{k+1}.
Nous donnons le pseudocode de *Prolonger* ci-dessous (algorithme 5).
De la ligne 3 à la ligne 9, la procédure commence par construire l'ensemble $\mathcal{V}'$ des empilements prolongés (un empilement $(v_1, \ldots v_k)$ va être prolongé par tous les $v_{k+1} \in V_{k+1}$ vérifiant une certaine propriété que nous allons préciser), puis la ligne 11 de la procédure procède à la construction des arêtes entre les nouveaux empilements, le multigraphe obtenu est ensuite renvoyé en résultat.

Commençons par détailler la construction de l'ensemble $\mathcal{V}'$ et notamment le rôle de la propriété $Pk + 1$: intuitivement, pour un empilement donné $v = (v_1, \ldots v_k) \in \mathcal{V}$, l'objectif est de ne "rater" aucun prolongement $v_{k+1} \in V_{k+1}$ qui pourrait mener à un n-empilement $(v_1, \ldots v_n) \in \mathcal{R}(V_1 \times V_2 \times \ldots \times V_n)$.

La façon de procéder va dépendre du type d'empilement considéré. Dans le cas des empilements-cliques, il est évident qu'il suffit de prolonger $(v_1, \ldots v_k)$, qui par construction forme une clique, par tous les nœuds $v_{k+1} \ni V_{k+1}$ qui sont liés à chacun des v_i, ..., v_k.

Nous présentons la condition de prolongement (testée ligne 5 de l'algorithme) sous la forme d'une propriété P_{k+1} : si un nœud $v_{k+1} \in V_{k+1}$ vérifie la propriété $P_{k+1}(v_1, \ldots, v_k, v_{k+1})$, alors on ajoutera $(v_1, \ldots, v_k, v_{k+1})$ à $\mathcal{V}'$.

1. Notons que, dans l'implémentation, la première phase n'est vraiment effectuée que lorsqu'elle est nécessaire, c'est-à-dire lorsque l'on vient d'ajouter une strate (c'est l'appel de la ligne 12 : $OTF(NC, i+1)$), que les nœuds de C n'ont pas tous été prolongés en nœuds de NC ou que la dernière strate ne forme pas une composante connexe unique.

Algorithm 4: Prolonger.

Global: Graphe de données stratifié $D = (V, E)$ pour les réseaux $G_i = (V_i, E_i)$, $i \in [|1, n|]$

Input: Multigraphe $C = (\mathcal{V}, \mathcal{E}_1, \ldots, \mathcal{E}_k)$ /* *multigraphe courant à prolonger* */

Output: Multigraphe $NewC = (\mathcal{V}', \mathcal{E}'_1, \ldots, \mathcal{E}'_{k+1})$ /* *multigraphe prolongé sur la strate* $k+1$ */

(1) **begin**
(2) /* *pour chaque nœud de* C *sélection des prolongements* */
(3) **for** $(v_1, \ldots, v_k) \in \mathcal{V}$ **do**
(4) **for** $v_{k+1} \in V_{k+1}$ **do**
(5) **if** $P_{k+1}(v_1, \ldots, v_k, v_{k+1})$ **then**
(6) $\mathcal{V}' \leftarrow \mathcal{V}' + \{v_{k+1}\}$;
(7) **end if**
(8) **end for**
(9) **end for**
(10) /*réintroduire les arêtes en respectant le graphe de données stratifié*/
(11) $(\mathcal{E}'_1, \ldots, \mathcal{E}'_{k+1}) \leftarrow ReconstruireAretes(\mathcal{V}', D)$;
(12) **return** $(\mathcal{V}', \mathcal{E}'_1, \ldots, \mathcal{E}'_{k+1})$;
(13) **end**

Propriété de Prolongement 1. ***(Empilements-Cliques)***
$Pclique_{k+1}(v_1, \ldots, v_k, v_{k+1}) = \forall i \in [|1, k|],\ (v_i, v_{k+1}) \in E_S$.

Dans le cas des empilements-étoiles, il suffit de prolonger $(v_1, \ldots v_k)$ par les nœuds $v_{k+1} \in V_{k+1}$ tels que v_{k+1} est lié au réseau central, c'est à dire $(v_1, v_{k+1}) \in E_S$. La propriété de prolongement s'écrit donc simplement :

Propriété de Prolongement 2. ***(Empilements-Étoiles)***
$Petoile_{k+1}(v_1, \ldots, v_k, v_{k+1}) = (v_1, v_{k+1}) \in E_S$.

Le cas des empilements-connexes (ou γ-denses) est un peu plus complexe puisque la propriété de connexité doit être vraie à la fin ($k = n$) mais pas nécessairement à un rang k quelconque. Rien n'empêche en effet que $(v_1, \ldots, v_k)$ soit complètement déconnecté, sans aucune correspondance, mais que cette correspondance soit assurée sur la dernière strate par un v_n relié à v_i, $\forall i \in [|1, n-1|]$.

La propriété de prolongement que nous utilisons s'écrit :

Propriété de Prolongement 3. ***(Empilements-Connexes)***
$Pconnexe_{k+1}(v_1, \ldots, v_k, v_{k+1}) = \exists \{v_{k+2}, \ldots v_n\} \in V_{k+2} \times \ldots V_n$ *tels que* $(v_1, \ldots v_n)$ *est connexe.*

Dans le cas des prolongements des empilements-cliques et des empilements-étoiles, le calcul des prolongements est donc très simple, il suffit d'un parcours des nœuds formant l'empilement et d'un test sur leurs correspondances avec V_{k+1}.

En pratique, la ligne (4) n'est donc pas un parcours sur tous les éléments de V_k mais simplement sur les voisins (suivant E_S) des v_i.

Par contre pour les empilements-connexes le calcul est plus complexe. Afin de l'accélérer, nous procédons dans un premier temps à un pré-traitement du graphe de données stratifié $D = (\bigcup_i V_i,\ E_P \cup E_S)$, dans lequel nous pré-calculons toutes les composantes connexes pour E_S.
Au moment de la recherche de prolongements, nous allons calculer l'intersection entre la composante connexe pour E_S contenant $\{v_1, \dots v_k\}$ et la $(k+1)$-ème strate. Il ne nous reste plus qu'à chercher, pour chacun des nœuds de l'ensemble obtenu, à construire un empilement-connexe le contenant et contenant aussi les v_i, $i \in [|1, k|]$. Ceci est réalisé par un parcours simple des voisins de v_{k+1} pour E_S en forçant à ne passer qu'une fois par chaque strate, et à passer par les v_i sur les k premières strates. Si l'on y parvient, c'est que l'empilement-connexe existe, que la condition nécessaire est vérifiée, nous pouvons donc prolonger l'empilement par v_{k+1}.

Ce processus est décrit dans la Figure 4.4.

Pour les empilements-γ-denses, nous allons commencer par tester une condition nécessaire, mais non suffisante, pour que l'empilement final soit une γ-quasi-clique. L'idée est d'imposer que dans le graphe induit par $(v_1, \dots v_{k+1})$, le degré de v_{k+1} soit suffisant pour que, une fois complété par les $n - k - 1$ strates restantes, l'empilement soit une γ-quasi-clique.
Formellement, cela veut dire que $(degré(v_{k+1}) + n - k - 1) \geq \gamma \times (n-1))$, ce qui peut s'écrire aussi de la façon suivante : $degré(v_{k+1}) \geq (\gamma \times (n-1) - (n-k-1))$. Si cette condition est vérifiée, il ne nous reste plus qu'à exhiber un n-uplet $(v_1, \dots v_n)$ connexe, le processus est exactement le même que dans le cas des empilements-connexes :

Propriété de Prolongement 4. ***(Empilements-γ-Denses)***
$Pdense_{k+1}(v_1, \dots, v_k, v_{k+1}) = (degré(v_{k+1}) \geq \gamma \times (n-1) - (n-k-1))$ et $\exists \{v_{k+2}, \dots v_n\} \in V_{k+2} \times \dots V_n$ tels que $(v_1, \dots v_n)$ est connexe,

4.2.2.1 Remarque sur les prolongements

En pratique, dans le cas des empilements-cliques, des empilements-étoiles et des empilements-γ-denses, notre implémentation revient à rechercher au rang k une condition nécessaire mais non-suffisante pour que le $k+1$-uplet soit prolongé au final en un n-empilement, cette condition devenant nécessaire et suffisante au

rang n. Par contre dans le cas des empilements-connexes, la condition au rang k est déjà nécessaire et suffisante pour que le $k+1$-uplet soit prolongé en n-empilement.

Prenons quelques instants pour expliquer ces choix.

Pour une définition d'empilements choisie, appellons **prolongement gagnant** de $(v_1, \dots v_k)$ tout $v_{k+1} \in V_{k+1}$ tel que on peut compléter $(v_1, \dots v_{k+1})$ en un n-uplet de $\mathcal{R}(V_1 \times V_2 \times \dots \times V_n)$.

Pour assurer l'exactitude du calcul, il suffit de garantir que nous ne ratons aucun prolongement gagnant, et qu'une fois parvenus à la dernière strate les n-uplets sont par construction des empilements du type choisi.

En pratique, calculer uniquement les prolongements gagnants de $(v_1, \dots v_k)$ serait idéal pour limiter la taille du multigraphe au maximum, mais leur calcul exact peut être coûteux, suivant les types d'empilements il peut donc être avantageux de se contenter d'un sur-ensemble V' des prolongements gagnants. Cela revient à trouver un équilibre entre le temps de calcul nécessaire pour calculer les prolongements et l'augmentation du multigraphe courant due aux prolongements.

Pour les empilements-cliques, les empilements-étoiles et les empilements-γ-denses, les empilements gagnants n'étant "pas trop nombreux", nous avons fait le choix de sélectionner un ensemble intermédiaire, qui soit un sur-ensemble des prolongements gagnants sans pour autant être la strate entière. Par contre, dans le cas des empilements-connexes, définition qui correspond au nombre maximum d'empilements, nous avons fait le choix de chercher exactement les prolongements gagnants afin de limiter la taille du multigraphe.

4.2.2.2 Reconstruction des arêtes

Une fois les prolongements calculés pour chacun des k-uplets de $\mathcal{V}$, l'ensemble $\mathcal{V}'$ est construit, mais il nous reste à construire les ensembles d'arêtes $\mathcal{E}'_1, \dots, \mathcal{E}'_{k+1}$ par la procédure *ReconstruireAretes*.
Nous voulons que $NewC$ respecte le graphe de données stratifié, c'est-à-dire que pour $U = (u_1, u_2, \dots, u_{k+1}) \in \mathcal{V}'$ et $V = (v_1, v_2, \dots v_{k+1}) \in \mathcal{V}'$, $(U, V) \in \mathcal{E}'_i$ si et ssi $(u_i, v_i) \in E_i$ ou $u_i = v_i$. La construction est donc directe, il suffit pour chaque paire de nœud (U, V) de $\mathcal{V}'$ d'aller consulter les nœuds associés sur chacune des $k+1$ premières strates du graphe de données. Si pour une strate donnée i les nœuds sont, soit identiques, soit reliés, il faut mettre une arête (U, V) dans $\mathcal{E}'_i$.

En pratique, notre implémentation est plus efficace : nous conservons une relation de parenté entre nœuds des deux multigraphes C et $NewC$, c'est-à-dire qu'on garde dans un nœud de $NewC$ la trace du "père" dont il est issu et, pour un nœud

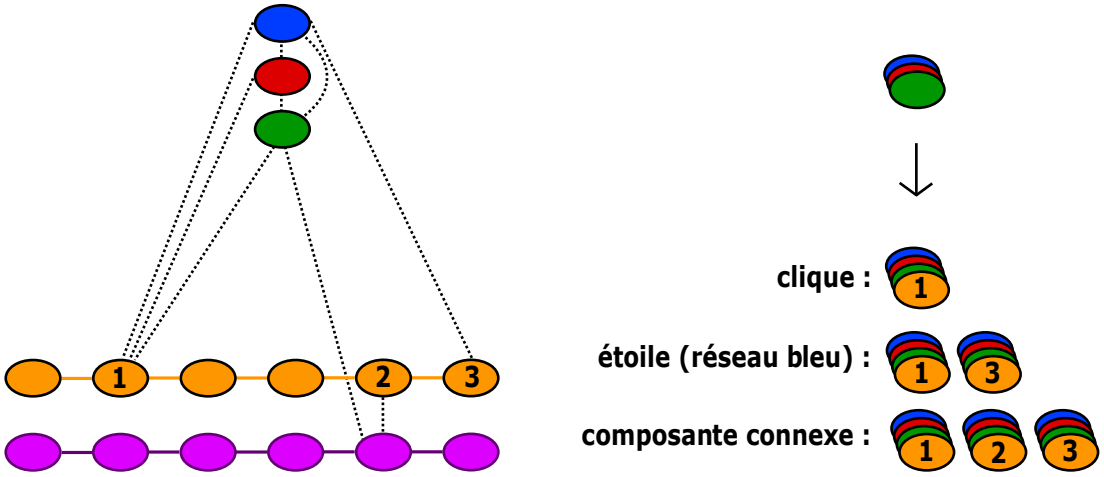


FIGURE 4.3 – Exemple de sélection de prolongements. Nous avons un triplet déjà construit, que nous cherchons à prolonger sur une nouvelle strate, de couleur orange. a) Pour les empilements-cliques, seul le nœud 1 de la strate orange est pertinent, parce qu'il est connecté à chacun des nœuds du triplet. b) Pour les empilements-étoile, nous obtenons les deux nœuds 1 et 3, parce qu'ils sont tous deux reliés par une arête pointillée au nœud bleu du triplet. c) Avec des empilements-connexes, nous devons récupérer uniquement le nœud 2 parce que c'est le seul prolongement gagnant.

de C, la trace de tous ses "fils", on injecte les arêtes de C qui, par construction, respectent le graphe de données, puis nous les complètons par les nouvelles arêtes dues à l'introduction de la nouvelle strate.

Nous donnons, sur la Figure 4.5, un exemple d'exécution d'OTF sur un graphe de données stratifiées représentant trois génomes.

4.2.3 Preuve de l'algorithme

La terminaison de l'algorithme est évidente puisque les procédures *Partitionner* et *Prolonger* se terminent et qu'à chaque appel d'OTF le multigraphe est soit prolongé sur une nouvelle strate, soit partitionné. Cela implique quà chaque appel soit le nombre de strates restantes diminue (et ce nombre est borné), soit la taille du multigraphe courant diminue (et elle est aussi bornée).

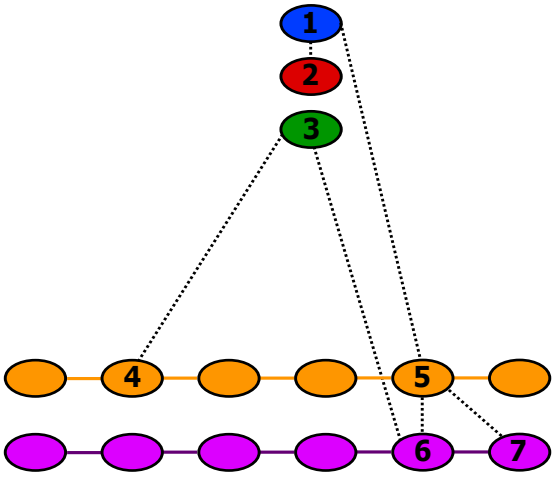


FIGURE 4.4 – Exemple de sélection de prolongements pour un empilement-connexe. Les deux seuls nœuds appartenant à l'intersection entre la composante connexe pour E_S contenant notre empilement courant et la quatrième strate sont les nœuds 4 et 5. Pour chacun de ces deux nœuds nous parcourons les arêtes de correspondance - en pointillés - en cherchant un empilement-connexe contenant les nœuds 1 2 et 3. Nous échouons pour le nœud 4 car on doit nécessairement repasser par un autre nœud de la même strate, mais nous réussissons pour le nœud 5 : le 5-uplet $(1, 2, 3, 5, 6)$ est connexe, 5 est donc un prolongement gagnant.

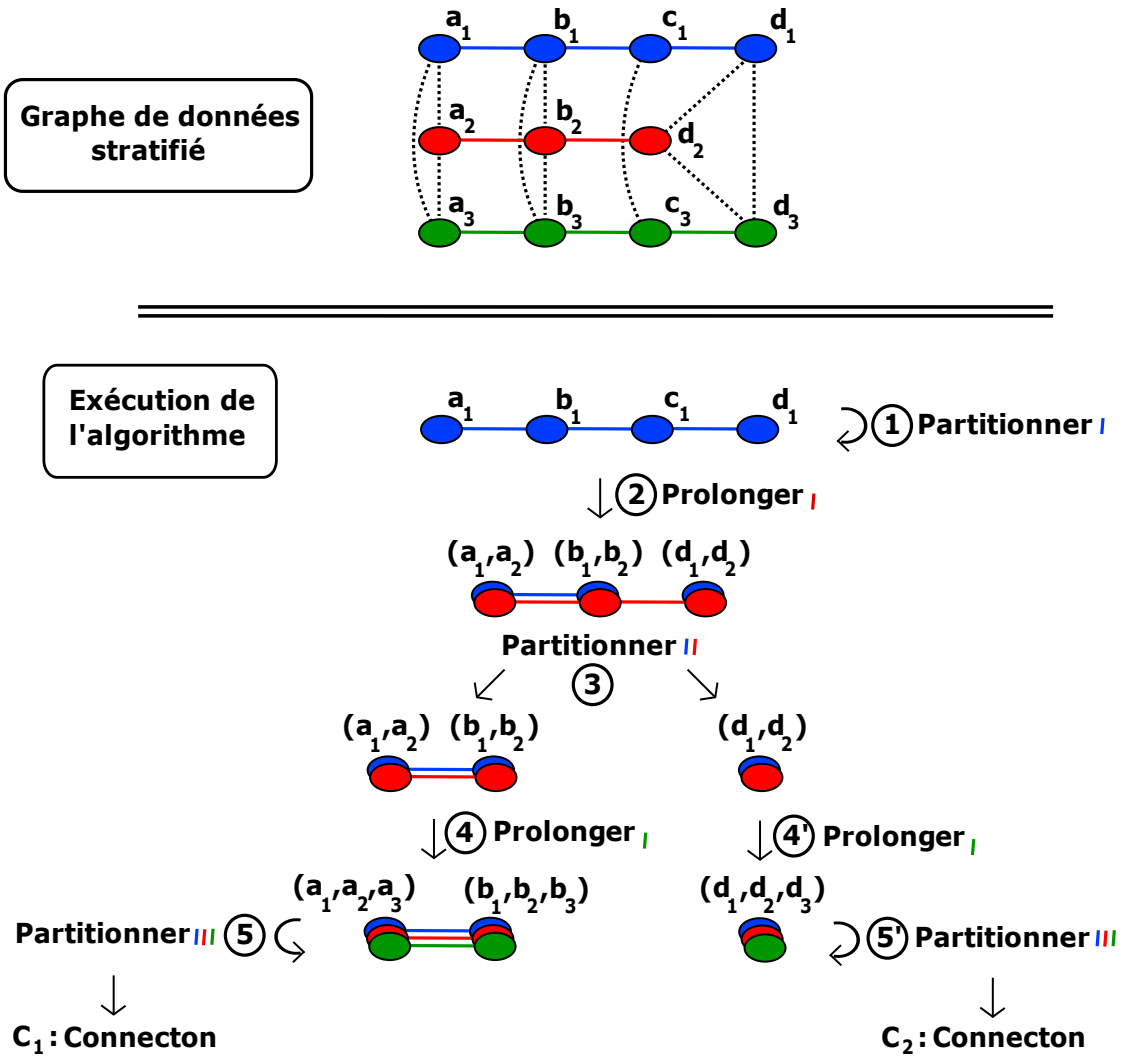


FIGURE 4.5 – Exemple d'exécution d'OTF. En haut nous donnons le graphe de données stratifié, représentant trois génomes. La relation de correspondance est représentée par des pointillés. OTF alterne des phases de partitionnement $(1, 3, 5, 5')$ et d'ajout de strate $(2, 4, 4')$. On remarque dans l'étape 3 qu'il faut bien partitionner sur toutes les couleurs courantes (bleu et rouge), puisque l'ajout d'une strate peut venir rompre la connexité sur n'importe quelle couleur : c'est le cas ici, le nœud c_1 n'a pas été prolongé, ce qui a rompu la connexité pour la couleur bleue. C_1 et C_2 sont des connectons, puisqu'on est parvenus à la dernière couleur et qu'ils sont stables pour l'opération de partitionnement.

Montrons maintenant qu'à tout instant intermédiaire dans le cours de l'exécution de l'algorithme, le multigraphe courant $C = (\mathcal{V}, \mathcal{E}_1, \ldots, \mathcal{E}_k)$ vérifie les deux propriétés suivantes :

- $\mathcal{V} \subseteq (V_1 \times V_2 \times \ldots \times V_k)$ et si $k = n$, $\mathcal{V} \subseteq \mathcal{R}(V_1 \times V_2 \times \ldots \times V_n)$,
- pour $U = (u_1, u_2, \ldots, u_k) \in \mathcal{V}$ et $V = (v_1, v_2, \ldots v_k) \in \mathcal{V}$, on a $(U, V) \in \mathcal{E}_i$ si et seulement si $(u_i, v_i) \in E_i$ ou $u_i = v_i$.

Ceci est évident pour le multigraphe courant initial $(\mathcal{V}_1, \mathcal{E}_1)$. Vérifions que cela reste vrai au cours de l'exécution. Les deux procédures *Partitionner* et *AjouterStrate* prennent en argument un multigraphe courant vérifiant ces contraintes, et rendent en résultat un ou plusieurs multigraphes les vérifiant aussi. En effet, dans le cas de *Partitionner*, les contraintes sont satisfaites naturellement puisque les résultats sont des sous-graphes du multigraphe courant. Pour *AjouterStrate* c'est le cas aussi, la première contrainte est garantie par la sélection des prolongements, et la seconde par la construction des arêtes.

Ces contraintes impliquent que, dans une branche donnée de notre algorithme, une fois parvenus à la dernière strate, le multigraphe courant C est bien un sous-graphe du multigraphe d'alignement. On sait aussi qu'il est n-connexe puisque c'est un point fixe de *Partitionner*.

Il nous reste à montrer que C est maximal. Ceci est garanti en pratique par deux propriétes :

1. *Partitionner* ne sépare jamais deux k-empilements qui peuvent être prolongés en des n-empilements d'un même connecton,
2. *Prolonger* construit nécessairement tous les prolongements gagnants.

Procédons par l'absurde. S'il n'était pas maximal, on pourrait lui ajouter un nombre d'empilements sans briser sa n-connexité. La seconde propriété implique qu'on les a construit dans une branche de notre algorithme. En supprimant au fur et à mesure les $(n-1)$ dernières couleurs de ces empilements, on retrouve donc à un moment de notre exécution ces empilements dans le même multigraphe courant que les empilements de C (dans le pire des cas c'est sur la première strate). La première propriété assure ensuite que ces empilements ne sont plus jamais séparés des empilements de C, ce qui est absurde puisqu'ils ne sont pas dans C.

C est donc un connecton, reste à montrer qu'on les obtient bien tous.

Procédons une fois encore par l'absurde, soit un connecton C' raté par l'algorithme. Supprimons au fur et à mesure les $(n-1)$ dernières couleurs de ce multigraphe en commençant par la fin. On sait qu'à un moment nous allons retrouver un sous-graphe inclus dans un multigraphe courant rencontré dans l'exécution de

notre algorithme, dans le pire des cas, cela sera le sous-graphe réduit à la première couleur. Les propriétés données plus haut garantissent ensuite que les empilements du connecton seront construits et qu'ils ne seront pas séparés : ce connecton C' supposé raté est finalement bien retrouvé par notre algorithme.

Les résultats de notre algorithme sont donc tous les connectons du MGA.

4.3 Améliorations et variantes

4.3.1 Pré-traitement

Une première amélioration technique de l'algorithme réside dans la suppression des nœuds inutiles du graphe de données stratifié donné en entrée.

Intuitivement, une condition nécessaire pour qu'un nœud du graphe de données stratifié appartienne à un empilement est le fait qu'il appartienne à une composante connexe de la relation de correspondance S qui passe par toutes les strates.

Nous allons donc procéder de la façon suivante :
- calculer les composantes connexes de la relation de correspondance (S) du graphe de données stratifié,
- pour chaque composante connexe, calculer le nombre de strates touchées
- supprimer les nœuds des composantes connexes touchant moins de n strates.

En pratique, ce pré-traitement permet de diminuer la taille du graphe de données et le nombre de nœuds des multigraphes courants, dans le cas où la condition de prolongement n'est pas nécessaire et suffisante.

Notes d'implémentation.
Dans l'implémentation, le paramètre correspondant est appelé **cleanup**, il a deux valeurs : *on* et *off*, suivant si l'on souhaite effectuer le pré-traitement ou pas.

4.3.2 Optimisation de l'ordonnancement des strates

L'ordre des réseaux ne joue pas sur l'exactitude du résultat mais par contre, il peut influer sur le temps de calcul : plus on partitionne tôt, plus on est efficace.

On peut s'en convaincre sur un exemple extrêmement simple : considérons un graphe de données stratifié tel qu'une strate soit complètement déconnectée des autres. Si l'algorithme commence par cette strate, le temps de calcul est très

court, on n'a de prolongement pour aucun des nœuds du multigraphe courant, et on reporte à l'utilisateur qu'il n'y a pas de connectons. Par contre si l'algorithme traite cette strate en dernier, il va construire potentiellement un grand multigraphe d'alignement avant de se rendre compte sur la dernière strate que c'était inutile.

L'intuition est simple : plus la nouvelle strate est éloignée au sens de la relation de correspondance des strates déjà traitées, moins on aura de prolongements possibles, et donc plus on a de chances d'éliminer des nœuds du multigraphe courant, ce qui peut permettre de le partitionner.

Une première idée d'optimisation de l'ordonnancement des strates est donc la suivante :

- calculer une matrice associant à chaque couple de strates le nombre de correspondances les reliant,
- choisir comme première et seconde strates les deux strates liées par le plus petit nombre de correspondances,
- choisir ensuite systématiquement la strate minimisant le nombre de correspondances avec les strates déjà choisies.

Ce type d'optimisation de l'ordre des strates (couleurs) est dit "global" puisqu'on calcule une fois pour toutes avant le début de l'algorithme l'ordonnancement des couleurs.

En pratique on se rend compte que ce type d'ordonnancement global n'est pas toujours efficace : on peut malgré tout, dans une branche de l'algorithme particulière, se retrouver avec de nombreux prolongements sur une strate donnée, même si globalement elle semblait pertinente. L'idéal serait donc d'ordonnancer localement, à l'intérieur de chaque branche.

C'est cette seconde optimisation que nous allons développer maintenant. Elle consiste à ordonnancer spécifiquement pour le multigraphe courant, et au dernier moment possible, en exploitant la structure de notre algorithme [2].

Nous choisissons notre première strate de la même manière que précédemment (une des 2 qui ont le nombre de correspondances minimal), puis à l'intérieur de l'algorithme, à la ligne 10, nous procédons à un ordonnancement à la volée. Le mécanisme est le suivant :

- calculer pour chaque strate restante le nombre de correspondances avec le multigraphe courant,
- choisir la strate minimisant ce nombre.

Cette heuristique locale vise à minimiser le nombre de prolongements des nœuds

2. Cela implique que dans les différentes branches de l'algorithme, nous n'allons probablement pas traiter les strates dans le même ordre, et qu'il faut donc réordonner les résultats à la fin pour qu'ils soient exploitables.

du multigraphe courant. En pratique, cela va nous permettre de limiter la taille du nouveau multigraphe, et, souvent, de partitionner plus tôt (parce que certains nœuds ne seront pas prolongés).

Notes d'implémentation.
Dans l'implémentation, le paramètre traitant l'optimisation de l'ordonancement des strates est appelé **optimizer**. Il peut prendre les valeurs suivantes :

- *off* : on conserve l'ordre dans lequel les réseaux sont fournis,
- *user* : on utilise un ordre spécifié par l'utilisateur,
- *global* : on utilise l'heuristique d'ordonnancement global,
- *local* : on utilise l'heuristique d'ordonnancement local.

4.3.3 Calcul de la correspondance à la volée

Lorsqu'on se trouve confronté à un problème d'alignement multiple local de génomes, un des outils classiques utilisés pour établir la correspondance entre gènes des différents génomes est le programme BLAST.

Pour construire un multigraphe d'alignement, il faut donc lancer BLAST sur chaque paire de génomes. En pratique sur une dizaine de génomes cela peut prendre quelques heures de calcul. Or parmi ces informations de correspondance, certaines ne seront pas utilisées par l'algorithme. L'exemple le plus simple est le cas d'empilements en étoile centrée sur le premier réseau, mais même pour des alignements en clique, beaucoup de correspondances peuvent s'avérer superflues.

Supposons par exemple que nous ayons dans le multigraphe courant un k-uplet $(v_1, \ldots v_k)$ qui n'apparaisse que dans cette branche de l'exécution, et qu'on constate qu'il n'a pas de prolongements. Dès lors, toutes les correspondances potentielles entre les v_i et les nœuds des strates restantes (de $k+2$ à n) sont inutiles.

De la même façon, si on ne recherche que les connectons ayant au moins X nœuds par strate (paramètre **mineltsize**), dès que nous avons un multigraphe courant qui ne convient pas, nous l'abandonnons, et nous n'avons donc plus besoin des connections entre les $(v_1, \ldots v_k)$ qui le constituaient et les nœuds des strates restantes.

Une extension logique de notre algorithme serait donc d'opérer le calcul de ces correspondances à la volée, au moment où elles sont nécessaires, c'est-à-dire au moment de la recherche des prolongements, à l'intérieur de la procédure $AjouterStrate$.

Nombre de génomes	2	3	4	5	6	7	8
Nombre de nœuds	9118	13590	17836	23261	27941	32393	36592
Nombre d'arêtes	13012	24287	39237	61588	78517	103963	133605
$C_3PartBFS$	1.3	5.9	NA	NA	NA	NA	NA
$C_3PartDFS$	1.3	5.9	NA	NA	NA	NA	NA
$OTF\ mineltsize = 1$	1.4	2	3.5	16.7	41.5	202	1266
$OTF\ mineltsize = 2$	1.3	1.5	1.9	2.3	2.3	3.3	4.3
$OTF\ mineltsize = 3$	1.2	1.3	1.4	1.7	1.6	2	2

TABLE 4.1 – Comparaison des temps d'exécution en secondes de l'algorithme original C_3Part à notre algorithme sur un ensemble croissant de génomes d'entérobactéries. Les nombres de nœuds et d'arêtes sont donnés pour le graphe de données stratifié. NA signifie que l'algorithme s'est arrêté à cause d'une consommation excessive de la mémoire ($> 1Gb$).

4.4 Analyse des performances en pratique : alignement multiple de génomes, comparaison à C_3Part

Nous avons implémenté les algorithmes décrits précédemment (BFS, DFS, OTF) en Java. Nous les avons testé sur huit génomes d'entérobactéries : *Citrobacter koseri*, *Enterobacter sp.*, *Erwinia carotovora*, *Escherichia coli,Klebsiella pneumoniae*, *Photorhabdus luminescens laumondii*, *Salmonella typhimurium* et *Shigella flexneri P. luminescens*, *S. flexneri*.
La relation S a été calculée en utilisant un seuil sur la P-value de BLAST : on introduit une arête entre deux gènes si et seulement si leur identié est supérieure à 40% (au niveau protéique) et l'alignement BLAST recouvre au moins 80% de la protéine la plus courte. Nous avons calculés les connectons pour différents nombres de génomes (de 2 à 8), les résultats sont donnés dans la Table 4.1.

On constate qu'OTF est clairement plus efficace lorsqu'on augmente le nombre de génomes. En pratique, pour des valeurs raisonnables de *mineltsize*, on observe que le temps de calcul augmente presque linéarement avec le nombre de réseaux, alors que la taille du multigraphe d'alignement augmente de façon exponentielle, et rend donc BFS et DFS inapplicables.

4.5 Analyse des performances en pratique : alignement multiple de réseaux PPI, comparaison à NetworkBLAST-M

Nous allons dans cette sous-partie détailler une application de notre algorithme dans le cadre de l'alignement multiple de réseaux d'interaction protéine-protéine. Afin de pouvoir évaluer la performance de l'algorithme, nous avons choisi d'utiliser le même ensemble de 10 réseaux PPI microbiens qui a été adopté dans plusieurs articles traitant d'alignement multiple de réseaux PPI. Ces 10 réseaux ne sont pas issus directement d'expériences, mais produits par un algorithme d'inférence SRINI, décrit dans [SNF+06]. SRINI génère un réseau d'interaction probabiliste en intégrant plusieurs sources d'information telles que la co-expression, la co-évolution ou encore la co-localisation sur le chromosome. Contrairement aux réseaux PPI expérimentaux, le résultat est un graphe complet dans lequel, à chaque paire de nœuds, est associée une probabilité d'interaction. Il faut donc choisir un seuil sur ces probabilités d'interaction pour retrouver un réseau PPI classique exploitable.

Pour à la relation de correspondance, nous avons utilisé les mêmes données que dans [SNF+06], *i.e.* une relation basée sur la similarité de séquence déterminée via BlastP avec un seuil Blast de 10^{-10} et en se limitant à 5 "hits" par protéine.

Nous avons choisi les empilements-cliques, imposant donc une similarité entre tous les couples de protéines au sein de notre empilement.

Les tailles des graphes de données stratifiés obtenus sont données dans la Table 4.2 pour différents nombres d'espèces sélectionnées (3, 5, 7 and 10 ; les espèces sélectionnées sont les mêmes que dans [SNF+06]).

La première observation que l'on peut faire est que OTF était en mesure de traiter ces grands réseaux alors que le calcul explicite du multigraphe d'alignement aurait été impossible pour C3Part (et NetworkBLAST) pour plus de 3 espèces.

Nous avons ensuite comparé nos résultats avec ceux obtenus par l'algorithme NetworkBLAST-M [KBS08], c'est-à-dire la version multiple de l'algorithme NetworkBLAST [SSK+05]. Comme nous l'avons mentionné auparavant (section 2.2.2.2), NetworkBLAST-M est un algorithme heuristique qui évite la construction du multigraphe d'alignement en définissant des n-spines dans le graphe de données stratifié (ce sont nos empilements) et en recherchant des ensembles de n-spines qui correspondent à des ensembles de nœuds "denses" sur chaque réseau.

Nous avons lancé NetworkBLAST-M à la fois en mode "relaxed" (où les n-spines sont définies comme des chemins avec une topologie identique), et dans

n	PPI Thresh.	# Prot	#Edges Simil.	#Edges PPI	Time (s) NBM (1)	Time (s) NBM (2)	Time (s) OTF	#Spine NBM	#Spine OTF	#Spine Common	#Prot NBM	#Prot OTF	#Prot Common
3	0.7	3751	1526	7450	1	1	4	91	194	76(84%)	235	339	207(88%)
	0.5	5322	2739	18511	2	2	6	169	457	128(76%)	413	628	354(86%)
	0.3	7826	4606	66484	9	5	12	291	972	195(67%)	679	1128	557(82%)
5	0.7	5910	4444	13061	12	2	6	82	261	58(71%)	337	370	266(79%)
	0.5	8169	7422	38645	34	3	9	168	566	73(43%)	599	656	390(65%)
	0.3	11404	11805	134437	175	12	19	201	1432	104(52%)	794	1231	562(71%)
7	0.7	8416	8721	18707	166	2	7	31	190	19(61%)	206	252	156(76%)
	0.5	12354	16897	60517	478	5	12	119	531	54(45%)	602	640	394(65%)
	0.3	16579	26452	211368	2832	17	38	193	2500	77(40%)	1002	1284	601(60%)
10	0.7	15411	21995	47340	9038	3	60	24	853	18(75%)	240	292	210(88%)
	0.5	23370	50758	173881	39644	10	74	111	2062	45(41%)	798	729	472(59%)
	0.3	30534	74126	616307	N/A	33	1143	N/A	13250	N/A	N/A	1613	N/A

TABLE 4.2 – Comparaison entre OTF et NETWORKBLAST-M (NBM) sur 10 espèces microbiennes. *PPI Thresh.* est le seuil de probabilité appliqué sur le réseau PPI complet ; *#Prot* est le nombre de protéines intervenant dans les empilements ; *#Edges Simil.* est le nombre d'arêtes de correspondance S ; *#Edges PPI.* est le nombre d'arêtes dans les réseaux ; les trois colonnes suivantes, intitulées "Time (s)", donnent les temps de calcul de NBM pour le mode "relaxed" (1) (où les n-spines sont des chemins avec une topologie identique) et pour le mode "tree-guided-path" (2) (où les n-spines sont des chemins compatibles avec un arbre phylogénétique donné en entrée), et les temps de calcul d'OTF. Dans le reste du tableau c'est le mode (2) que nous allons conserver. Les colonnes *#Spine* donnent le nombre de n-uplets trouvés par les différents algorithmes (empilements dans notre cas, n-spines pour NETWORKBLAST-M) ainsi que la proportion de n-uplets en commun. De la même manière, les colonnes *#Prot NBM* et *#Prot OTF* donnent les nombres de protéines présentes dans les résultats, puis *#Prot Common* donne la proportion de protéines retrouvées par les deux algorithmes.

le mode 'tree-guided-path' (où les n-spines sont définie comme des chemins compatibles avec un arbre phylogénétique donné en entrée) qui s'avère être beaucoup plus rapide [KBS08].

Le temps de calcul de OTF reste comparable à celui du mode "tree-guided-path", permettant dans la majorité des cas de calculer les connectons en quelques minutes.

Une comparaison plus intéressante entre les deux algorithmes est en terme de résultats. Puisque les choix de construction du multigraphe et de topologie des résultats diffèrent, nous ne pouvons pas comparer directement les complexes protéiques trouvés. Nous avons donc décidé de comparer les résultats en terme d'empilements retrouvés et de protéines intervenant dans ces empilements, c'est-à-dire les colonnes *#Spine Common* et *#Prot Common* du Tableau 4.2.

Malgré des philosophies différentes, on trouve que les deux algorithmes obtiennent des résultats très similaires, à la fois en terme d'empilements et de protéines. Pour 3 espèces, on pouvait s'y attendre puisque les conditions de clique et de chemin sont assez proches, OTF retrouve ainsi jusqu'à 84% des empilements et 88% des protéines obtenues par NetworkBLAST-M. Evidemment, la taille de ce recouvrement diminue lorsqu'on augmente le nombre d'espèces, mais reste remarquablement haut, même pour 10 espèces.

Une explication pour ce phénomène réside dans l'observation faite par Kalaev *et al.* [KBS08] que la majorité des empilements obtenus sont en fait des cliques.

Notre algorithme permet donc d'obtenir des résultat similaires à ceux obtenus par NetworkBLAST-M, avec des temps d'exécution raisonnables, et il comporte un avantage notable : nous avons séparé la procédure d'alignement du choix heuristique d'une fonction de score estimant la pertinence d'un point de vue biologique. Cela implique par exemple que notre approche peut être utilisée comme pré-filtrage pour d'autres outils plus spécialisés.

Chapitre 5

Alignements local multiple partiel

Jusqu'à présent, la définition que nous avons utilisée pour les alignements locaux est une définition fortement contrainte qui impose, pour n réseaux primaires, la présence systématique d'empilements de n nœuds.

Mais plusieurs éléments peuvent remettre en question ce modèle.

Le premier est la réalité biologique : prenons l'exemple de l'alignement de génomes de différentes espèces pour identifier des synténies, s'il y a eu délétion d'un gène dans une de ces espèces ou si un des gènes a tellement divergé qu'aucun lien de similarité n'existe plus, l'empilement correspondant ne sera pas retrouvé, et l'alignement risque d'être coupé en deux, voire d'être perdu si les alignements résultants sont trop petits.

Le second vient de la présence d'erreurs dans les bases de données. Toujours dans le domaine génomique, il peut s'agir d'erreurs de séquençage, auquel cas toutes les arêtes de correspondance du (ou des) nœud(s) où se situe l'erreur vont être incorrectes, ce qui rendra la construction de l'empilement impossible. Dans le cas de données métaboliques, en plus des erreurs de séquençage, il peut y avoir des erreurs d'annotation fonctionnelle des enzymes.

Afin de retrouver malgré tout des alignements locaux pertinents, nous relâchons dans cette partie la contrainte d'empilement en autorisant les empilements d'au moins q nœuds avec $2 \leq q \leq n$, q sera appelé le quorum.

En pratique, nous proposons deux définitions différentes d'**alignement locaux partiels**.

Dans la première Section, nous donnons le formalisme commun à ces deux définitions, en expliquant la notion de **nœud joker**, l'extension de la relation de correspondance au cas avec quorum, et le MultiGraphe d'Alignement Partiel

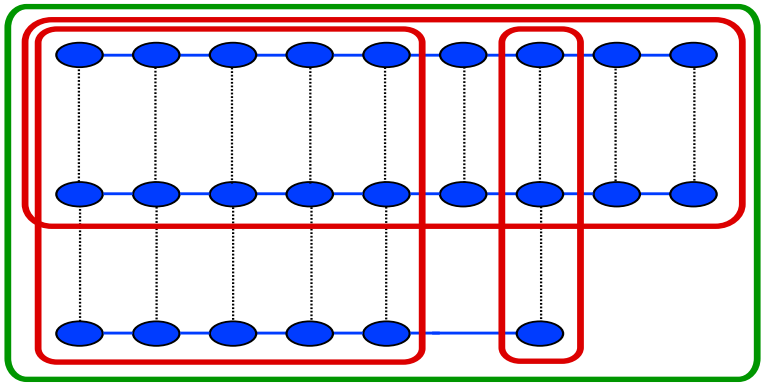


FIGURE 5.1 – Alignements partiels par strate (en rouge) et par empilement (en vert) dans un graphe de données stratifié correspondant à trois génomes.

($MGAP$).

Nous détaillons ensuite la première définition : les **alignements locaux partiels par strate**. Cette première définition conserve une contrainte forte sur les empilements : dans un alignement local partiel par strate, tous les empilements doivent contenir le même nombre de nœuds et ces nœuds doivent appartenir au même sous-ensemble de réseaux. Intutitivement cela revient à imposer que l'alignement local partiel par strate soit un alignement local sur un sous-ensemble $I, |I| \geq q$ des n réseaux et que les $n - |I|$ autres réseaux soient purement et simplement ignorés.

Enfin, nous donnons la seconde définition : les **alignements locaux partiels par empilement**, qui autorisent tous les empilements de $q \leq k \leq n$ nœuds, peu importe les réseaux sur lesquels ils se trouvent, du moment que la connexité sur chaque réseau reste respectée.

Dans la suite, par simplicité, nous parlerons d'alignements partiels par strate et partiels par empilement, la notion d'alignement local étant sous-entendue.

La Figure 5.1 donne un exemple d'alignements partiels par strate et par empilement pour un graphe de données stratifié représentant trois génomes.

5.1 Formalisation

5.1.1 Relation de correspondance multiple partielle

5.1.1.1 Introduction du nœud joker

Afin d'introduire un quorum, la première étape est de définir comment des nœuds issus de différents réseaux vont être empilés.

Intuitivement, l'introduction d'un quorum q revient à travailler avec des empilements de k nœuds (avec $q \leq k \leq n$) au lieu d'empilements de n nœuds. Afin de conserver un formalisme simple, nous allons, malgré tout, continuer à travailler avec des empilements de n nœuds en introduisant un nœud "joker", noté $*$.

Un empilement de k nœuds sera donc en pratique "complété" par $(n-k)$ nœuds jokers, ce qui nous permettra d'utiliser un formalisme similaire à celui du multigraphe d'alignement.

Pour construire ce formalisme, il nous faut définir ce qu'est la n-correspondance partielle.

5.1.1.2 n-correspondance partielle

Soit $V_i^* = V_i \cup \{*\}$. Nous commençons par définir une fonction *cover* qui à un empilement donné v associe l'ensemble des nœuds de v qui ne sont pas des jokers. Formellement :

$$\forall v = (v_1, v_2, \ldots, v_n) \in (V_1^* \times V_2^* \times \ldots \times V_n^*), cover(v) = \{v_i \neq *, i \in [|1, n|]\}.$$

Nous pouvons maintenant définir la n-correspondance partielle. Cela va être simplement un sous-ensemble du produit cartésien $(V_1^* \times V_2^* \times \ldots \times V_n^*)$ avec la condition supplémentaire que $|cover(v)| \geq q$.

Définition 5.1. *Etant donnés n réseaux primaires $G_i = (V_i, E_i)$, $i \in [|1, n|]$ et le graphe de données stratifié associé $D = (V, E)$, pour un quorum fixé q, nous appellerons* n-correspondance partielle *une n-correspondance $\mathcal{R}(V_1^* \times V_2^* \times \ldots \times V_n^*)$ telle que $\forall v \in \mathcal{R}(V_1^* \times V_2^* \times \ldots \times V_n^*)$, $|cover(v)| \geq q$. Nous utiliserons dans la suite la notation $\mathcal{R}_q(V_1^* \times V_2^* \times \ldots \times V_n^*)$, les éléments de cette n-correspondance seront appelés des* empilements partiels.

Notons que, comme la n-correspondance, la n-correspondance partielle est calculée en utilisant la relation S.

5.1.1.3 Définition des empilements partiels

Rappelons les différents empilements que nous avions détaillés dans la section 3.2 : empilement-connexe, empilement-étoile, empilement-clique, empilement-γ-dense, empilement-chemin, empilement-arbre.

Dans tous les cas, nous allons maintenant imposer que l'empilement partiel v vérifie $|cover(v)| \geq q$, et nous imposerons la contrainte (de connexité, de clique, d'arbre, *etc*) à l'ensemble des nœuds non-jokers, c'est-à-dire à $cover(v)$.

La Définition 5.2 regroupe les extensions de plusieurs types d'empilements : empilements-connexes, empilements-cliques et empilements-connexes-denses.

Définition 5.2. ***Empilement-Connexe (resp. Empilement-Clique, Empilement-γ-Dense) Partiel***
Etant donnés n réseaux primaires $G_i = (V_i, E_i)$, $i \in [|1, n|]$ et le graphe de données stratifié associé $D = (V, E)$, un n-uplet $(v_1, \dots v_n) \in (V_1^* \times V_2^* \times \dots \times V_n^*)$ est un empilement-connexe*(resp.* empilement-clique*,* empilement-γ-dense*) partiel si et seulement si $|cover(v)| \geq q$ et si le sous-graphe induit par $cover(v)$ est une composante connexe (resp. une clique, resp. une γ-quasi-clique connexe).*

De même, l'extension des empilements-arbres se fait naturellement.

Définition 5.3. ***Empilement-Arbre Partiel***
Etant donné le graphe de données stratifié $D = (\bigcup_i V_i,\ E_P \cup E_S)$, un n-uplet $(v_1, \dots v_n) \in (V_1^* \times V_2^* \times \dots \times V_n^*)$ est un empilement-arbre partiel *pour une arborescence donnée T si et seulement si pour tout clade $I \in \mathcal{C}(T)$, $cover(v_1, \dots v_n)_I$ admet un chemin pour E_S.*

Enfin les empilements-chemins partiels (Def. 5.4) sont définis comme les empilements tels que pour la permutation donnée en entrée, les nœuds non-jokers forment un chemin pour E_S.

Définition 5.4. ***Empilement-Chemin Partiel***
Etant donné le graphe de données stratifié $D = (\bigcup_i V_i,\ E_P \cup E_S)$, un n-uplet $(v_1, \dots v_n) \in (V_1^* \times V_2^* \times \dots \times V_n^*)$ est un empilement-chemin partiel *pour une permutation p des indices donnée si et seulement si $|cover(v)| \geq q$ et $\forall i,\ j \in [|1, n-1|]$, $(i < j,\ v_{p(i)} \neq *,\ v_{p(j)} \neq *,$ et $\forall k \in]i, j[|,\ v_{p(k)} = * \Rightarrow (v_{p(i)}, v_{p(j)}) \in E_S)$.*

Dans le cas des empilements-étoile (Def. 5.5), nous imposons une correspondance avec le réseau central, le nœud de ce réseau ne peut donc pas être un joker.

Définition 5.5. ***Empilement-Etoile Partiel***
Etant donnés le graphe de données stratifié $D = (\bigcup_i V_i,\ E_P \cup E_S)$, un n-uplet

$(v_1, \dots v_n) \in (V_1^* \times V_2^* \times \dots \times V_n^*)$ *est un* empilement-étoile *partiel si et seulement si* $|cover(v)| \geq q$, $v_1 \neq *$ *et* $\forall i \neq k$, $(v_1, v_i) \in E$ *ou* $v_i = *$.

A partir de la définition choisie pour les empilements partiels, nous avons donc formalisé $\mathcal{R}_q(V_1^* \times V_2^* \times \dots \times V_n^*)$, ce qui nous permet de définir maintenant le multigraphe d'alignement partiel ($MGAP$).

5.1.2 MultiGraphe d'alignement partiel ($MGAP$)

Le $MGAP$ est une extension du MGA (MultiGraphe d'Alignement). Il résume à la fois la n-correspondance partielle et la connexité sur les réseaux.

Définition 5.6. *Pour un graphe de données stratifié* $D = (V = \bigcup_i V_i,\ E = E_P \cup E_S)$ *et une* n*-correspondance partielle* $\mathcal{R}_q$*, le* multigraphe d'alignement partiel ($MGAP$) *est le graphe* $M = (\mathcal{V}, \mathcal{E}_1, \dots, \mathcal{E}_n)$ *tel que :*

- $\mathcal{V} = \mathcal{R}_q(V_1^* \times V_2^* \times \dots \times V_n^*)$
- $\forall u = (u_1, u_2, \dots, u_n) \in \mathcal{V}$ *and* $v = (v_1, v_2, \dots, v_n) \in \mathcal{V}$,
  $(u, v) \in \mathcal{E}_i \Leftrightarrow ((u_i, v_i) \in E_i \vee (v_i = u_i \neq *))$

En d'autres termes, les nœuds du multigraphe sont des n-uplets formés de nœuds des réseaux primaires et de nœuds jokers, et il y a une arête $e \in E'_i$ entre deux nœuds s'ils possèdent des nœuds non-joker à la position i qui sont connectés sur le $i^{ème}$ réseau ou qui sont identiques. Comme précédemment, nous ferons référence à ces arêtes comme des arêtes de "couleur" i.

On introduit ici quelques définitions utiles pour la suite.

Définition 5.7. *Un nœud* $(v_1, \dots v_n) \in \mathcal{V}$ *du MGAP est un* empilement complet *si et seulement si* $\forall i$, $v_i \neq *$.

Définition 5.8. *Un nœud* $(v_1, \dots v_n) \in \mathcal{V}$ *du MGAP est un* empilement i-incomplet *si et seulement si* $v_i = *$.

Définition 5.9. *Etant donné un ensemble* $\mathcal{V}' \subseteq \mathcal{V}$ *d'empilements du MGAP, on note* $\mathcal{V}_{*i}$ *l'ensemble des empilements* i*-incomplets de* $\mathcal{V}'$*. Inversement* $\mathcal{V}' - \mathcal{V}'_{*i}$ *sera donc l'ensemble des empilements de* $\mathcal{V}'$ *qui n'ont pas de joker en position* i.

La figure 5.2 donne un exemple simple de graphe de données stratifié pour trois réseaux, ainsi que le $MGAP$ correspondant pour des empilements-cliques partiels pour un quorum $q = 2$.

Munis de ces définitions, formalisons maintenant les deux types d'alignements partiels : par strate et par empilement.

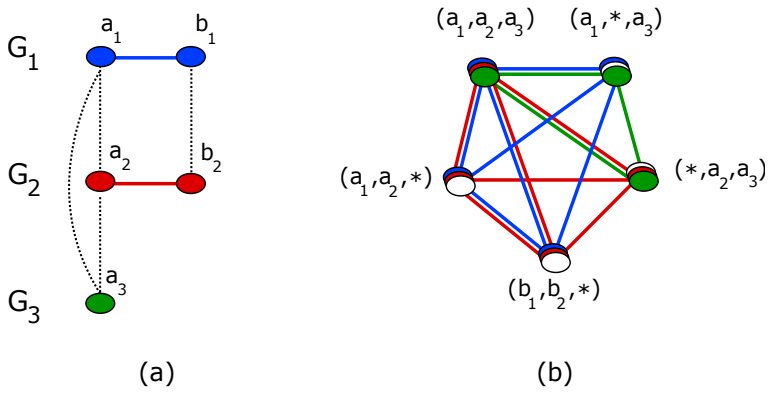


FIGURE 5.2 – Exemple de graphe de données stratifié pour trois réseaux primaires (a) et le $MGAP$ correspondant (b). La relation de correspondance S est représentée par des lignes pointillées.

5.2 Alignement partiel par strate

Informellement, l'idée est que, dans un connecton, les jokers sont joués sur les strates entières (en ce sens, la contrainte de quorum est imposée "globalement" à une strate).

5.2.1 Définition des alignements partiels par strate

Pour un sous-graphe $D' = (V', E')$ du graphe de données stratifié $D = (\bigcup_i V_i, E_P \cup E_S)$, notons D'_I la restriction de D' aux réseaux $i \in I$. Nous disposons aussi de la n-correspondance partielle $\mathcal{R}_q(V_1^* \times V_2^* \times \ldots \times V_n^*)$.
Notons $\mathcal{R}_{qI}$ les ensembles de $|I|$-uplets de nœuds des réseaux $i \in I$ qui peuvent être complétés en n-uplets de $\mathcal{R}_q(V_1^* \times V_2^* \times \ldots \times V_n^*)$.

Les alignements partiels par strate sont définis de la façon suivante.

Définition 5.10. *Les alignements partiels par strate q sont les sous-graphes maximaux D' de D tels que $\exists I \subseteq [|1, n|]$:*

1. $|I| \geq q$,
2. D'_I *est un alignement local*[1] *de* D_I *pour la* $|I|$*-correspondance* $\mathcal{R}_{qI}$,
3. $D'_{[|1,n|]-I} = \emptyset$.

Sur l'ensemble des strates d'indice pris dans I, ces alignements sont exactement les alignements locaux définis dans le Chapitre 3, et ils ne comportent aucun nœud

1. Au sens de la définition 3.13 du Chapitre 3

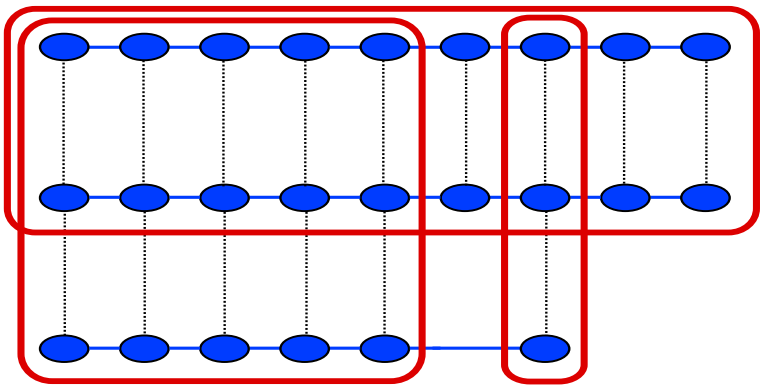


FIGURE 5.3 – Alignements partiels par strate dans un graphe de données stratifié correspondant à trois génomes.

sur les autres strates. La Figure 5.3 donne un exemple simple d'alignements partiels par strate dans le cas de trois réseaux représentant des génomes.

Nous pouvons maintenant donner la définition des objets correspondants dans le $MGAP$.

5.2.2 Définition des connectons partiels par strate

Dans le cadre de la définition du $MGAP$ que nous venons de donner, les connectons partiels par strate sont définis de la façon suivante.

Définition 5.11. *Un* connecton partiel par strate *est un sous-graphe maximal* $(\mathcal{V}', \mathcal{E}'_1, \ldots, \mathcal{E}'_n)$ *du multigraphe d'alignement partiel vérifiant la propriété suivante :* $\exists I \subseteq [|1, n|]$, $|I| \geq q$ *tel que* $\forall i \in I$, $(\mathcal{V}', \mathcal{E}'_i)$ *est connexe et* $\forall v = (v_1, \ldots v_n) \in \mathcal{V}'$, $cover(v) = \{v_i,\ i \in I\}$ [2].

Informellement, la première partie de la définition garantit que le résultat restreint aux couleurs de I va être un connecton [3], et la seconde partie garantit que sur les autres couleurs $i \notin I$, tous les empilements de V ont un nœud joker. Enfin, dans cette définition, le terme "maximal" signifie qu'aucun nœud supplémentaire du $MGAP$ ne peut être ajouté sans invalider une des deux conditions.

Comme précédemment, on remarque que les connectons partiels par strate forment une partition des nœuds du $MGAP$. Nous en donnons ci-dessous une démonstration :

2. Notons que cette condition impose l'unicité de l'ensemble I.
3. Au sens de la définition donnée dans le Chapitre 3.

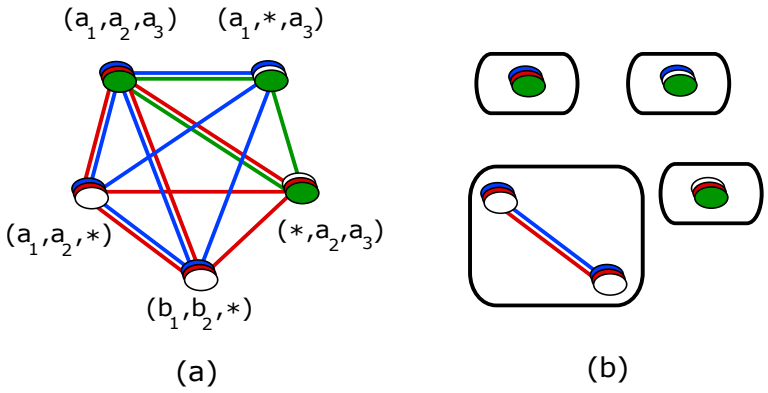


FIGURE 5.4 – Exemple de $MGAP$, et la partition correspondante en connectons partiels par strate.

Démonstration. Procédons par l'absurde.
Supposons que l'on ait extrait les connectons partiels par strate d'un $MGAP$ et qu'ils n'en forment pas une partition.
Cela implique qu'on aurait deux connectons partiels par strate $C_1 = (U_1, F_1, \dots F_n)$ et $C_2 = (U_2, F'_1, \dots F'_n)$ distincts, dont l'intersection ne serait pas nulle.
Nous allons maintenant montrer que $C = C_1 \cup C_2$ serait alors un connecton partiel, ce qui viendra contredire la maximalité de C_1. Prenons un nœud $v = (v_1, \dots v_n)$ dans $(U_1 \cap U_2)$, et appellons I l'ensemble d'indices tel que $cover(v) = \{v_i,\ i \in I\}$, on a $|I| \geq q$ puisque v est dans le connecton partiel C_1.
$v \in U_1$ implique que $\forall w = (w_1, \dots w_n) \in U_1$ $cover(w) = \{w_i,\ i \in I\}$.
De la même façon, $v \in U_2$ implique que $\forall w' = (w'_1, \dots w'_n) \in U_2$ $cover(w') = \{w'_i,\ i \in I\}$.
Enfin par définition des connectons partiels par strate, on sait aussi que $\forall i \in I$, les graphes (U_1, F_i) et (U_2, F'_i) sont connexes.
$C_1 \cap C_2 \neq \emptyset$ implique que $\forall i \in I$, $(U_1 \cup U_2, F_i \cup F'_i)$ est connexe.
Nous venons de montrer que $C = (U_1 \cup U_2, F_1 \cup F'_1, \dots F_n \cup F'_n)$ est un sous-graphe du $MGAP$ tel que $\exists I \subseteq [|1, n|]$, $|I| \geq q$ tel que $\forall i \in I$, $(U_1 \cup U_2, F_i \cup F'_i)$ est une composante connexe et $\forall v = (v_1, \dots v_n) \in U_1 \cup U_2$, $cover(v) = \{v_i,\ i \in I\}$. Cela vient contredire la maximalité de C_1 (et celle de C_2 d'ailleurs).
Les connectons partiels par strate forment donc bien une partition des nœuds du $MGAP$.

Nous donnons sur la figure 5.4 la partition du $MGAP$ utilisé dans l'exemple précédent en connectons partiels par strate.

Cette figure permet d'illustrer une propriété importante : les connectons partiels par strate ne correspondent pas tous à des alignements partiels par strate.

Sur la Figure 5.4 on se rend compte que les sous-graphes induits dans le graphe de données stratifié par les connectons $\{(a_1, *, a_3)\}$ et $\{(a_1, a_2, *)\}$ sont inclus dans le sous-graphe induit par le connecton $\{(a_1, a_2, a_3)\}$. Seul ce dernier correspond donc à un alignement partiel par strate q.

Notons $\sqsubseteq$ la relation d'inclusion entre nœuds du $MGAP$ définie de la façon suivante : $u \sqsubseteq v \Leftrightarrow cover(u) \subseteq cover(v)$.

Nous pouvons étendre cette relation aux sous-graphes du $MGAP$, pour deux sous-graphes du $MGAP$ $C_1 = (U_1, F_1, \dots F_n)$ et $C_2 = (U_2, F'_1, \dots F'_n)$, $C_1 \sqsubseteq C_2 \Leftrightarrow \forall u_1 \in U_1$, $\exists u_2 \in U_2$, tel que $u_1 \sqsubseteq u_2$.

Nous allons maintenant montrer que les alignements partiels par strate sont exactement les connectons partiels par strate maximaux pour la relation $\sqsubseteq$.

Démonstration. Nous allons commencer par montrer que les alignements partiels par strate sont des connectons partiels par strate maximaux pour $\sqsubseteq$, puis nous prouverons l'implication inverse.
Soit D' un alignement partiel par strate pour un quorum q. On sait que $\exists I \subseteq [|1, n|]$, $|I| \geq q$, D'_I vérifie la contrainte d'empilement pour la $|I|$-correspondance $\mathcal{R}_{q|I}$ de D_I et $D'_{[|1,n|]-I} = \emptyset$. On en déduit que D' vérifie la contrainte d'empilement pour R_q, ce qui implique donc que D' correspond exactement à $C = (\mathcal{V}', \mathcal{E}'_1, \dots \mathcal{E}'_n)$, un sous-graphe du $MGAP$. D'_I vérifiant de plus la contrainte de localité, on obtient que $\exists I \subseteq [|1, n|]$, $|I| \geq q$ tel que $\forall i \in I$, $(\mathcal{V}', \mathcal{E}'_i)$ est une composante connexe et $\forall v = (v_1, \dots v_n) \in \mathcal{V}'$, $cover(v) = \{v_i,\ i \in I\}$.On sait que C est un sous-graphe maximal du $MGAP$ parce que tout ajout à C se présenterait sous la forme d'un ensemble d'empilements de R_q, dont les sous-graphes induits dans le graphe de données stratifié pourraient être ajoutés à D' sans briser les contraintes sur D'_I : ce qui viendrait donc contredire sa maximalité. Nous venons de prouver que C est un connecton partiel, il nous reste à montrer que C est maximal pour $\sqsubseteq$.Procédons par l'absurde. Supposons qu'il existe un connecton partiel $C' = (\mathcal{U}, \mathcal{F}_1, \dots, \mathcal{F}_n)$ tel que $C \sqsubseteq C'$. Cette condition implique directement que le sous-graphe D'' induit par C' dans le graphe de données stratifié inclut D'. C' étant un connecton partiel on a ensuite l'existence d'un unique J, $|J| \geq q$ tel que $\forall i \in J$, $(\mathcal{U}, \mathcal{F}_i)$ est une composante connexe et $\forall v = (v_1, \dots v_n) \in \mathcal{U}$, $cover(v) = \{v_i,\ i \in J\}$. On en déduit que D''_J est un alignement local de D_J pour la $|J|$-correspondance $\mathcal{R}_{qJ}$ et $D''_{[|1,n|]-J} = \emptyset$. Ce qui vient contredire la maximalité de D'. Tout alignement partiel par strate correspond donc à un connecton partiel par strate maximal pour $\sqsubseteq$.
Il nous reste encore à montrer que tout connecton partiel par strate maximal pour $\sqsubseteq$ correspond bien à un alignement partiel par strate.
Soit $C = (\mathcal{V}', \mathcal{E}'_1, \dots \mathcal{E}'_n)$ un connecton partiel maximal pour $\sqsubseteq$. On déduit sans peine qu'il correspond à un sous-graphe D' de D tels que $\exists I \subseteq [|1, n|]$, $|I| \geq q$, D'_I est un alignement local de D_I pour la $|I|$-correspondance $\mathcal{R}_{qI}$ et $D'_{[|1,n|]-I} = \emptyset$. Reste

à montrer que ce sous-graphe est maximal.Procédons par l'absurde : s'il n'était pas maximal on pourrait lui ajouter un ensemble de nœuds sans briser la contrainte. Soit D'' le sous-graphe de D obtenu, on sait que $D' \subseteq D''$ et que $\exists J \subseteq [|1,n|]$, $|J| \geq q$, D''_J est un alignement local de D_J pour la $|J|$-correspondance $\mathcal{R}_{qJ}$ et $D''_{[|1,n|]-J} = \emptyset$. Nous avons deux cas de figure, soit $J = I$, soit $I \subset J$.Si $J = I$, la contrainte d'empilement assure que nous avons ajouté des nœuds bien empilés pour $\mathcal{R}_{qJ}$, qui sont donc bien empilés pour $\mathcal{R}_q$, donc qui apparaissent sous la forme d'empilements du $MGAP$ qui seraient connectés aux empilements de C pour chaque strate, ce qui viendrait contredire la maximalité de C.Si $J \subset I$, on sait que l'on a ajouté des nœuds seulement sur les nouvelles strates de $J - I$. En effet si on en avait ajouté sur les strates de I, le sous-graphe restreint à I serait un alignement local pour R_{qI} (parce que D'' en est un pour R_{qJ}), ce qui vient contredire la maximalité de C (même raisonnement que précédemment). La contrainte d'empilement assure que D'' correspond à un sous-graphe $C' = (\mathcal{U}, \mathcal{F}_1, \ldots, \mathcal{F}_n)$ du $MGAP$, ce sous-graphe vérifie $\exists J \subseteq [|1,n|]$, $|J| \geq q$ tel que $\forall i \in J$, $(\mathcal{U}, \mathcal{F}_i)$ est une composante connexe et $\forall v = (v_1, \ldots v_n) \in \mathcal{U}$, $cover(v) = \{v_i,\, i \in J\}$, or on remarque que $C \sqsubseteq C'$. Les deux connectons étant distincts, cela contredit la maximalité de C pour la relation $\sqsubseteq$.

En pratique nous allons donc calculer les connectons partiels par strate, puis, dans une phase de post-traitement, nous éliminerons les connectons non maximaux au sens de $\sqsubseteq$.

Comme pour le MGA, il est en pratique impossible de construire le $MGAP$ en entier, à la fois à cause de la dégénérescence de la relation de correspondance S et à cause de la présence de nœuds jokers.

Une approche brutale pour calculer les connectons partiels par strate serait d'énumérer pour tous les k de q à n les C_n^k combinaisons de k réseaux parmi n et d'utiliser l'algorithme de partitionnement à la volée (OTF) décrit dans le Chapitre 4. Bien entendu, cette approche deviendrait rapidement impraticable. Nous allons donc adapter OTF à ce nouveau problème.

Cette nouvelle version d'OTF sera appelée OTFS, le S rappelant qu'il s'agit d'alignements partiels par strate.

Informellement, nous allons, à chaque appel de *Prolonger*, prolonger les empilements par des nœuds jokers en plus des prolongements habituels. La procédure *Partitionner* assurera le reste, c'est-à-dire le fait que sur une strate donnée, la classe courante possède soit aucun joker, soit uniquement des jokers.

Rappelons rapidement l'idée générale de OTF : il s'agit d'effectuer un parcours en profondeur d'abord (DFS) sur les classes en commençant par les composantes connexes de la première couleur (le premier réseau primaire), puis d'ajouter des

couleurs de façon incrémentale.

Ainsi le multigraphe n'est pas calculé explicitement, des parties plus petites (classes) sont calculées à la volée dans chaque branche du parcours DFS.

L'algorithme complet a été décrit dans le chapitre précédent, nous allons donc simplement rappeller les deux opérations principales utilisées pendant l'exploration et qui devront être modifiées pour prendre en compte les jokers.

Ces opérations sont :
- $Partitionner_k$ qui partitionne une classe sur les couleurs 1 à k,
- $Prolonger_{k+1}$ qui ajoute la $(k+1)^{ème}$ couleur au multigraphe d'alignement partiel.

5.2.3 Initialisation.

Comme dans OTF, l'algorithme OTFS est initialisé avec le premier réseau. Afin de traiter le cas spécifique où l'on veut ignorer le(s) premier(s) réseau(x) primaire(s), nous allons simplement ajouter une classe initiale supplémentaire formée du singleton $\{*\}$.

5.2.4 Procédure *Partitionner*

Comme dans OTF, l'opération $Partitionner_k$ calcule les composantes connexes sur chacune des couleurs à son tour. Si pour une couleur i la classe est partitionnée, la procédure renvoit les différentes classes obtenues.

La gestion des jokers se fait très simplement : nous n'utilisons en pratique qu'un seul nœud joker physique, déconnecté des autres nœuds. La procédure *Partitionner* va donc naturellement, en partitionnant sur une couleur i, séparer les empilements qui possèdent un joker sur cette couleur des autres empilements.

Cela assure qu'à la fin de l'algorithme OTFS, les classes résultantes pour une couleur donnée contiendront soit aucun joker, soit uniquement des jokers.

Comme dans le Chapitre 4, l'utilisateur peut donc choisir comme *Partitionner* tout algorithme exact de partitionnement.

Enfin, comme dans OTF, lorsqu'une classe C est "stable" pour les couleurs 1 à k, c'est-à-dire lorsqu'elle vérifie $Partitionner_{1-k}(C) = C$, elle est étendue à la $(k+1)^{ème}$ couleur par la procédure PROLONGER.

5.2.5 Procédure *Prolonger*

La procédure PROLONGER doit être adaptée pour s'appliquer correctement à un multigraphe partiel, c'est-à-dire pour tenir compte des jokers. La modification se situe dans la sélection des prolongements (lignes 4 et 5) : les candidats v_{k+1} sont pris dans l'ensemble V^*_{k+1} pour autoriser les prolongements par un joker, et la propriété P_{k+1} va changer pour tenir compte des nouveaux types d'empilements.

Algorithm 5: Prolonger.

Global: Graphe de données stratifié $D = (V, E)$ pour les réseaux $G_i = (V_i, E_i)$, $i \in [|1, n|]$
Input: Multigraphe $C = (\mathcal{V}, \mathcal{E}_1, \ldots, \mathcal{E}_k)$ /* *multigraphe partiel courant à prolonger* */
Output: Multigraphe $NewC = (\mathcal{V}', \mathcal{E}'_1, \ldots, \mathcal{E}'_{k+1})$ /* *multigraphe partiel prolongé sur la strate* $k+1$ */

```
(1)   begin
(2)     /* pour chaque nœud de C sélection des prolongements */
(3)     for (v_1, ..., v_k) ∈ V do
(4)       for v_{k+1} ∈ V*_{k+1} do
(5)         if P_{k+1}(v_1, ..., v_k, v_{k+1}) then
(6)           V' ← V' + {(v_1, ..., v_{k+1})} ;
(7)         end if
(8)       end for
(9)     end for
(10)    /*réintroduire les arêtes en respectant le graphe de données
        stratifié*/
(11)    (E'_1, ..., E'_{k+1}) ← ReconstruireAretes(V', D) ;
(12)    return (V', E'_1, ..., E'_{k+1}) ;
(13)    end
```

Voyons comment les propriétés de prolongement sont changées pour tenir compte du quorum. Pour chaque type d'empilement partiels, nous allons avoir deux cas : le cas d'un prolongement en joker et le cas d'un prolongement par un nœud de la $k+1$-ème strate. Supposons, dans un premier temps, que l'empilement à prolonger $v = (v_1, \ldots v_k)$ soit tel que $cover(v) \neq \emptyset$. Nous donnons dans la Figure 5.5, un exemple de prolongements d'un tel empilement.

Dans le cas des empilements-cliques partiels, pour un prolongement en joker nous nous contenterons de vérifier que l'ajout de ce joker ne vient pas contredire la condition de quorum. Pour un prolongement par un nœud de la $k+1$-ème strate,

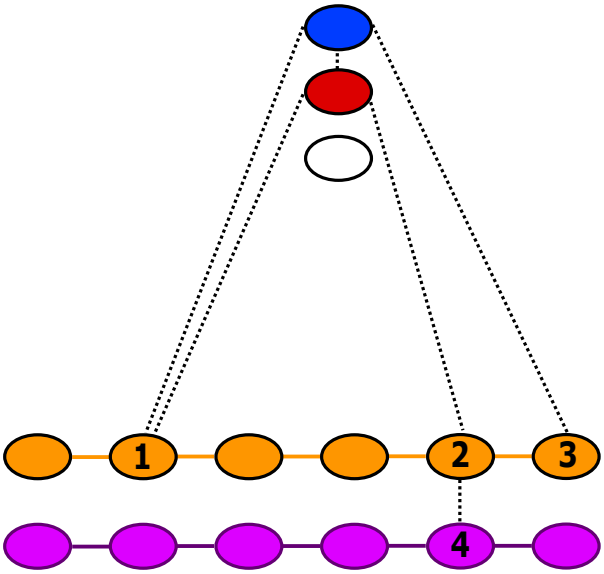


FIGURE 5.5 – Exemple de sélection de prolongements d'un empilement partiel non-vide pour un quorum $q = 3$ sur 5 réseaux. Nous avons un triplet déjà construit, formé d'un nœud du réseau bleu, un nœud du réseau rouge et un nœud joker. Nous cherchons à le prolonger sur la strate orange. Dans tous les cas nous pourrons le prolonger par un nœud joker, puisqu'il reste suffisamment de strates après pour ne pas briser la condition de quorum. a) Pour les empilements-cliques partiels, seul le nœud 1 de la strate orange est pertinent, parce qu'il est connecté à chacun des nœuds non-jokers du triplet. b) Pour les empilements-étoile, nous obtenons les deux nœuds 1 et 3, parce qu'ils sont tous deux reliés par une arête pointillée au nœud bleu du triplet. c) Avec des empilements-connexes, nous devons récupérer uniquement le nœud 2 parce que c'est le seul prolongement gagnant.

nous allons par contre imposer qu'il soit en correspondance avec chacun des nœuds de $cover(v)$. En pratique la propriété s'exprime de la façon suivante :

Propriété de Prolongement 5. *(Empilements-Cliques Partiels)*

- $Pclique_{k+1}(v_1, \ldots, v_k, *) = (|cover(v_1, \ldots, v_k, *)| \geq q - (n - k - 1))$,
- $Pclique_{k+1}(v_1, \ldots, v_k, v_{k+1} \neq *) = \forall v \in cover(v_1, \ldots, v_k),\ (v, v_{k+1}) \in E$.

L'idée derrière la condition sur $|cover(v_1, \ldots, v_k, *)|$ est qu'il reste $n - k - 1$ strates, donc le $k + 1$-uplet peut encore être complété par $n - k - 1$ nœuds non-jokers, la condition à la fin est donc $n - k - 1 + |cover(v_1, \ldots, v_k, *)| \geq q$. Le second cas correspond simplement au test de clique.

Dans le cas des empilements-étoiles partiels, pour un prolongement en joker la condition nécessaire sera, une fois encore, que ce joker ne vienne pas briser la condition de quorum. Pour un prolongement par un nœud de la $k + 1$-ème strate, nous imposerons qu'il soit en correspondance avec le nœud v_1 du réseau central.

Propriété de Prolongement 6. *(Empilements-Etoiles Partiels)*

- $Petoile_{k+1}(v_1, \ldots, v_k, *) = |cover(v_1, \ldots, v_k, *)| \geq q - (n - k - 1))$,
- $Petoile_{k+1}(v_1, \ldots, v_k, v_{k+1}) = (v_1, v_{k+1}) \in E$.

Enfin, pour un empilement connexe partiel, nous allons chercher, comme précédemment, les prolongements gagnants, nous voulons donc que pour chaque prolongement, y compris le prolongement en joker, on puisse exhiber des nœuds qui complètent le $k+1$-empilement partiels en un n-empilement partiel v qui convienne, c'est à dire tel que $cover(v)$ est connexe et $|cover(v)| \geq q$.

Propriété de Prolongement 7. *(Empilements-Connexes Partiels)*
$Pconnexe_{k+1}(v_1, \ldots, v_k, v_{k+1}) = \exists \{v_{k+2}, \ldots v_n\} \in V^*_{k+2} \times \ldots V^*_n$ *tels que*

- $cover(v_1, \ldots v_n)$ *est connexe,*
- $|cover(v_1, \ldots v_n)| \geq q$.

L'implémentation des prolongements des empilements-cliques partiels et des empilements-étoiles partiels ne pose pas de problèmes particuliers, il s'agit simplement de parcourir $cover(v)$ et de tester sur les correspondances des nœuds avec V_{k+1}. Pour les prolongements des empilements-connexes partiels, le fonctionnement est le même que pour les empilements-connexes : on commence par calculer l'intersection de la $k + 1$-ème strate avec la composante connexe pour E_S qui contient $cover(v_1, \ldots v_k)$, puis on parcourt les voisins de ces nœuds pour E_S en cherchant un n-empilement partiel passant une ou zéro fois par chaque strate, contenant $cover(v_1, \ldots v_k)$ et vérifiant la condition de quorum.

Une fois les empilements prolongés par des éléments de V^*_{k+1}, il ne reste plus qu'à ajouter les arêtes, comme décrit dans la section précédente.

Notons qu'en pratique l'algorithme fonctionne de façon plus efficace, en séparant dès la procédure *Prolonger* les prolongements en joker - qui seront dans une classe à part - des autres prolongements. Cela nous permet d'éviter la construction d'un grand nombre d'arêtes entre empilements qui de toute façon seraient séparés par la procédure *Partitionner*.

Revenons au cas particulier écarté précédemment : si $cover(v) = \emptyset$, nous savons, par construction, que v est le seul empilement du multigraphe courant, puisque la procédure *Partitionner* est conçue de telle sorte que dans chacun de ses résultats on n'a jamais sur une strate à la fois des nœuds jokers et des nœuds non-jokers.

Nous n'avons donc pas d'autre choix que de sélectionner comme prolongements de v tous les nœuds de V_{k+1}, ainsi que $*$ si cela ne contredit pas la condition de quorum.

Propriété de Prolongement 8. *(tout type d'empilement partiel)*

- $P_{k+1}(*, *, \ldots, *, v_{k+1} = *) = (n - k - 1) \geq q$,
- $P_{k+1}(*, *, \ldots, *v_{k+1} \neq *) = vrai$.

Intuitivement, il est normal que l'on soit obligés de prolonger par tout le $k+1$-ème réseau puisque le cas de figure que nous sommes en train de traiter revient en pratique à ignorer complètement les k premiers réseaux.

Notes d'implémentation.
Le paramètre associé au quorum est appelé $maxstar$, il correspond en pratique au nombre maximal de jokers que l'on peut utiliser. Autrement dit, $maxstar = n - q$.

Nous donnons, sur la Figure 5.6 un exemple simple d'exécution d'OTFS pour $maxstar = 1$.

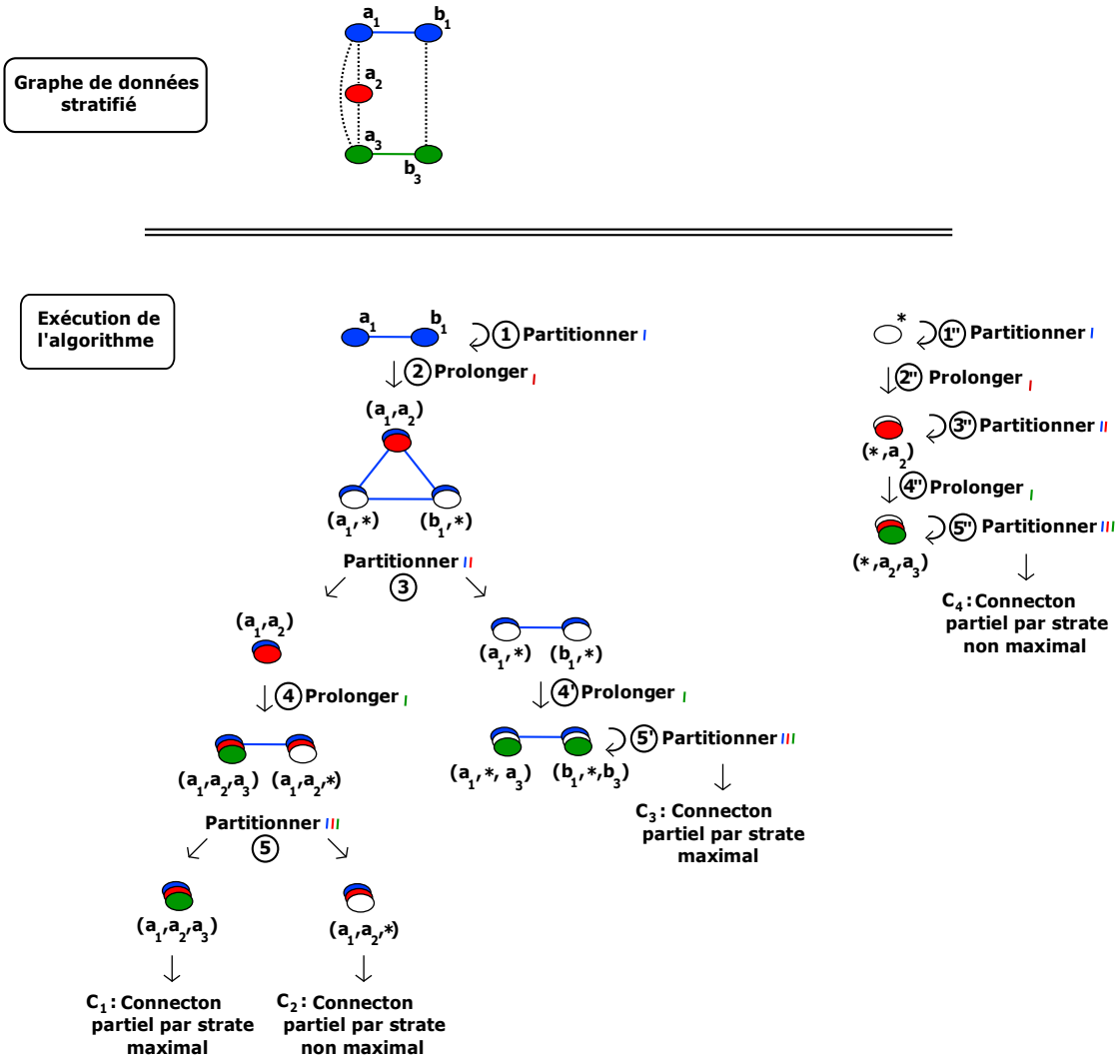


FIGURE 5.6 – Exemple d'exécution d'OTFS pour $maxstar = 1$. En haut nous donnons le graphe de données stratifié. Les ronds blancs représentent des nœuds jokers. Par rapport à OTF, OTFS commence avec deux classes : à gauche nous suivons l'exécution sur la première classe, constituée du premier réseau (bleu), à droite nous suivons l'exécution sur la seconde classe, formée d'un unique nœud joker. Tant que cela ne brise pas la condition de quorum, l'opération d'ajout de strate propose maintenant un prolongement par un nœud joker en plus des prolongements "classiques". On remarque aussi que les étapes de partitionnement séparent systématiquement les empilements qui n'ont pas des nœuds jokers sur les mêmes strates (c'est le cas dans les étapes 3 et 5). Une fois les connectons partiels par strate calculés, l'algorithme élimine ceux qui ne sont pas maximaux, les connectons maximaux sont donc C_1 et C_3 (car $C_2 \sqsubseteq C_1$ et $C_4 \sqsubseteq C_1$). Si on lance OTF sur le même exemple, on ne retrouve que C_1.

5.2.6 Amélioration : pré-traitement

Le principe est identique au pré-traitement décrit dans la Section 4.3.1 : nous allons supprimer les nœuds dont on sait avant le lancement d'OTF qu'ils ne pourront pas être dans des empilements partiels pour le quorum q.

Le processus est similaire, il suffit de supprimer maintenant les nœuds appartenant à des composantes connexes de S dans lesquelles moins de q strates sont représentées :

- calculer les composantes connexes de la relation de correspondance (S) du graphe de données stratifié,
- pour chaque composante connexe retrouver le nombre de strates touchées
- supprimer les nœuds des composantes connexes touchant moins de q strates, et les arêtes associées.

5.3 Alignement partiel par empilement

Dans cette section, nous commençons par donner la seconde définition des alignements partiels, le quorum est ici uniquement imposé "localement" dans chacun des empilements.
Informellement, il s'agit donc d'une définition moins contrainte que la définition des alignements partiels par strate. En effet, nous autorisons, dans chaque empilement, jusqu'à $n - q$ nœuds manquants sans imposer que ces nœuds manquants soient dans les mêmes réseaux, et sans imposer non plus que les empilements contiennent le même nombre de nœuds manquants.
Cela implique notamment que tous les alignements partiels par strate sont des alignements partiels par empilement.

Ensuite, nous fournissons la nouvelle définition des connectons partiels par empilement dans le $MGAP$, et enfin nous détaillons les modifications algorithmiques.

5.3.1 Définition des alignements partiels par empilement

Cette fois, il ne s'agit plus d'ignorer globalement un certain nombre de réseaux. Les nœuds manquants peuvent être sur des strates différentes, la seule contrainte est qu'il n'y en ait pas plus de $n - q$ par empilement.

Afin de définir les alignements partiels par empilement, il nous faut donc revenir à la contrainte d'empilement définie en section 3.

Rappelons rapidement que cette contrainte assure que dans un alignement local

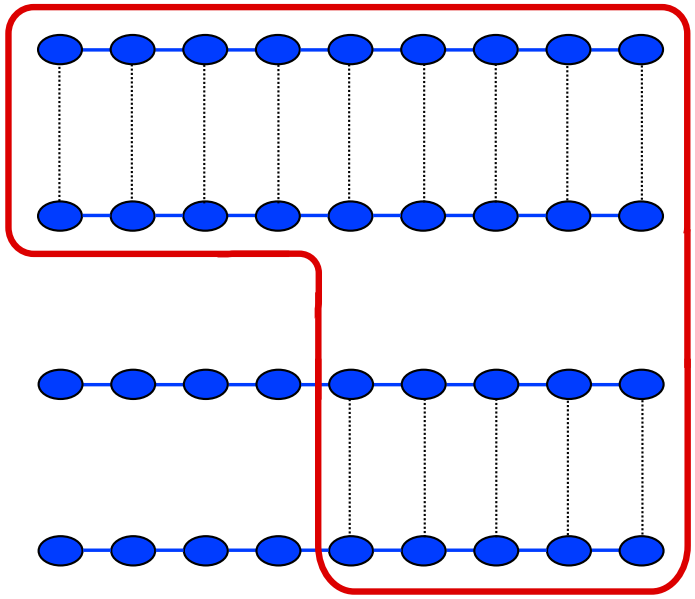


FIGURE 5.7 – Exemple d'alignement partiel par empilement pour un quorum de 2 sur 4 réseaux. L'alignement entouré en rouge convient, or il n'y a aucune correspondance entre la partie de l'alignement située sur les deux réseaux du haut et celle située sur les deux réseaux du bas.

D' chaque nœud est bien empilé - c'est-à-dire appartient bien à un empilement inclus dans D'.

Informellement on se rend compte que cette définition peut facilement être transposée au cas du quorum par empilement : il suffit de remplacer la notion d'empilement par la notion d'empilement partiel.

On remarque néanmoins qu'une nouvelle difficulté apparaît : pour un quorum suffisamment petit, on peut imaginer un sous-graphe D' du graphe de données stratifié qui vérifie ces contraintes, mais se trouve séparé en plusieurs parties sans correspondance S entre elles. Nous en donnons un exemple sur la Figure 5.7. Ce type de situation n'est clairement pas acceptable d'un point de vue biologique.

Nous allons donc imposer, dans le cas du quorum par empilement, que $q > \frac{n}{2}$ ce qui règle en pratique le problème et reste raisonnable au vu de notre objectif.

Pour un quorum $q > \frac{n}{2}$, on commence donc par choisir une n-correspondance partielle de la même façon que dans la section 5.2, en choisissant le type d'empilements que l'on veut imposer (empilement-clique partiel, empilement-chemin partiel, *etc*).

On redéfinit ensuite ce que sont les nœuds bien empilés dans un sous-graphe du graphe de données stratifié.

Définition 5.12. *Etant donnés n réseaux primaires $G_i = (V_i, E_i)$, $i \in [|1, n|]$, le graphe de données stratifié associé $D = (V, E)$ et la n-correspondance avec un quorum q choisi $R_q(V_1^* \times V_2^* \times \ldots \times V_n^*)$, on dira qu'un nœud $v_i \in V_i$ est* bien empilé *dans un sous-graphe $D' = (V', E')$ de D si et seulement si $\exists$ un ensemble $\{v_j\}_{j \neq i} \subset V'^*$ telle que $(v_1, \ldots, v_n) \in R_q(V_1^* \times V_2^* \times \ldots \times V_n^*)$.*

Ce qui nous permet de définir la contrainte d'empilement partiel.

Définition 5.13. *Etant donnés n réseaux primaires $G_i = (V_i, E_i)$, $i \in [|1, n|]$, le graphe de données stratifié associé $D = (V, E)$ et la n-correspondance choisie $R(V_1 \times V_2 \times \ldots \times V_n)$, on dira qu'un sous-graphe $D' = (V', E')$ de D satisfait la contrainte d'empilement partiel si et seulement si chacun des nœuds de V' est bien empilé dans D' (au sens de la définition précédente).*

Nous pouvons donc définir formellement les alignements maximaux pour un quorum par empilement $q > \frac{n}{2}$.

Définition 5.14. *Un sous-graphe D' d'un graphe de données stratifié D est un alignement maximal pour un quorum par empilement local $q > \frac{n}{2}$ s'il vérifie la contrainte d'empilement partiel et la contrainte de localité*[4]*, et si l'on ne peut pas lui ajouter de nœud sans briser une de ces deux contraintes.*

Sur la figure 5.8, nous donnons un exemple d'alignement partiel par empilement dans le cas de trois réseaux représentant des génomes.

La définition du $MGAP$ est la même que celle donnée en section 5.2, il ne nous reste plus qu'à définir les connectons partiels par empilement dans ce $MGAP$ qui correspondent à notre nouvelle définition d'alignements locaux.

5.3.2 Définition des connectons partiels par empilement

Etant donné $\mathcal{V}'$ un ensemble de nœuds du $MGAP$, on note $\mathcal{V}'_{*i}$ l'ensemble des empilements de $\mathcal{V}'$ qui possèdent un joker en position i (définition 5.8).

Définition 5.15. *Un* connecton partiel *est un sous-graphe maximal $(\mathcal{V}', \mathcal{E}'_1, \ldots, \mathcal{E}'_n)$ du multigraphe d'alignement partiel tel que $\forall i$, $(\mathcal{V}' - \mathcal{V}'_{*i}, \mathcal{E}'_i)$ est une composante connexe et $\forall v \in \mathcal{V}'$, $cover(v) \geq q$.*

4. Au sens de la définition 3.12 de la section 3.

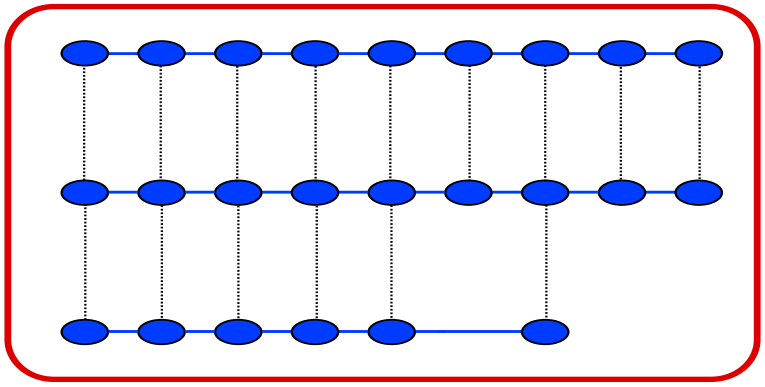


FIGURE 5.8 – Alignement local partiel par empilement dans un graphe de données stratifié correspondant à trois génomes. On constate que l'ensemble forme un alignement local

Informellement, $(\mathcal{V}' - \mathcal{V}'_{*i}, \mathcal{E}'_i)$ est la "projection" du sous-graphe sur la $i^{ème}$ strate : nous nous contentons de dire que sur chaque couleur le sous-graphe doit être connexe si l'on ignore les nœuds jokers.

La maximalité est, encore une fois, du point de vue des nœuds, il n'est pas possible d'ajouter un nœud à un connecton sans rompre la condition de connexité.

Contrairement à la définition de la section précédente (Section 5.2), cette définition autorise la présence sur une même couleur dans un connecton partiel de nœuds jokers et non-jokers.

Les résultats correspondent bien aux alignements maximaux partiels par empilement. Notons $D' = (V', E')$ le sous-graphe du graphe de données stratifié induit par $(\mathcal{V}', \mathcal{E}'_1, \ldots, \mathcal{E}'_n)$. $(\mathcal{V}', \mathcal{E}'_1, \ldots, \mathcal{E}'_n)$ étant un sous-graphe du $MGAP$, D' vérifie naturellement la contrainte d'empilement partiel, la connexité de $(cover_i(\mathcal{V}'), \mathcal{E}'_i)$ implique celle de $(V' \cap V_i, E' \cap E_i)$, ce qui implique que la contrainte de localité est elle aussi vérifiée. Enfin la maximalité des connectons partiels par empilement implique celle des alignements locaux maximaux. En effet, si l'alignement local n'était pas maximal on pourrait lui ajouter un certain nombre de nœuds sans rompre les contraintes, la contrainte d'empilement impliquerait que l'ensemble des empilements correspondants appartiendraient au $MGAP$, et la contrainte de localité impliquerait qu'en étendant le connecton par ces empilements on conserverait un connecton partiel. Ce qui est absurde puisqu'un connecton partiel est maximal.

En revanche, on remarque que les connectons partiels par empilement ne forment pas une partition des nœuds du $MGAP$.

En d'autres termes, un même empilement du $MGAP$ peut appartenir à plu-

sieurs connectons partiels par empilement. Plus précisément, si nous disons d'un nœud v tel que $|cover(v)| = n$ (resp. $q \leq |cover(v)| < n$) est "complet" (resp. "incomplet"), alors les connectons partiels par empilement forment une partition des nœuds complets, mais pas des nœuds incomplets. Une conséquence directe de cet état de fait est que les algorithmes classiques de partitionnement de graphe ne sont pas directement utilisables.

Une approche possible serait de commencer avec une partition des nœuds complets (considérés comme des "graines"), puis d'étendre les connectons avec des nœuds incomplets. Une telle heuristique ne permettrait notamment pas de prendre en compte des connectons formés uniquement de nœuds incomplets.

Comme dans le cas du quorum par strate, nous faisons donc le choix d'étendre l'algorithme de partitionnement de graphe décrit dans la section OTF. Les modifications vont intervenir dans les deux fonctions au cœur de l'exploration, l'opération de partitionnement et l'opération d'ajout de strate. L'algorithme résultant est appelé OTFE.

Dans l'approche avec un quorum par strate, à chaque *Prolonger*, deux prolongements de la classe courante sont construits - tant que c'est possible d'après le quorum. Le premier est le prolongement classique opéré dans OTF, le second est un prolongement de tous les empilements par des nœuds jokers.

Avec la nouvelle définition des connectons partiels par empilement nous voyons bien que cette méthode ne fonctionne plus : nous voulons autoriser pour une même couleur la cohabitation entre nœuds jokers et nœuds des réseaux primaires.

5.3.3 Initialisation.

L'initialisation est une fois encore légèrement modifiée, l'algorithme OTFE va commencer avec une classe formée du singleton $\{*\}$ et une seconde classe correspondant au premier réseau mais cette seconde classe contiendra elle aussi - en plus du premier réseau - un empilement formé d'un nœud joker.

C'est cet empilement supplémentaire qui, une fois prolongé, va nous permettre de rendre compte des erreurs ou nœuds manquants possibles sur les premières strates.

5.3.4 Procédure *Partitionner*

Comme dans OTF, l'opération *Partitionner* dans OTFE calcule les composantes connexes sur chacune des couleurs à son tour.

La différence principale est que nous ne voulons plus, pour une couleur i, séparer les empilements qui possèdent un joker sur cette couleur des autres empilements : les empilements avec joker seront considérés en pratique comme appartenant à chacune des composantes connexes.

Sur une couleur donnée i, ceci est simplement effectué de la façon suivante :

1. retirer temporairement les empilements qui possèdent un nœud joker à la position i,
2. Partitionner l'ensemble restant sur i,
3. remettre les empilements dans chacune des classes résultantes (s'il y en a).

Une fois que la classe C vérifie $Partitionner_{1-k}(C) = C$, elle doit être étendue à la $(k+1)^{ème}$ couleur.

5.3.5 Procédure *Prolonger*

Le pseudocode de la procédure reste le même que dans la section 5.2.

Tant que $cover(v) \neq \emptyset$, les propriétés de prolongements sont conservées puisque la définition des empilements partiels reste la même.

Nous donnons sur la Figure 5.9 un exemple de déroulement de l'algorithme.

Détaillons maintenant le cas $cover(v) = \emptyset$. En pratique, c'est un cas qui va se poser assez souvent, puisque jusqu'à la $(n-q+1)$-ème strate, il y a un empilement de ce type dans chaque classe.

Ce sont ces empilements qui nous permettent d'autoriser que la classe contienne au final des empilements avec des nœuds manquants sur les k premières strates.

Si dans le multigraphe courant cet empilement est seul, nous allons utiliser la propriété de prolongement décrite dans la section 5.2. Si par contre il y a d'autres empilements dans le multigraphe courant, nous allons pouvoir limiter la quantité de prolongements en utilisant le fait qu'il y'aura toujours au moins un empilement non-vide du multigraphe courant qui se retrouvera dans le connecton partiel final. Nous donnons un exemple pratique de prolongement dans la Figure 5.10.

Nous avons défini une première propriété de prolongement des empilements vides, qui dépend de l'ensemble $\mathcal{V}$ des empilements du multigraphe courant.

Cette première propriété exploite simplement le fait que tout alignement local

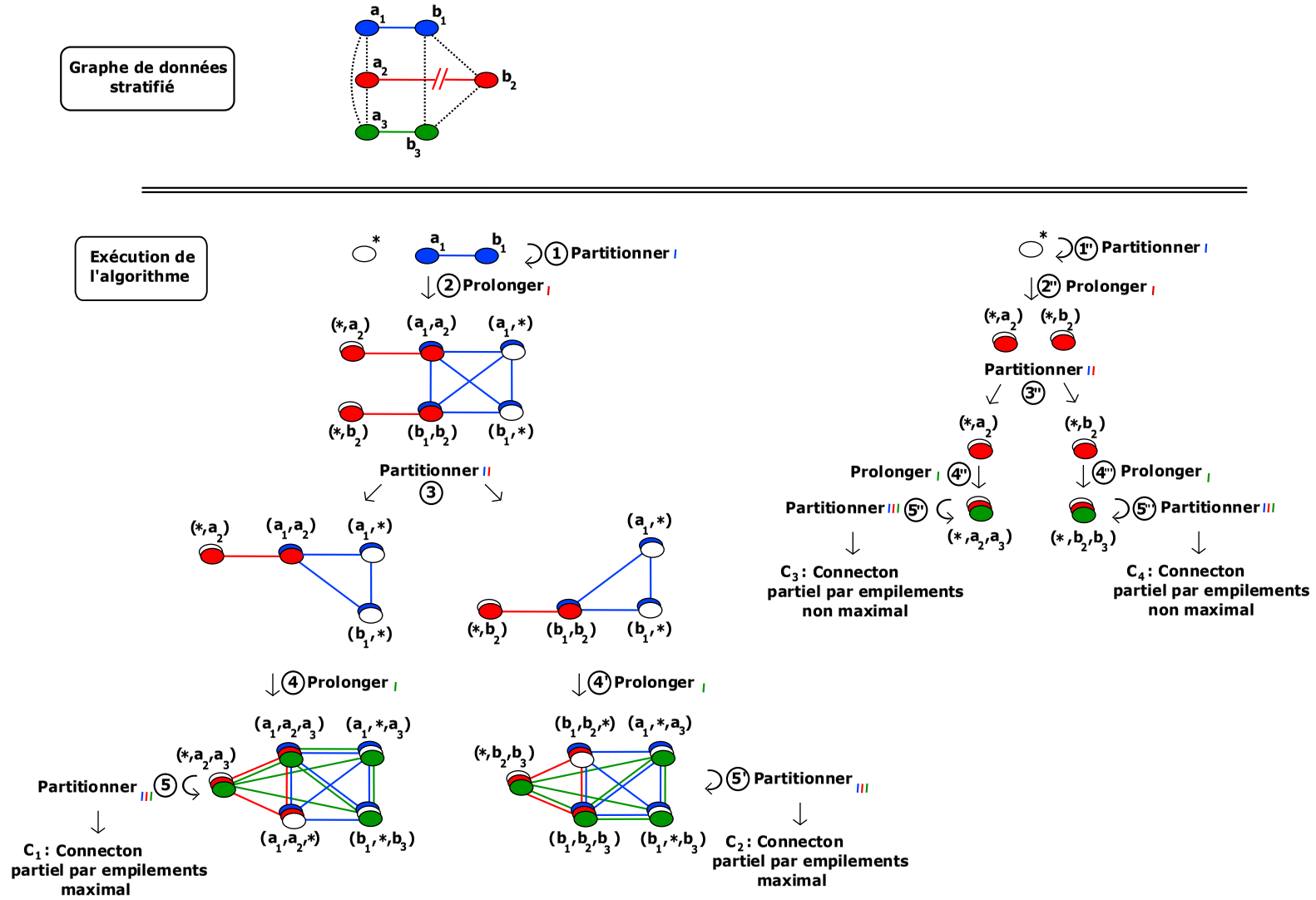


FIGURE 5.9 – Exemple d'exécution d'OTFE pour $maxstar = 1$. En haut nous donnons le graphe de données stratifié (a_2 et b_2 sont déconnectés). L'initialisation change encore par rapport à OTFS : OTFE commence toujours avec deux classes, mais la première classe (à gauche) est maintenant formée du premier réseau (bleu) auquel on ajoute un nœud joker, la seconde classe est toujours formée d'un unique nœud joker. Par rapport à OTFS, l'opération de partitionnement ne sépare plus les empilements qui ont des jokers sur des strates différentes. Elle consiste à 1) retirer provisoirement les empilements qui possèdent un joker sur la couleur courante, 2) partitionner sur cette couleur les empilements restants, 3) réinjecter les empilements retirés dans les CC résultantes (s'il y en a). L'opération d'ajout de strate est aussi modifiée pour gérer de façon efficace le prolongement des empilements vides. Les alignements locaux partiels par empilements obtenus sont formés des nœuds $\{a_1, a_2, a_3, b_1, b_3\}$ (C_1) et $\{a_1, a_3, b_1, b_2, b_3\}$ (C_2). Sur le même exemple, la définition "par strate" respectée par OTFS aurait donné l'alignement $\{a_1, a_3, b_1, b_3\}$.

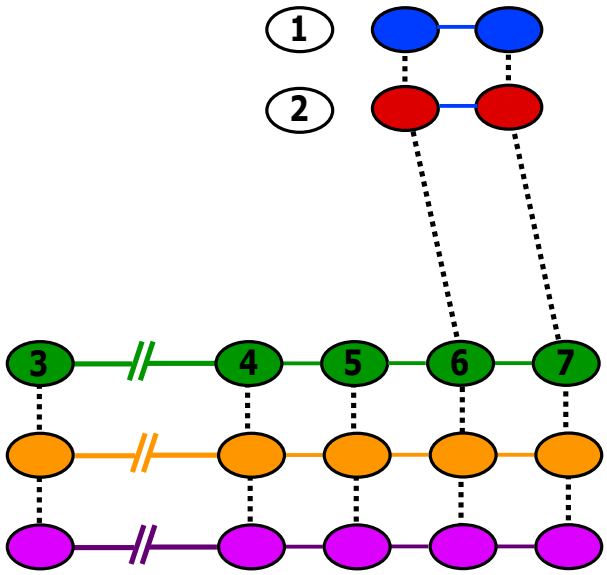


FIGURE 5.10 – Pour $maxstar = 2$, exemple de sélection de prolongements pour un empilement vide $(1, 2)$ dans un multigraphe courant contenant aussi deux empilements complets. Peu importe la propriété choisie (parmi 9, 10 et 11), nous allons prolonger $(1, 2)$ par 4, 5, 6 et 7, mais pas par 3 puisqu'il n'est pas sur la même composante connexe que les prolongements des empilements complets.

partiel par empilement doit être connexe. Dans le graphe de données stratifié $D = (\bigcup_i V_i,\ E_P \cup E_S)$, soit $CC(\mathcal{V})$ les composantes connexes englobant le sous-graphe induit par $\mathcal{V}$ dans le graphe de données stratifié (il peut y en avoir plusieurs, nous savons simplement qu'à la fin de l'algorithme, le sous-graphe sera connexe).

Propriété de Prolongement 9. *(prolongement des empilements vides)*
$P^{\mathcal{V}}_{k+1}(*, *, \ldots, *, v_{k+1}) = v_{k+1} \in (V_{k+1} \cap CC(\mathcal{V}))$

On peut utiliser une propriété de prolongement des empilements vides plus contrainte exploitant les contraintes d'empilement et de localité, afin de limiter un peu plus l'augmentation de la taille du multigraphe.

Notons $chemin_i$ un chemin sur la i-ème strate du graphe de données stratifié (c'est-à-dire une suite de nœuds reliés par des arêtes prises dans E_i).

On sait que sur toutes les strates, on doit retrouver des composantes connexes. Au lieu de parcourir pour chaque nœud non-joker des empilements de $\mathcal{V}$ toute sa composante connexe, nous allons donc nous limiter à des chemins de la forme $chemin_i \rightarrow e \in E_S \rightarrow chemin_{j \neq i}$ *etc* $chemin_{k+1}$.

Soit $onewaypath(\mathcal{V})$ l'ensemble des nœuds obtenus, notre propriété de prolongement est la suivante.

Propriété de Prolongement 10. *(prolongement des empilements vides)*
$P^{\mathcal{V}}_{k+1}(*, *, \ldots, *, v_{k+1}) = v_{k+1} \in onewaypath(\mathcal{V})$

Nous pouvons faire encore mieux en exploitant le fait que $q > \frac{n}{2}$. Notons $chemin_S$ un chemin utilisant les arêtes de l'ensemble E_S.

Cette condition sur le quorum implique en effet que, dans un connecton partiel, pour tout couple d'empilements on pourra toujours trouver une couleur sur laquelle ni l'un ni l'autre ne possède de joker. La conséquence est que nous pouvons limiter notre parcours à des chemins de la forme $chemin_S \rightarrow chemin_i \rightarrow chemin_S$.

Soit $twowaypath(\mathcal{V})$ l'ensemble des nœuds obtenus, notre propriété de prolongement est la suivante.

Propriété de Prolongement 11. *(prolongement des empilements vides)*
$P^{\mathcal{V}}_{k+1}(*, *, \ldots, *, v_{k+1}) = v_{k+1} \in twowaypath(\mathcal{V})$

Avec la propriété de prolongement des empilements "vides" choisie, nous pouvons donc construire les empilements, puis les relier de la même façon que dans la section 5.2.

Notes d'implémentation.
L'utilisateur peut fixer un nombre minimal d'empilements complets dans chaque connecton partiel. Le paramètre correspondant est appelé **mincore**.
En pratique, fixer **cleanup** ≥ 1 accélère notablement OTFE.

5.4 Application de l'alignement partiel par strate : alignement multiple de génomes bactériens, comparaison à i-ADHoRe

5.4.1 Objectif

L'objectif de ce travail est de comparer les synténies trouvées par l'approche d'alignement partiel par strate avec celles trouvées par le programme i-ADHoRe dont nous avons déjà parlé au Chapitre 2 et qui constitue un des programmes les plus populaires pour cet usage. La comparaison porte à la fois sur l'efficacité algorithmique mais aussi sur la nature des syntons trouvés dans les deux approches. Rappelons que i-ADHoRe est une heuristique et qu'il n'autorise pas de permutation de l'ordre des gènes dans les blocs de synténies. La comparaison des résultats permet entre autres :

1. de tester l'efficacité de l'heuristique (rate-t-elle beaucoup de cas ?),
2. d'estimer l'importance de la prise en compte des permutations (sont-elles observées en pratique ?).

5.4.2 Données

Afin de tester le comportement des deux approches dans des situations suffisamment diverses, nous avons constitué cinq groupes composés chacun de 10 bactéries situées à des distances phylogénétiques variables (Table 5.1). Le groupe le plus hétérogène est le groupe **bacteria**, composé de 10 espèces bactériennes choisies dans des groupes phylogénétiques éloignés. Le plus homogène est le groupe **entero** composé des 10 *Enterobacteriaceae*. La composition précise de chacun des groupes est donné dans la Table 5.1.

bacteria		**Bacteria**		
acnum	*abbrev.*	*Espèce*	*Mb*	*Ngènes*
AE000512_GR	thmar	Thermotoga maritima (strain JCM 10099 / DSM 3109)	1.9	1853
AE000520_GR	trpal	Treponema pallidum (strain Nichols)	1.1	1028
AE001273_GR	chtra	Chlamydia trachomatis (strain D/UW-3/Cx)	1.0	895
AE009951_GR	funuc	Fusobacterium nucleatum nucleatum (strain JCM 8532)	2.2	2069
AL009126_GR	basub	Bacillus subtilis (strain 168)	4.2	4237
AM398681_GR	flpsy	Flavobacterium psychrophilum (strain JIP02/86)	2.8	2432
BA000022_GR	syspp	Synechocystis sp. (strain PCC 6803)	3.6	3166
BX248353_GR	codip	Corynebacterium diphtheriae (strain NCTC 13129)	2.5	2317
CP000359_GR	degeo	Deinococcus geothermalis (strain DSM 11300)	2.5	2330
U00096_GR	escol	Escherichia coli (strain K12)	4.6	4320
firmi		**Bacteria ; Firmicutes**		
acnum	*abbrev.*	*Espèce*	*Mb*	*Ngènes*
AE016830_GR	enfae	Enterococcus faecalis (strain V583 / ATCC 700802)	3.2	3113
AL009126_GR	basub	Bacillus subtilis (strain 168)	4.2	4237
AL591824_GR	limon	Listeria monocytogenes (serovar 1/2a, strain EGD-e)	2.9	2852
AP008230_GR	dehaf	Desulfitobacterium hafniense (strain Y51)	5.7	5058
CP000414_GR	lemes	Leuconostoc mesenteroides (subsp. mesenteroides)	2.0	1970
CP000703_GR	staur	Staphylococcus aureus (strain JH9)	2.9	2701
CP000924_GR	thpse	Thermoanaerobacter pseudethanolicus (ATCC 33223)	2.4	2243
CP000939_GR	clbot	Clostridium botulinum (strain Okra / Type B1)	4.0	3652
CP001033_GR	stpne	Streptococcus pneumoniae (strain CGSP14)	2.2	2206
FM177140_GR	lacas	Lactobacillus casei (strain BL23)	3.0	3042
proteo		**Bacteria ; Proteobacteria**		
acnum	*abbrev.*	*Espèce*	*Mb*	*Ngènes*
AE001439_GR	hepyl	Helicobacter pylori (strain J99)	1.6	1491
AE005673_GR	cacre	Caulobacter crescentus (strain CB15 / ATCC 19089)	4.0	3738
AE016825_GR	chvio	Chromobacterium violaceum (strain IFO 12614)	4.7	4407
AE017282_GR	mecap	Methylococcus capsulatus (strain Bath / NCIMB 11132)	3.3	2960
CP000112_GR	dedes	Desulfovibrio desulfuricans (strain G20)	3.7	3775
CP000251_GR	andeh	Anaeromyxobacter dehalogenans (strain 2CP-C)	5.0	4346
CP000661_GR	rhsph	Rhodobacter sphaeroides (strain ATCC 17025)	3.2	3111
CP000744_GR	psaer	Pseudomonas aeruginosa (strain PA7)	6.6	6286
CP000814_GR	cajej	Campylobacter jejuni (subsp. jejuni, serovar O :6)	1.6	1626
U00096_GR	escol	Escherichia coli (strain K12)	4.6	4320
gamma		**Bacteria ; Proteobacteria ; Gammaproteobacteria**		
acnum	*abbrev.*	*Espèce*	*Mb*	*Ngènes*
AE017282_GR	mecap	Methylococcus capsulatus (strain Bath / NCIMB 11132)	3.3	2960
AM920689_GR	xacam	Xanthomonas campestris (pathovar campestris)	5.1	4510
CP000127_GR	nioce	Nitrosococcus oceani (strain ATCC 19707 / NCIMB 11848)	3.5	2976
CP000462_GR	aehyd	Aeromonas hydrophila (subsp. hydrophila, ATCC 7966)	4.7	4122
CP000675_GR	lepne	Legionella pneumophila (strain Corby)	3.6	3204
CP000681_GR	shput	Shewanella putrefaciens (strain CN-32 / ATCC BAA-453)	4.7	3972
CP000744_GR	psaer	Pseudomonas aeruginosa (strain PA7)	6.6	6286
CP001091_GR	acple	Actinobacillus pleuropneumoniae (serovar 7, AP6 / AP76)	2.3	2131
CP001132_GR	acfer	Acidithiobacillus ferrooxidans (strain ATCC 53993)	2.9	2826
U00096_GR	escol	Escherichia coli (strain K12)	4.6	4320
enterob		**Bacteria ; Proteobacteria ; Gammaproteobacteria ; Enterobacteriales ; Enterobacteriaceae**		
acnum	*abbrev.*	*Espèce*	*Mb*	*Ngènes*
AE005674_GR	shfle	Shigella flexneri (serovar 2a, strain 301)	4.6	4395
AE006468_GR	satyp	Salmonella typhimurium (strain ATCC 700720)	4.9	4455
AE009952_GR	yepes	Yersinia pestis (biovar Mediaevalis, strain KIM5)	4.6	4104
AP008232_GR	soglo	Sodalis glossinidius (strain morsitans)	4.2	2432
BX470251_GR	phlum	Photorhabdus luminescens laumondii (strain TT01)	5.7	4897
BX950851_GR	ercar	Erwinia carotovora (subsp. atroseptica, ATCC BAA-672)	5.1	4491
CP000653_GR	enspp	Enterobacter sp. (strain 638)	4.5	4115
CP000822_GR	cikos	Citrobacter koseri (strain ATCC BAA-895)	4.7	5003
CP000964_GR	klpne	Klebsiella pneumoniae (strain 342)	5.6	5425
U00096_GR	escol	Escherichia coli (strain K12)	4.6	4320

TABLE 5.1 – Détail des cinq groupes d'espèces bactériennes sélectionnés. *acnum* est le numéro d'accession *Genome Reviews* ; *abbrev.* correspond à l'abbréviation utilisée pour la bactérie ; la colonne *Espèce* donne l'espèce bactérienne (et la souche) ; dans la colonne *Mb* nous donnons la taille du génome en Mb ; *Ngènes* est le nombre de gènes présents dans le génome.

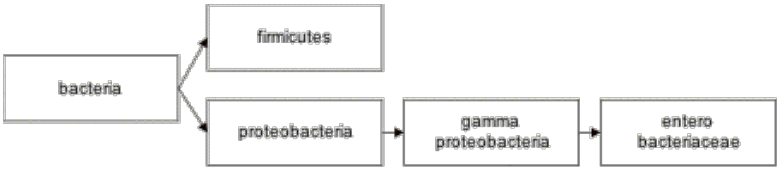


FIGURE 5.11 – Relation entre les cinq groupes d'espèces bactériennes sélectionnés.

5.4.3 Homogénéité des groupes

Afin de mesurer l'homogénéité d'un groupe, pour chacune des $(10 \times 9)/2 = 45$ paires de génomes (G_1, G_2) du groupe on calcule les similarités gène à gène à l'aide de BLAST. Deux gènes g_1 et g_2 sont dit similaires si le "hit" BLAST de g_1 sur g_2 vérifie les conditions de filtrages suivantes :

- $P - value \leq 10^{-10}$,
- $\%identité \geq 40\%$,
- $couvertureMin \geq 80\%$ avec $couvertureMin = \frac{Longueur(hit)}{min(Longueur(g_1), Longueur(g_2))}$.

Ceci permet ensuite de définir un indice de similarité $\mathcal{S}$ entre les génomes G_1 et G_2 : $\mathcal{S}(G1, G2) = min(F(G1, G2), F(G2, G1))$
où $F(A, B)$ est le nombre de gènes dans A ayant au moins un gène similaire dans B sur le nombre de gènes dans A, c'est-à-dire la fraction des gènes de A ayant au moins un gène similaire dans B.

Notons que le *min* vient de la lègère asymétrie des résultats de BLAST lors d'un scan réciproque d'un génome A contre un génome B. Cette asymétrie provient du calcul de la P-value qui fait intervenir la taille de la banque criblée.

La distribution des scores $\mathcal{S}$ observés pour chacun des 5 groupes est donnée sur la Figure 5.12.

On constate que l'ordre d'homogénéité des groupes est : **bacteria** < **proteo** < **firmi** < **gamma** < **entero**.

Le groupe **bacteria** est très hétérogène (de l'ordre de 10% en moyenne de gènes similaires pour deux génomes du groupe). Le groupe **entero** est le plus homogène (60% en moyenne de gènes similaires). Les groupes **firmi** et **gamma** présentent des caractéristiques voisines en terme de pourcentage de gènes similaires.

Une façon légèrement différente de représenter ceci, utile en pratique pour comprendre le comportement des programmes, consiste à regarder, pour chaque groupe, le nombre de liens de correspondance par la relation de similarité c'est-à-dire le nombre d'arêtes inter-strates dans le graphe de données stratifié. Plus précisément, une quantité intéressante est, le nombre (ou pourcentage) de gènes qui présentent au moins un lien vers k autres couches ($k = 1, \ldots, 9$). La figure 5.13

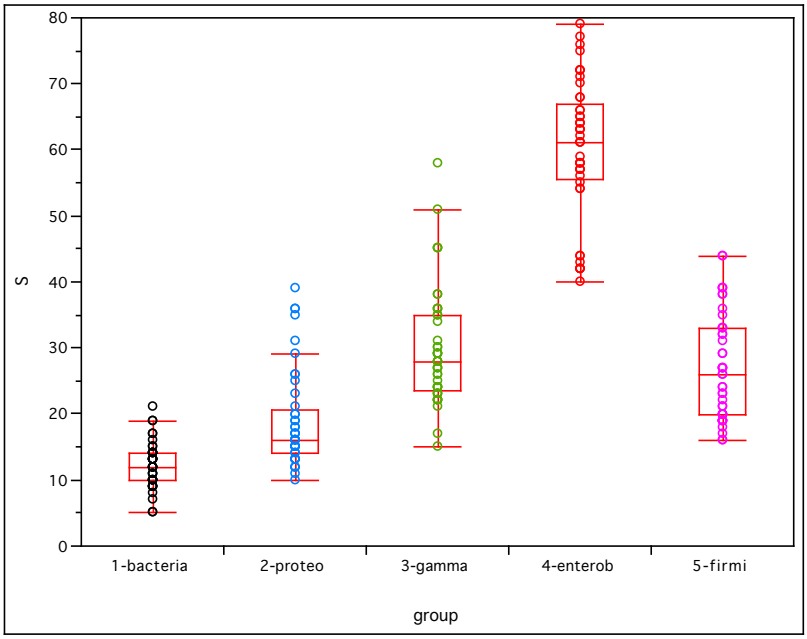


FIGURE 5.12 – Distribution des scores de similarité entre génomes ($\mathcal{S}$) pour chacun des groupes bactériens. *Le bord inférieur des boîtes rouges représente le quantile-25 (c'est-à-dire que 25% des points se situent en dessous de cette valeur), le bord supérieur représente le quantile-75 (25% des points se situent au dessus de cette valeur). La boîte ("quantile box") contient dont 50% des points autour de la médiane, sa hauteur s'appelle la distance interquantile. La médiane est représentée par le trait au milieu de la boîte. Les "moustaches" ("whiskers") s'étendent de part et d'autre de la boîte jusqu'au premier point situé à au moins 1.5 fois la distance interquantile si un tel point existe ou au dernier point de la distribution s'il n'existe pas. Ceci permet d'identifier les "outliers" éventuels.*

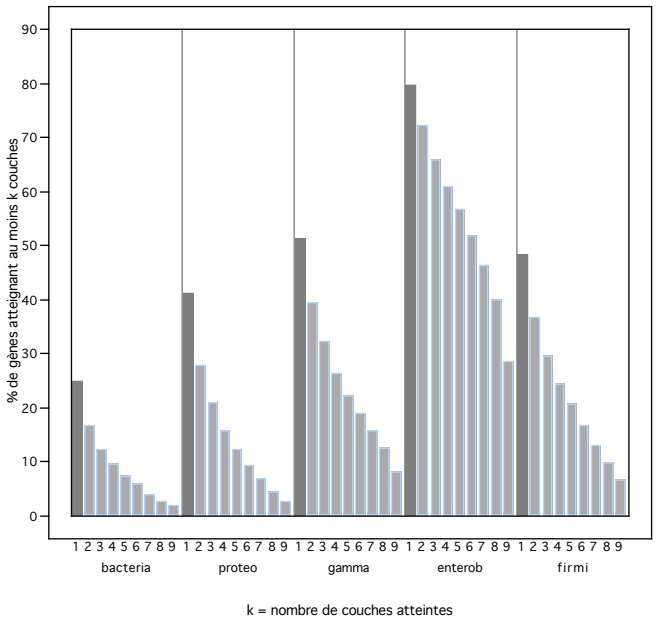


FIGURE 5.13 – Pour chacun des groupes bactérien, pourcentage de gènes présentant au moins un lien vers k autres couches, pour des valeurs de k comprises entre 1 et 9.

représente ainsi, pour chaque groupe, le pourcentage de gènes (en ordonnée) qui présentent au moins un lien vers k (abscisse) autres couches. Cette distribution permet d'estimer grossièrement le nombre de nœuds jokers qui seront nécessaires (paramètre $maxstar$). Ainsi, pour le groupe des **entero** près d'un tiers des gènes présentent un lien vers les 9 autres couches. Alors que pour le groupe **bacteria** les liens à deux couches représentent moins de 20% des gènes.

5.4.4 Temps d'exécution

Pour toutes les expériences qui suivent, qu'il s'agisse d'OTF ou de I-ADHORE, les données primaires sont celles définies au paragraphe précédent (cinq groupes bactériens et la relation de similarité inter-strates définie par le score d'alignement BLAST seuillé comme indiqué précédemment).

Concernant OTF, il a été paramétré - sauf indication contraire - avec les valeurs suivantes :

- fermeture transitive du graphe de données : $deltagap = 5$ (on autorisera donc des "trous" d'au plus 5 gènes)
- conservation partielle de la topologie : $deltashuf = \infty$ (pas de contrainte de

conservation de la topologie, toutes les permutations de gènes sont autorisées)
- taille minimale d'un synton ("mineltsize") = 3 gènes
- les empilements sont définis comme des empilements-γ-denses partiels de S (Section 5.1.1.3) et le paramètre de densité γ sera une des variables de l'analyse (γ varie entre 10% et 100%).
- le quorum par strates (q) est introduit sous la forme du paramètre "maxstar" qui indique le nombre maximum de jokers dans un empilement. Il s'agit de l'autre variable de l'analyse (variant entre 0 et 5).

Pour I-ADHORE, nous avons utilisé les paramètres par défaut sauf pour les paramètres "gap_size", "tandem_gap_size" et "cluster_gap" qui ont tous été mis à la valeur 5 de manière à correspondre au plus près au paramètre *deltagap* d'OTF.

Enfin le paramètre "anchor_points" correspond dans I-ADHORE au nombre minimal de gènes dans un cluster. Il a été mis à la valeur 3 afin de correspondre au paramètre "mineltsize" d'OTF.

Les temps d'exécution pour différentes valeurs des paramètres γ (densité) et *maxstar* (nombre de nœuds jokers) sont donnés sur la Figure 5.14 en comparaison du temps d'exécution pour I-ADHORE. Les valeurs de paramètres pour lesquels aucun temps n'est donné correspondent soit à un temps de calcul excessif (nous avons borné à 1 heure), soit à une quantité mémoire trop importante (bornée à 1 Gb).

On constate que pour les espèces éloignées (**bacteria**), les temps d'exécution sont très rapides et très inférieurs à ceux d'I-ADHORE. Il convient néanmoins de nuancer ce résultat par le fait qu'il n'y a pas explicitement dans I-ADHORE d'équivalent du paramètre maxstar, c'est-à-dire de quorum, I-ADHORE calcule (de manière heuristique) les "multiplicons" pour 2 à n (ici $n = 10$) génomes (cf Section 2.4.6). Il faudrait donc, pour comparer de manière plus équitable les deux approches, soit sommer l'ensemble des temps d'exécution d'OTF pour toutes les valeurs de maxstar, soit comparer avec une seule exécution d'OTF pour $maxstar = 8$. Le problème est qu'OTF est, contrairement à I-ADHORE, un algorithme exact, ce qui implique la taille du résultat croît exponentiellement lorsque *maxstar* augmente. Nous analyserons ceci un peu plus loin. En conséquence, une comparaison précise des temps d'exécution entre les deux approches reste difficile et les valeurs représentées sur la Figure 5.14 doivent être considérées comme des indications plus qu'une "compétition".

Pour les espèces intermédiaires (**proteo**, **firmi**), les temps restent raisonnables pour les plus fortes valeurs de densité (jusqu'à $\gamma = 50$) mais peuvent croître de manière importante pour les faibles densités. A nouveau, ceci est lié au caractère exhaustif du résultat fourni.

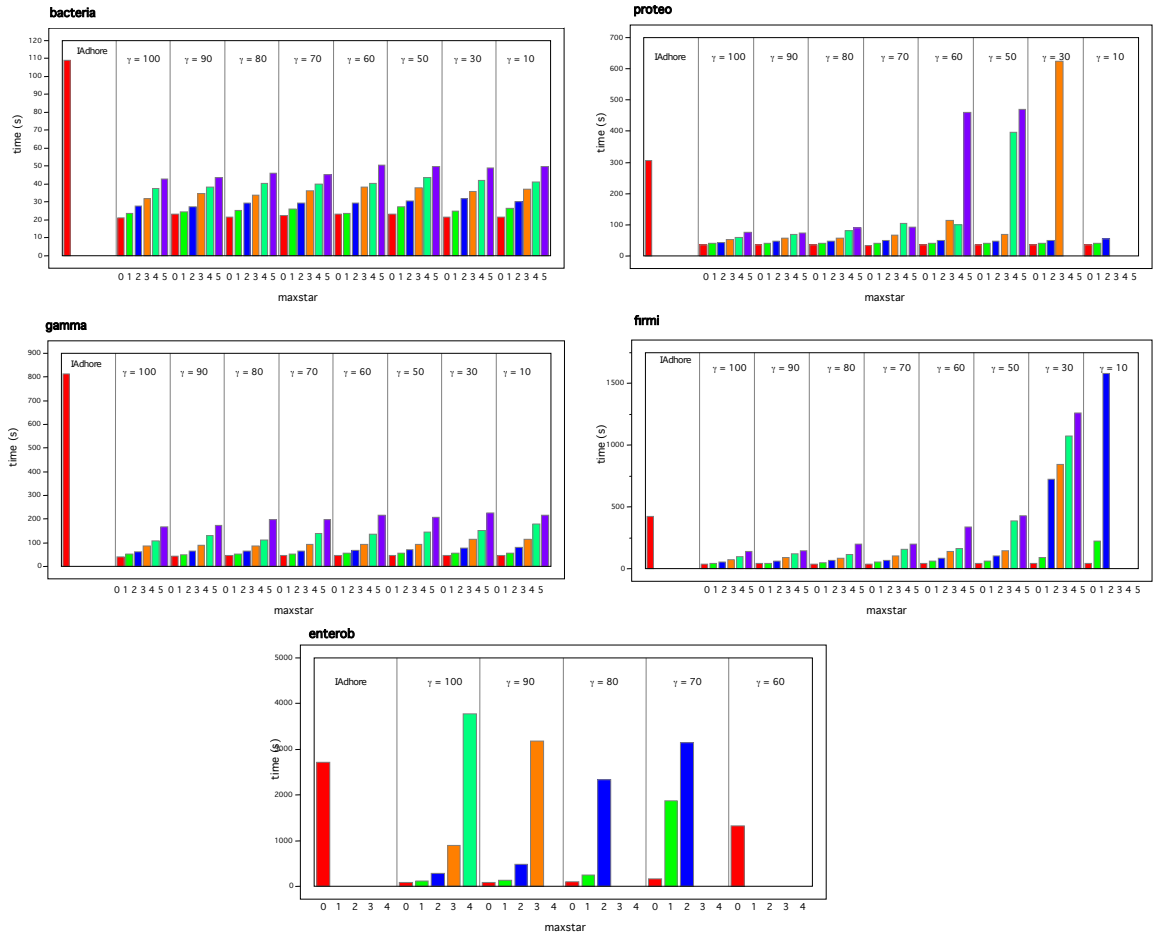


FIGURE 5.14 – Résultats : temps d'exécution de i-ADHoRe et OTF pour chacun des groupes bactériens. Les temps d'exécution d'OTF sont donnés pour des valeurs variables de γ et $maxstar$.

Enfin, pour les espèces très proches (**entero**), le calcul n'est réalisable que pour les très fortes densités ($\gamma = 100$ ou 90), au delà, le temps et le nombre de solutions sont prohibitifs. Dans le cas d'espèces aussi voisines, les empilements sont quasiment systématiquement des cliques mais l'existence de duplications de gènes (en particulier en tandem) rend la combinatoire explosive. Nous reviendrons à la fin de ce document (Chapitre 6.1) sur la question de ces duplications.

A ce stade, il est intéressant de s'arrêter quelques instants sur la relation entre le temps d'exécution et le nombre de solutions trouvées c'est à dire la taille de la sortie produite.

Il existe différentes façons de mesurer cette taille :

1. en nombre de connectons trouvés,
2. en nombre total d'empilements retenus,
3. en nombre total de gènes différents impliqués dans ces empilements.

En pratique les deux premières méthodes sont sensiblement équivalentes (à un facteur constant près). Ceci vient du fait que, quels que soient les paramètres employés, la distribution de taille des connectons est dominée par les petites tailles et il y a donc proportionnalité entre le nombre de connectons et le nombre d'empilements.

La Figure 5.15 montre donc le temps d'exécution en fonction soit du nombre de connectons finalement trouvés (courbes de gauche), soit du nombre de gènes différents impliqués dans ces connectons (courbes de droite). On constate que, pour **bacteria** et **gamma**, ce temps est pratiquement linéaire, mais avec une pente qui dépend du paramètre de densité : plus ce paramètre est faible, plus la pente est importante car le nombre d'empilements réalisables croît. En revanche, pour les espèces très proches (**enterob**), la relation n'est plus linéaire. Expérimentalement elle semble varier de manière quadratique avec le nombre de solutions. Une interprétation possible de ceci est que, pour les espèces éloignées, la combinatoire des empilements (c'est à dire liée à la relation de similarité S) est faible : pour un ensemble de gènes "homologues", il n'y a que peu de façons de réaliser les empilements. En revanche, pour les espèces proches, les groupes de gènes homologues forment pratiquement des cliques de S et l'algorithme doit générer beaucoup plus de combinaisons.

La comparaison directe des résultats d'I-ADHoRe et d'OTF n'est pas immédiate. Dans OTF les solutions (connectons) répondent à une définition précise donnée *a priori* alors qu'I-ADHoRe définit ses solutions (multiplicons) de manière procédurale. Ces mutiplicons sont présentés à l'utilisateur sous la forme d'un DAG (graphe orienté acyclique) traduisant les relations d'inclusion entre les niveaux (multiplicons de taille 2, puis multiplicons de taille 3 inclus dedans, *etc*). Il est

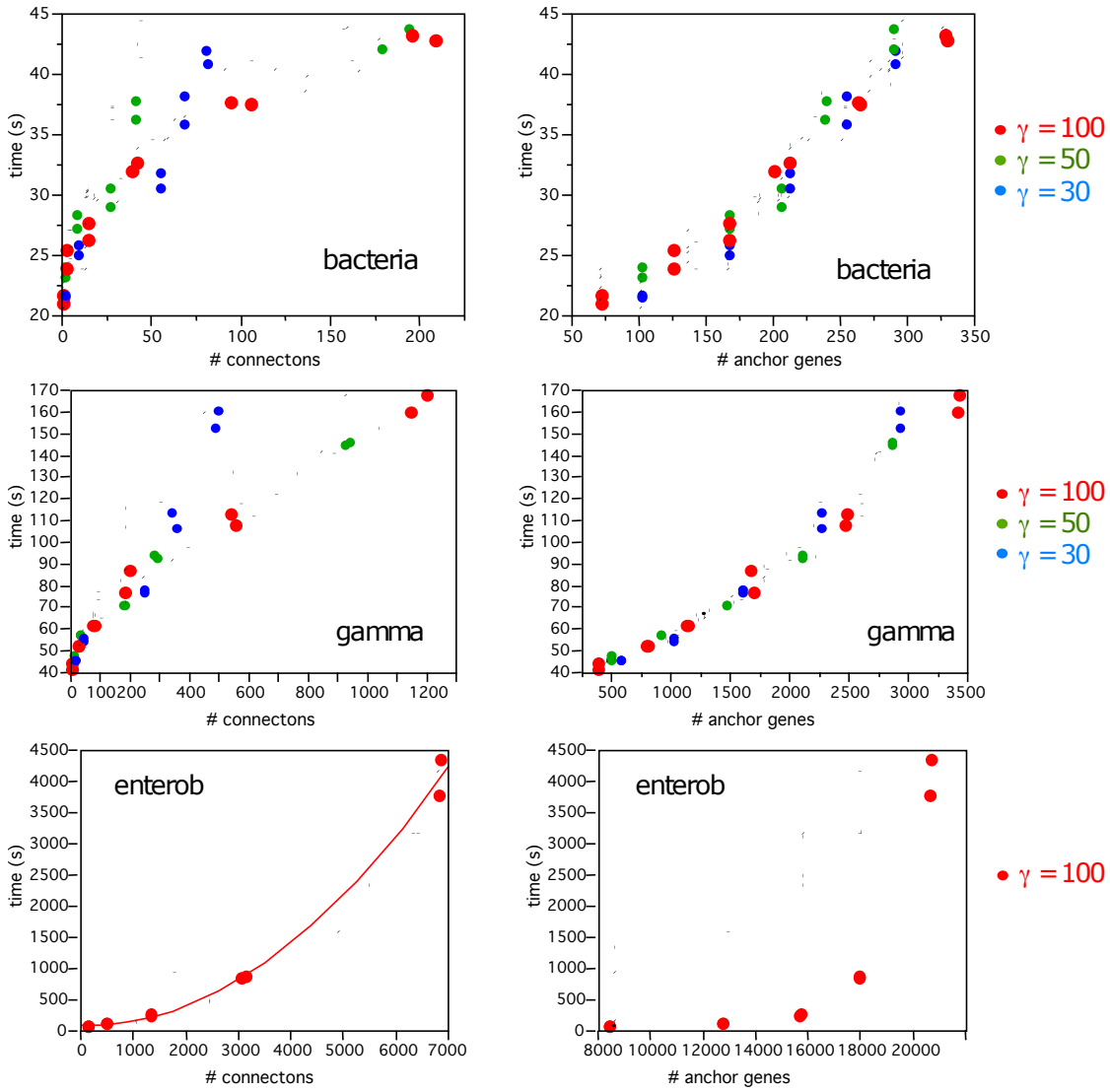


FIGURE 5.15 – Résultats en fonction du nombre de connectons (courbes de gauche) ou du nombre de gènes différents impliqués dans ces connectons (courbes de droite).

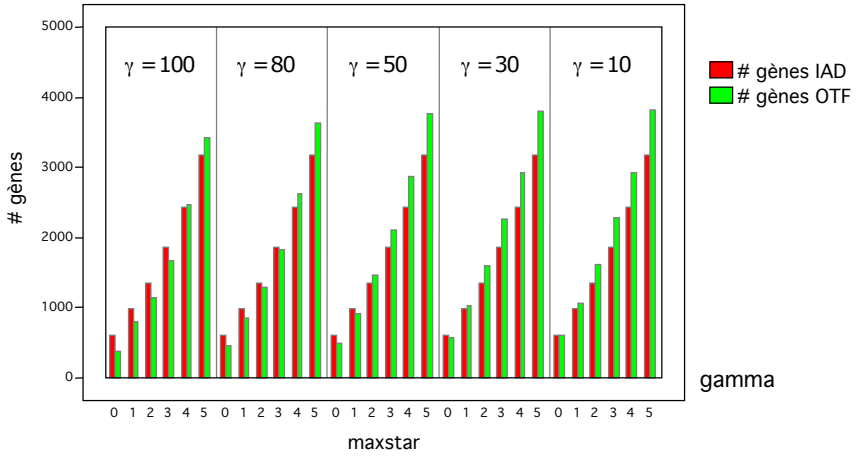


FIGURE 5.16 – Comparaison des cardinalités $|E_{IAD}|$ et $|E_{OTF}|$ pour le groupe **gamma**.

donc difficile de comparer directement cette arborescence avec nos connectons.

Nous avons donc opté pour une approche de comparaison simplifiée, mais aussi plus pragmatique : l'idée est de projeter les résultats (connectons ou multiplicons) sur les génomes de départ et de comparer les ensembles de gènes concernés. Pour un quorum q donné ($2 \leq q \leq n$) nous allons donc conserver dans les résultats de I-ADHORE l'ensemble de tous les multiplicons de niveau au moins égal à q, puis rassembler les gènes concernés en un seul ensemble nommé E_{IAD}. Pour OTF, on rassemblera de la même façon l'ensemble des gènes impliqués dans les connectons obtenus pour le paramètre *maxstar* dans un ensemble E_{OTF}. Les deux ensembles $E_{IAD}0$ et E_{OTF} sont ensuite comparés en utilisant un indice de similarité classique : l'indice de *Jaccard*.

Il nous faut préciser un peu ce que nous entendons par "gènes impliqués" dans un multiplicon ou un connecton. En effet, l'un et l'autre contiennent deux types de gènes : ceux impliqués dans l'alignement (c'est à dire dans un empilement), que nous appelerons "gènes d'ancrage" et ceux introduits au titre de "gaps" que nous appelerons "gènes gaps". Les mesures effectuées en prenant en compte les gènes d'ancrage seuls ou les gènes d'ancrage et de gaps sont très similaires, dans la suite nous ne donnerons donc que les résultats limités aux gènes d'ancrage.

La Figure 5.16 montre la comparaison des cardinalités $|E_{IAD}|$ et $|E_{OTF}|$ pour le groupe **gamma**. On remarque que le paramètre de densité γ a un effet relativement faible sur le nombre de gènes finalement trouvés, nous allons donc, afin de simplifier les graphiques, considérer le nombre moyen de gènes obtenus pour l'ensemble des valeurs de γ.

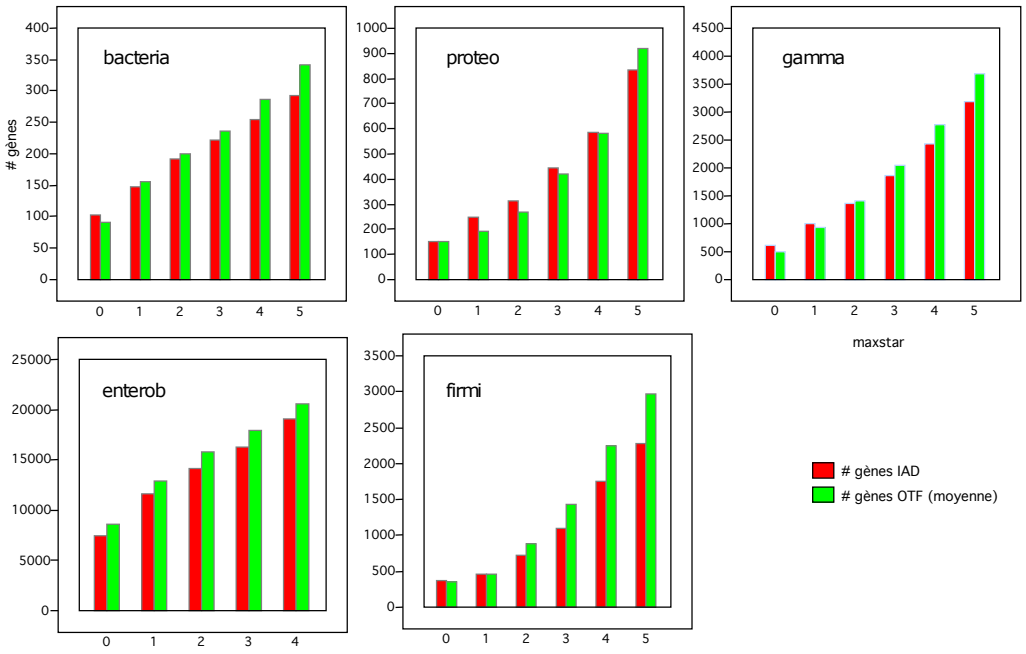


FIGURE 5.17 – Comparaison des cardinalités $|E_{IAD}|$ et $|E_{OTF}|$ moyennées sur γ pour l'ensemble des groupes bactériens.

Ceci est donné sur la Figure 5.17 qui montre la comparaison de $|E_{IAD}|$ et $|E_{OTF}|$ (moyens) pour l'ensemble des groupes bactériens.

On constate que les nombres de gènes trouvés entre I-ADHORE et OTF sont très voisins. Globalement, les gènes impliqués dans les multiplicons / connectons représentent entre 0.5% (**bacteria**) et 40% (**enterob**) du nombre total de gènes dans ces espèces. Pour **bacteria** et **gamma**, I-ADHORE capte légèrement plus de gènes qu'OTF lorsque le quorum est élevé (c'est à dire *maxstar* faible), la situation tend à s'inverser pour les quorums plus faibles. Pour les espèces proches, OTF trouve toujours légèrement plus de gènes.

Il nous reste néanmoins à vérifier qu'il s'agit bien des mêmes gènes. Afin d'analyser précisément ceci, nous allons définir un indice de similarité entre ensemble, inspiré de l'indice de Jaccard :
$J(E_{IAD}, E_{OTF}) = s \times \frac{|E_{IAD} \cap E_{OTF}|}{|E_{IAD} \cup E_{OTF}|}$ avec $s = signe(|E_{OTF}| - |E_{IAD}|)$.

Le signe s nous permet de visualiser simplement qui d'OTF ou I-ADHORE trouve le plus de gènes en empilements ($s > 0$ si $|E_{OTF}| > |E_{IAD}|$).

Les résultats sont présentés sur la Figure 5.18.

Tout d'abord, on observe que quels que soient les réglages, l'indice de *Jaccard* est toujours proche de ± 1. Les ensembles trouvés par les deux approches sont donc

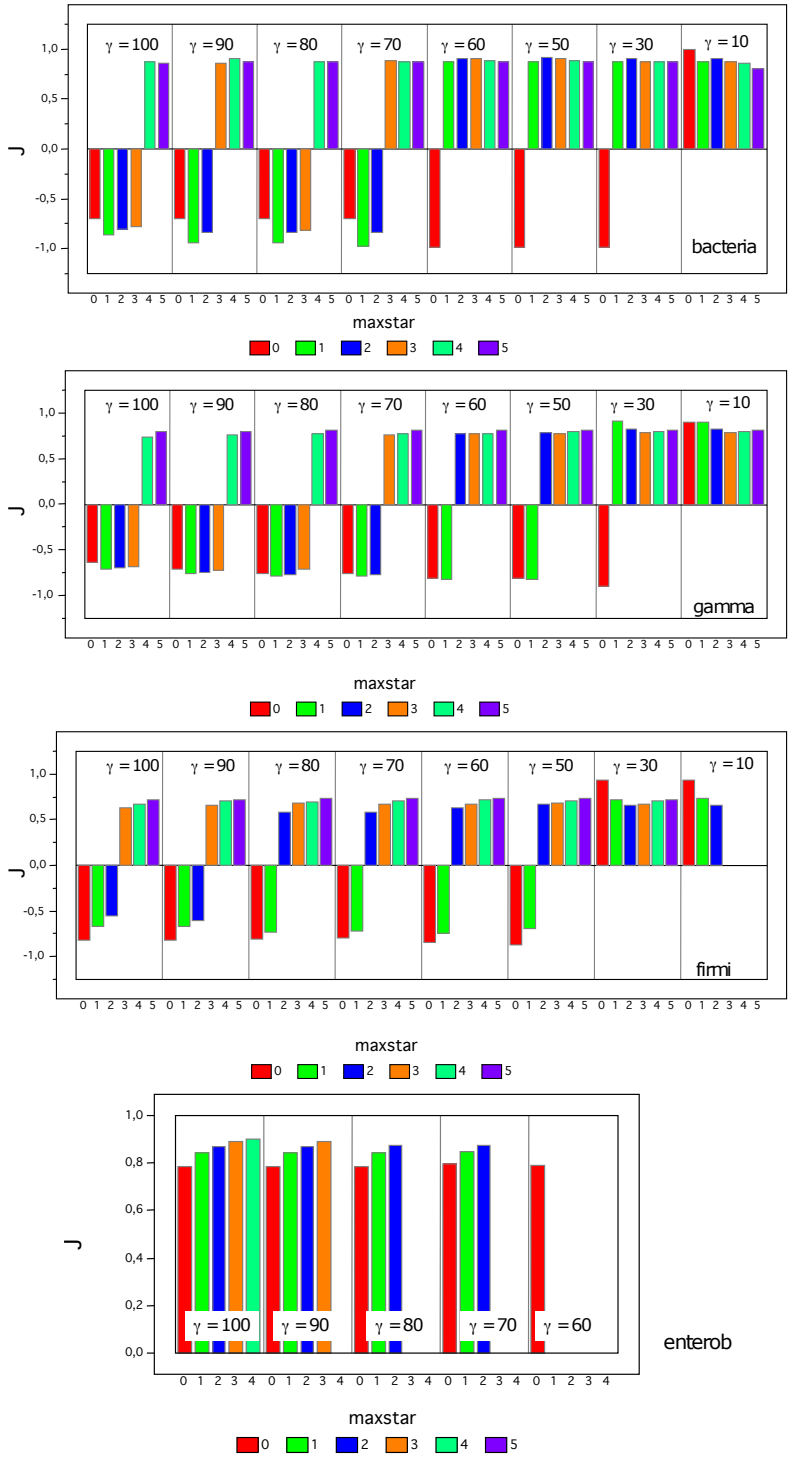


FIGURE 5.18 – Indice de *Jaccard* signé : $J(E_{IAD}, E_{OTF})$ pour les différents groupes bactériens, en fonction de la densité γ et de *maxstar*.

bien les mêmes.

Ensuite on constate que, pour les espèces éloignées, i-ADHoRe trouve lègèrement plus de gènes qu'OTF pour les forts quorums (*maxstar* faible) et les fortes densités ($\gamma = 100$) mais que la situation s'inverse aux quorums ou aux densités plus faibles. Ceci s'interprète aisément en termes de condition d'empilement.

Lorsque la condition est forte (typiquement $\gamma = 100$ c'est à dire si on impose des empilements-cliques partiels), OTF peut manquer des empilements parce qu'un gène sera soit manquant soit trop divergent dans un ou deux espèces. En revanche si on relaxe la condition d'empilement, soit en diminuant γ, soit pour γ fixé en diminuant le quorum, les empilements seront à nouveau détectés.

Il faut noter que, dans i-ADHoRe, les empilements ne sont pas parfaitement bien définis (l'algorithme les construits par une heuristique gloutonne). Au vu des résultats présentés ici, i-ADHoRe correspond en moyenne à des empilements-γ-denses partiels avec $\gamma = 70$.

Pour les espèces proches (**enterob**), on constate que même pour une condition d'empilement en clique ($\gamma = 100$) et de forts quorums ($maxstar = 0$), OTF trouve toujours plus de gènes qu'i-ADHoRe (10 à 20% de plus) . Il y a au moins deux interprétations (complémentaires) possibles à ceci :

1. i-ADHoRe est une heuristique et peut donc manquer des connectons détectés par OTF,
2. i-ADHoRe n'autorise pas explicitement de permutations de l'ordre des gènes dans les multiplicons alors qu'OTF en revanche les accepte.

En pratique les différences observées viennent des deux sources.

En conclusion, nous avons montré que, dans le domaine bactérien, OTF avec quorum se comportait au moins aussi bien qu'i-ADHoRe (qui est le programme de référence), autant en terme de temps d'exécution que de résultats. De notre point de vue, le fait qu'OTF repose sur une définition précise des connectons et qu'il s'agisse d'un algorithme exact, reste un avantage très important. Pour être tout à fait justes il nous faut néanmoins remarquer qu'il reste un cas de figure où i-ADHoRe présente un avantage : c'est celui des synténies intra-chromosomiques (c'est à dire des blocs conservés à l'intérieur d'un même chromosome) qu'il sait parfaitement détecter alors qu'OTF, dans sa forme actuelle ne sait pas les traiter. La modification n'est conceptuellement pas extrêmement difficile à réaliser (il faut ajouter des arêtes de la relation S à l'intérieur d'une même strate) mais nous ne l'avons pas encore implémentée.

Chapitre 6

Améliorations et variantes

Dans ce chapitre, nous allons détailler plusieurs améliorations et variantes de notre algorithme. Nous allons commencer par détailler le problème posé par des nœuds en tandem dans le multigraphe, et proposer des solutions. Ensuite nous introduirons une variante de l'algorithme, permettant de traiter le cas spécifique d'une relation de correspondance qui est une relation d'identité. Nous appliquerons enfin cette variante à un problème biologique particulier : l'analyse des perturbations induites par des virus sur l'interactome humain. Nous montrerons que si nos algorithmes généraux ne sont sans doute pas optimaux dans ce cas spécifique, ils restent néanmoins applicables.

6.1 Problème des tandems

Un problème spécifique qui apparaît lorsqu'on cherche à aligner plusieurs génomes est le problème des duplications en tandem. La plupart des duplications aboutissent en effet à deux copies d'un même gène situées côte à côte sur le chromosome.
Si, sur chacun des n génomes, ces copies sont aussi présentes, cela signifie que l'on risque de devoir construire jusqu'à 2^n empilements, ainsi que toutes les arêtes qui relient ces empilements entre eux. Nous donnons un exemple de nœuds en tandem sur la Figure 6.1.

Nous allons commencer par donner une première solution simple dans le cas de copies exactes qui consiste en un pré-traitement du graphe de données stratifié.

Nous verrons dans un deuxième temps comment traiter le problème plus général des nœuds en tandem qui ne sont pas des copies exactes, ou plus exactement qui n'ont pas strictement le même voisinage selon la relation de correspondance.

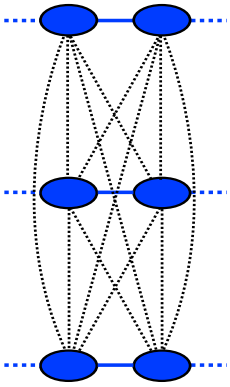


FIGURE 6.1 – Exemple de graphe de données stratifié avec une duplication en tandem apparaissant sur trois génomes. Les deux gènes n'ayant pas encore divergé sur aucun des trois génomes, on retrouve toutes les arêtes de correspondance, on obtient donc 2^3 empilements différents.

6.1.1 Pré-traitement

Etant donné le graphe de données stratifié $D = (\bigcup_i V_i, E_P \cup E_S)$, notons $\mathbb{V}_S(u)$ (resp. $\mathbb{V}_P(u)$) l'ensemble des voisins selon E_S (resp. E_P) d'un nœud u de D. Nous donnons ci-dessous une première définition formelle des nœuds en tandem.

Définition 6.1. *Dans le graphe de données stratifié $D = (\bigcup_i V_i, E_P \cup E_S)$, deux nœuds u, u' du même réseau V_i sont des* nœuds en tandem *si et seulement si $(u, u') \in E_P$ et $\mathbb{V}_S(u) = \mathbb{V}_S(u')$.*

Autrement dit, deux nœuds d'un même réseau sont "en tandem" s'ils sont voisins pour E_P et s'ils ont exactement les mêmes voisins par E_S.

Une première approche naturelle consiste à identifier ces nœuds dans une phase de pré-traitement et à les "comprimer" le temps du calcul des connectons.

En pratique, nous allons remplacer dans le graphe de données stratifié chaque paire de nœuds en tandem (u, u') par un nœud-tandem u'' tel que $\mathbb{V}_{\mathbb{P}}(u'') = \mathbb{V}_{\mathbb{P}}(u) \cup \mathbb{V}_{\mathbb{P}}(u')$ et $\mathbb{V}_{\mathbb{S}}(u'') = \mathbb{V}_{\mathbb{S}}(u') = \mathbb{V}_{\mathbb{S}}(u)$. Nous conservons l'information dans le nœud-tandem, ce qui nous permettra de le "décomprimer" une fois les connectons calculés.

Cette première solution reste exacte, deux nœuds en tandem u et u' ont exactement les mêmes voisins par E_S et sont voisins sur E_P, la condition de maximalité implique donc qu'on ne peut pas avoir dans un même connecton (peu importe la définition choisie) d'empilement contenant u sans avoir un empilement identique avec u' à la place de u.

Par contre en imposant que les voisinages selon E_S soient exactement les mêmes, nous échouons à comprimer un bon nombre de nœuds correspondant à des duplications en tandem qui ont un peu divergé.

6.1.2 Compression à la volée

Dans cette sous-partie, l'objectif va être de traiter les cas de duplications en tandem divergentes. Nous avons choisi une approche consistant à définir une condition générale de compression basée uniquement sur le multigraphe courant. Cette condition définit une relation entre les empilements que nous noterons $\mathcal{C}$.

Après chaque phase d'ajout de strate, nous allons parcourir les empilements du multigraphe courant, identifier les couples (u, v) avec $u = (u_1, \dots u_k)$ et $v = (v_1, \dots v_k)$ tels que $u\mathcal{C}v$, et pour chacun de ces couples nous allons "comprimer" u dans v.
En pratique cela revient à conserver dans v un pointeur vers le n-uplet u, et à supprimer u et toutes les arêtes qui le connectaient dans le multigraphe courant.
L'opération inverse est appelée "décompression", elle consiste à extraire le n-uplet u de v et à le réintroduire ainsi que toutes les arêtes associées dans le multigraphe courant.
Cette dernière opération est coûteuse dans le cas général puisqu'a priori la décompression d'un empilement $u = (u_1, \dots u_k)$ va nous forcer à parcourir les empilements $v = (v_1, \dots v_k)$ et à vérifier pour chaque $i \in [|1, k|]$ l'existence de l'arête (u_i, v_i) dans l'ensemble E_i.

Notons $Decompression(M)$ le résultat de la décompression de M.

Afin de conserver un algorithme exact, nous allons vérifier que pour un multigraphe courant comprimé M, en notant $Partitionner(M) = \{M_1, \dots M_l\}$ les l classes résultant de la partition de M et $AjouterStrate_k(M)$ le multigraphe résultant de l'ajout de la k-ème strate, on a bien :

- $Partitionner(Decompression(M)) = \{Decompression(M_1), \dots Decompression(M_l)\}$.
- $AjouterStrate_k(Decompression(M)) = Decompression(AjouterStrate_k(M))$.

Une première remarque est que nous allons vouloir systématiquement que $u\mathcal{C}v \Rightarrow \forall i \in [|1, k|](u, v) \in E_i''$. Cette condition est en effet une condition suffisante pour que u et v n'appartiennent pas à des classes différentes à l'issue du partitionnement sur les k premières strates. Autrement dit, elle garantit que

$Partitionner(Decompression(M)) = \{Decompression(M_1), \dots Decompression(M_l)\}$.

Pour alléger le texte nous noterons $u \equiv v \Leftrightarrow \forall i \in [|1,k|](u,v) \in E''_i$.

On pourrait définir une condition moins forte, mais cela nous ferait sortir du cas des nœuds en tandem, et cette condition a l'avantage d'être extrêmement simple à tester.

Ensuite, il nous reste à assurer que $AjouterStrate_k(Decompression(M)) = Decompression(AjouterStrate_k(M))$. L'idée est simple : nous allons calculer les prolongements pertinents de u et v, notés respectivement $Prol(u)$ et $Prol(v)$. Pour les prolongements dans $Prol(u) \cap Prol(v)$ nous allons prolonger l'empilement comprimé, pour le reste des prolongements nous allons prolonger l'empilement décomprimé correspondant.

On remarque que la décompression n'aura donc lieu qu'au moment de l'ajout d'une nouvelle strate. Tout l'enjeu du choix de la condition $\mathcal{C}$ va être de maximiser le nombre de compressions tout en minimisant la complexité des décompressions dans la phase $AjouterStrate$.

Dans le multigraphe courant $(\mathcal{V}', \mathcal{E}'_1, \dots, \mathcal{E}'_n)$, $\forall i \in [|1,k|]$, $\forall u \in \mathcal{V}'$ notons $\mathbb{V}_i(u)$ l'ensemble $\{v \in \mathcal{V}' \text{ tq } (u,v) \in \mathcal{E}'_i\}$. Autrement dit $\mathbb{V}_i(u)$ est le voisinage de l'empilement u d'après la i-ème couleur. Notons aussi $\mathbb{V}(u) = \bigcup_{i \in [|1,k|]} \mathbb{V}_i(u)$ l'ensemble des voisins de u dans le multigraphe courant.

Une première condition de compression est la condition $\mathcal{C}_1$:

Condition de Compression 1. $u\mathcal{C}_1 v \Leftrightarrow u \equiv v$ *et* $\forall i \in [|1,k|]$, $\mathbb{V}_i(u) = \mathbb{V}_i(v)$.

Informellement, cela signifie que deux empilements u et v seront comprimés s'ils sont liés par toutes les couleurs et s'ils ont exactement les mêmes voisins sur chacune des couleurs. En pratique, cela nous garantit que pour un empilement u comprimé dans un empilement v, la décompression de u peut se faire de la façon suivante :

1. ajouter u comme nœud du multigraphe courant,
2. $\forall i \in [|1,k|]$, $\forall w \in \mathbb{V}_i(v)$, ajouter (u,w) dans E''_i,
3. ajouter (u,v) dans chacun des E''_i.

Cette première condition est très contrainte et peut donc échouer à comprimer efficacement certains empilements. Afin d'en comprimer plus, nous proposons une

seconde condition moins contrainte, qui se contente d'une inclusion $\mathbb{V}_i(u) \subseteq \mathbb{V}_i(v)$. Cette condition est notée $\mathcal{C}_2$.

Condition de Compression 2. *$u\mathcal{C}_2 v \Leftrightarrow u{\equiv}v$ et $\forall i \in [|1,k|]$, $\mathbb{V}_i(u) \subseteq \mathbb{V}_i(v)$ (notons que cela implique que $\mathbb{V}(u) \subseteq \mathbb{V}(v)$) .*

Cela signifie que deux empilements u et v seront comprimés s'ils sont liés par toutes les couleurs et si v possède au moins toutes les arêtes de u. Avec cette condition, la décompression de $u = (u_1, \dots u_k)$ se fait de la façon suivante :

1. ajouter u comme nœud du multigraphe courant,
2. $\forall i \in [|1,k|]$, $\forall w = (w_1, \dots w_k) \in \mathbb{V}_i(v)$, si dans le graphe de données stratifié $(u_i, w_i) \in E_i$, alors ajouter (u,w) dans E_i'',
3. ajouter (u,v) dans chacun des E_i''.

L'affaiblissement de la condition de compression nous force donc maintenant à aller chercher des informations dans le graphe de données stratifié. Cela reste assez peu coûteux en pratique parce que la condition nous permet de nous limiter aux arêtes sortantes de v, nous ne faisons en pratique que $\sum_{i \in [|1,k|]} |\mathbb{V}_i(v)|$ tests.

Enfin, une troisième condition de compression encore moins contrainte est la condition $\mathcal{C}_3$:

Condition de Compression 3. *$u\mathcal{C}_3 v \Leftrightarrow u{\equiv}v$ et $\mathbb{V}(u) \subseteq \mathbb{V}(v)$.*

Autrement dit, deux empilements u et v seront comprimés s'ils sont liés par toutes les couleurs et si v est voisin de tous les voisins de u. Le processus de décompression associé est le suivant :

1. ajouter u comme nœud du multigraphe courant,
2. $\forall w = (w_1, \dots w_k) \in \mathbb{V}(v)$, $\forall i \in [|1,k|]$ si dans le graphe de données stratifié $(u_i, w_i) \in E_i$, alors ajouter (u,w) dans E_i'',
3. ajouter (u,v) dans chacun des E_i''.

En pratique, cela correspond cette fois à $k \times |\mathbb{V}(v)|$ tests. Pour un empilement v relié à chacun de ses voisins par toutes les couleurs, les conditions $\mathcal{C}_2$ et $\mathcal{C}_3$ correspondront donc à la même complexité, mais dès que v est relié par moins de couleurs, la condition $\mathcal{C}_2$ est plus efficace.

```
Notes d'implémentation.
Le paramètre compressor peut prendre les valeurs suivantes, du
moins comprimé au plus comprimé (mais aussi le plus coûteux en
décompression) :
 ident : condition 1,
 insetcolor : condition 2,
 inset : condition 3.
```

6.2 Variante : cas d'une relation d'identité

Nous appellons relation d'identité une relation de correspondance transitive et fonctionnelle telle que chaque nœud soit en correspondance avec un unique nœud dans chacun des autres réseaux.

D'un point de vue biologique, ce type de relation serait par exemple utilisée pour relier des réseaux primaires représentant la co-expression d'un même ensemble de gènes dans plusieurs expériences. Dans ce type de réseau, les nœuds sont des gènes et deux gènes sont reliés par une arête s'ils sont co-exprimés, la co-expression étant expérimentalement suggérée, par exemple, par des expériences de type *micro-array*. On dispose donc de n réseaux primaires avec les mêmes nœuds (gènes) mais des arêtes différentes.
Une application typique est, par exemple, de retrouver les ensembles de gènes co-exprimés dans au moins q des n expériences.

Voyons maintenant ce que cela change d'un point de vue algorithmique. Soit un graphe de données stratifié D avec ce type de relation de correspondance.
Nous pouvons facilement construire le multigraphe d'alignement associé, puisque le calcul des empilements est immédiat (un empilement correspond à une classe de la relation d'identité).
Les difficultés liées au calcul du MGA étant inexistantes dans ce cas de figure précis, le calcul optimal des connectons reposera entièrement sur l'algorithme de partitionnement choisi et l'algorithme de construction à la volée n'est d'aucun intérêt pratique.

Par contre, si nous souhaitons introduire un quorum, cette approche redevient intéressante.

En pratique, disposer d'une relation de correspondance qui est une relation d'identité nous garantit que pour chaque empilement courant, nous aurons au

maximum deux prolongements pertinents, le nœud joker et le prolongement normal par la classe d'équivalence. Cette information va nous permettre d'améliorer notablement les performances. En effet, dans le cas général, après chaque ajout d'une strate $(k+1)$, nous procédons systématiquement à un partitionnement complet sur les k premières couleurs, ce qui est raisonnable, puisqu'il est rare que chacun des empilements soient prolongés, et parce que nous voulons limiter au maximum la taille du multigraphe courant.

Dans le cas d'une relation d'identité, par contre, nous savons que chacun des empilements sera prolongé, nous allons donc pouvoir procéder simplement au partitionnement sur la nouvelle strate, puis prolonger chacun des résultats sur la strate suivante. C'est seulement une fois parvenus à la dernière strate que nous relancerons le partitionnement sur toutes les strates.

Intuitivement, ce choix est un choix entre une complexité en temps et en espace : si nous partitionnons sur toutes les couleurs tout de suite, nous garantissons une taille minimale du multigraphe courant, mais nous y consacrons du temps. Dans le cas d'une relation d'identité, la taille n'est plus vraiment un problème, donc nous pouvons nous contenter à chaque fois du partitionnement sur la strate ajoutée, et repousser le calcul du partitionnement complet à la fin.

6.3 Application à l'analyse des perturbations induites par des virus sur l'interactome humain

Nous allons présenter dans cette section un problème spécifique d'alignement local voisin du problème précédent, que nous avons rencontré en discutant avec Vincent Navratil au Laboratoire de Biométrie et Biologie Evolutive à Lyon. Nous commencerons par formaliser le problème posé avant de le résoudre sur les données qu'il nous a fournies en utilisant la variante décrite dans la section précédente.

Notons que notre objectif est simplement ici de montrer la diversité des problèmes biologiques qui peuvent s'exprimer sous la forme d'une recherche de connectons, notre algorithme n'a pas vocation a être optimal sur ce type de données très contraintes.

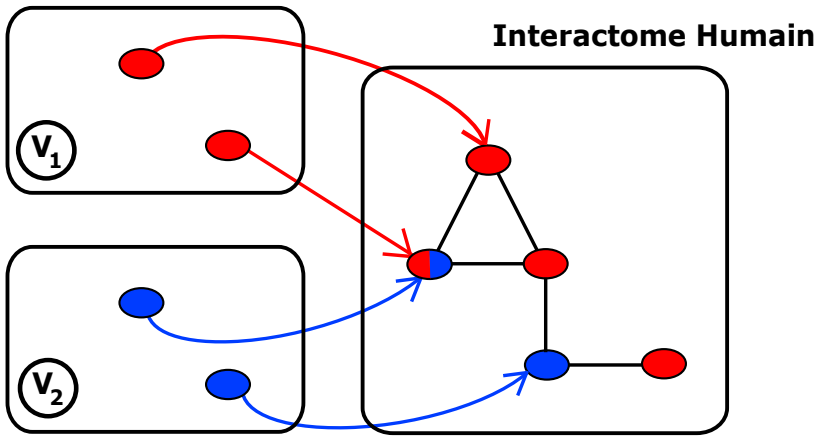


FIGURE 6.2 – Exemple de représentation des perturbations induites par deux virus V_1 et V_2 sur l'interactome humain. On colorie en rouge les nœuds de l'interactome humain qui interagissent avec des protéines du virus V_1 et en bleu ceux qui interagissent avec des protéines du virus V_2.

6.3.1 Formalisation

L'objectif est d'analyser les perturbations (potentiellement) introduites par les protéines virales sur l'interactome humain.

Dans ce but, on dispose d'une part d'un graphe représentant l'interactome humain (dans lequel les nœuds sont des protéines et les arêtes désignent une interaction - éventuellement pondérée - entre deux protéines) et, d'autre part, d'un ensemble de couples d'interaction entre des protéines virales et humaines.

Ces protéines virales sont regroupées en ensembles disjoints, associés à une espèce ou une famille virale, notés V_i, $i \in [|1, \ldots, n|]$. Dans la suite, un tel ensemble V_i sera également désigné par le terme "couleur".

Dans un premier temps, une façon simple de représenter l'effet d'un ensemble de virus sur l'interactome humain est de considérer la coloration induite sur les nœuds humains par les nœuds viraux avec lesquels ils interagissent (Figure 6.2).

Intuitivement, avec cette représentation, l'objectif va être de caractériser les nœuds de l'interactome humain perturbés par des virus, pour essayer de comprendre pourquoi ils sont visés.

Formellement, on peut avoir deux niveaux d'analyse différents :

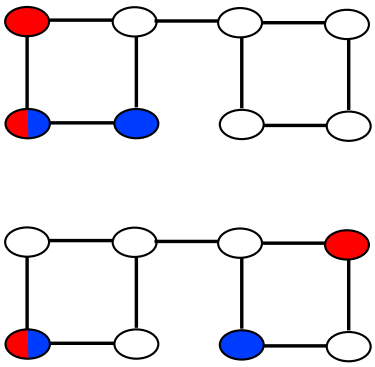


FIGURE 6.3 – Deux exemples d'interactomes perturbés par des virus : les propriétés de centralité et degré sont identiques mais la topologie des nœuds colorés change.

1. une analyse isolée de chaque nœud coloré : on peut s'intéresser à corréler certaines propriétés topologiques des nœuds humains (degré, centralité [1]) à leur coloration. Dans le cas $n = 1$ (ou n faible) l'analyse est simple puisqu'un nœud est coloré ou pas. Dans le cas où n est grand, l'analyse se complique par la combinatoire des couleurs possibles par nœud. On pourra donc, soit simplifier l'analyse en considérant par exemple uniquement le nombre total de couleurs par nœud (quelles qu'elles soient), soit lister une sous-partie des 2^n combinaisons de couleur possibles (par exemple les plus grandes observées).

2. une analyse des réseaux de nœuds colorés : il s'agit d'un second niveau d'analyse dans lequel on ne s'intéresse pas uniquement aux nœuds colorés mais également à leurs voisins colorés (plus ou moins immédiats) et, d'une manière générale, à la topologie des sous-réseaux colorés induits.

Pour illustrer la différence entre ces deux niveaux d'analyse, considérons les deux cas de la figure 6.3.

En termes de degré, centralité *etc.* des nœuds colorés (qu'il s'agisse des rouges ou des bleus), les deux cas sont identiques et vont produire les mêmes résultats statistiques. Ce qui change entre les deux cas, c'est la topologie des sous-réseaux colorés, par exemple le fait que les nœuds colorés sont adjacents dans le premier cas et pas dans le second. Une façon de caractériser ceci est d'étudier les mêmes paramètres que dans le cas (1) (degré, centralité) mais en se limitant maintenant aux sous-réseaux colorés. La difficulté principale tient, comme précédemment, au fait qu'il faudrait, en toute rigueur, tester les 2^n combinaisons de couleurs possibles.

1. En anglais cette propriété est appellée "betweenness".

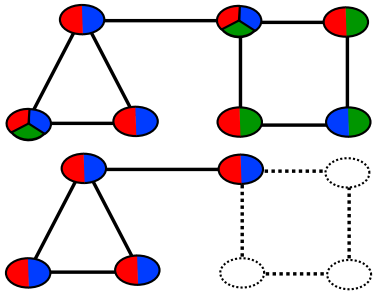


FIGURE 6.4 – En haut, exemple d'interactome perturbé par trois virus . En bas, le plus grand ensemble de nœuds de cet interactome qui sont connectés et perturbés par au moins $q = 2$ virus identiques.

Il est donc crucial, dans ce second type d'analyse, de pouvoir identifier rapidement les sous-réseaux potentiellement intéressants. Ceci nous amène naturellement à un premier type de question :

Problème 1. *Etant donné un graphe $G = (V, E)$ et n couleurs $i \in [|1, n|]$ associées aux nœuds, trouver le plus grand ensemble V' de nœuds connectés colorés par au moins q couleurs identiques (avec $1 \leq q \leq n$).*

Prenons comme exemple le graphe de la figure 6.4 à trois couleurs ($n = 3$). Pour $q = 2$, le plus grand ensemble (solution unique dans ce cas) est obtenu pour les couleurs $\{bleu, rouge\}$, alors que pour $q = 1$, la plus grande solution (couleur rouge) est de taille 6.

Une variante, utile en pratique, du problème précédent est :

Problème 2. *Etant donné un graphe $G = (V, E)$ et n couleurs $i \in [|1, n|]$ associées aux nœuds et un entier K ($1 \leq K \leq |V|$) trouver tous les ensembles de taille supérieure à K de nœuds connectés colorés par au moins q couleurs identiques (avec $1 \leq q \leq n$).*

Ces deux problèmes peuvent être résolus de la façon brutale suivante :

1. énumérer les C_n^q combinaisons de couleurs,
2. pour chaque combinaison rechercher les composantes connexes du graphe induit,
3. conserver la plus grande composante connexe (ou bien les composantes connexes de taille $> K$ dans le cas du problème 2).

Nous allons adopter une approche moins brutale, en adaptant le problème à notre formalisme, puis en appliquant OTF.

En pratique, nous allons transposer les couleurs aux arêtes de la façon suivante : pour u, $v \in V$, l'arête (u, v) sera colorée par $i \in [|1, n|]$ si et seulement si u et v sont tous les deux colorés par i. Notons $couleurs(v)$ l'ensemble des couleurs d'un nœud $v \in V$.

Pour un graphe $G = (V, E)$, on construit le multigraphe suivant : $\mathcal{M}(G) = (V, E_1, \dots E_n)$, où $(u, v) \in E_i \Leftrightarrow i \in couleurs(u) \cap couleurs(v)$.

Dès lors, on peut reformuler nos deux problèmes :

Problème 3. *Etant donné un graphe $G = (V, E)$ et n couleurs $i \in [|1, n|]$ associées aux nœuds, trouver le plus grand connecton partiel de $\mathcal{M}(\mathcal{G})$ pour un quorum $1 \leq q \leq n$.*

Problème 4. *Etant donné un graphe $G = (V, E)$, n couleurs $i \in [|1, n|]$ associées aux nœuds et un entier K ($1 \leq K \leq |V|$), trouver les connectons partiels de $\mathcal{M}(\mathcal{G})$ pour un quorum $1 \leq q \leq n$ de taille supérieure à K.*

Le Problème 3 équivaut au Problème 1, et le Problème 4 équivaut au Problème 2. Résoudre les Problèmes 3 et 4 est facile : on se retrouve dans le cas de la Section 6.2, la relation est une relation d'identité, le multigraphe est déjà construit et il ne nous reste plus qu'à le parcourir.

OTF va donc pouvoir résoudre ces problèmes, de façon plus efficace que l'approche brutale, mais moins efficace qu'une approche dédiée.

6.3.2 Données

Les données ont été fournies par V. Navratil [NdCM+08] et extraites de la base de données $VirHostNet$. On dispose en pratique de deux ensembles d'interactions PPI : un interactome humain-humain et un interactome virus-humain. Les interactions humain-humain sont représentées sous la forme d'un graphe comportant 22983 nœuds (protéines) et 69334 arêtes (interactions).
Parmi les 22983 nœuds, on remarque que seuls 10545 sont connectés par au moins une arête, les autres nœuds sont des singletons.
Notons H_H l'ensemble des nœuds de degré ≥ 1, et notons $I_H = (H_H, E_H)$ l'**interactome humain réduit**, c'est à dire privé des nœuds "singletons".
Dans I_H, la distribution des degrés est maximum en 1 et décroit assez lentement. le degré maximum est de 1038, la moyenne est à 13 et la médiane à 5.

Les données concernant l'interactome virus-humain sont fournies non par virus individuel mais par famille virale. Cet interactome est filtré de façon à ne conserver

quorum (# couleurs)	Taille de la plus grande solution	Nombre de solutions maximales
2	42	1
3	11	1
4	6	1
5	5	1
6	2	2

TABLE 6.1 – Solutions du Problème 4 pour des quorums différents.

que les familles virales contenant au moins 15 interactions. Au total cela représente 13 familles de virus et 1768 interactions, qui sont résumées dans la Figure 6.5. Dans la suite une "couleur" sera donc associée à une famille virale.

On constate une assez forte hétérogénéité entre les familles. Certaines familles (*Retrovirus*, *Herpes*) donnent lieu à beaucoup d'interactions et touchent beaucoup plus de protéines humaines. Il est néanmoins important de noter que ces valeurs ne sont pas normalisées par la taille (en protéines) des virus ou par le nombre de représentants viraux dans chaque famille.
Enfin, la Figure 6.5 donne la distribution du nombre de protéines humaines touchées par nombre de couleurs. On remarque que la majorité des protéines sont touchées par une seule couleur (*i.e.* une seule famille virale) mais que 67 protéines sont touchées par 2 familles ou plus et que 3 protéines humaines sont touchées par 7 couleurs simultanément.

On constate une saturation, c'est à dire que lorsque le nombre d'interactions croît, le nombre de protéines humaines touchées croît moins vite. Ce qui signifie que les mêmes protéines humaines sont touchées par plusieurs protéines virales (de la même famille).

6.3.3 Analyse des sous-réseaux colorés conservés

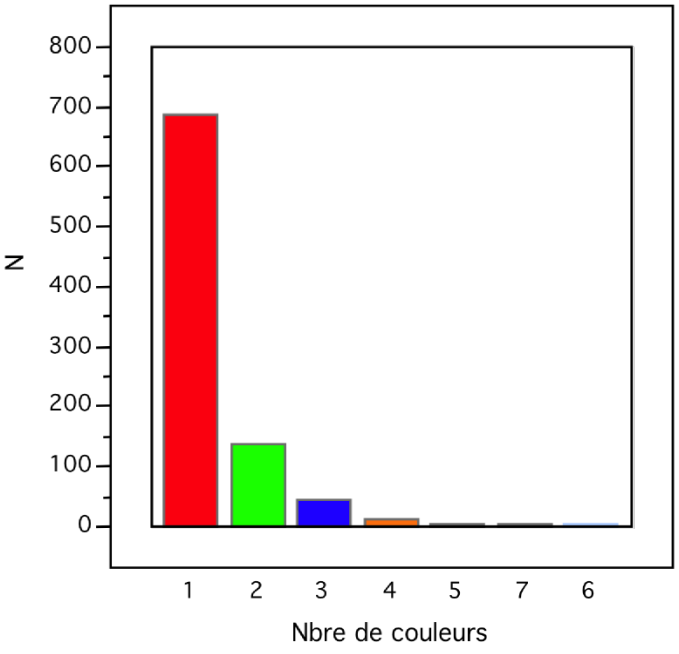
Nous commençons par résoudre le Problème 3 pour différents nombres minimums de couleurs (c'est-à-dire différents quorums) : les résultats sont récapitulés dans la Table 6.1.
Notons que l'exécution en pratique est instantanée.

Pour un quorum de 7 couleurs, les solutions sont des singletons (ce qui signifie que les 3 nœuds à 7 couleurs ne sont pas connectés entre eux). Le plus grand quorum pour des connectons de taille au moins 2 est donc $q = 6$.

Il existe 2 solutions (de taille 2) à 6 couleurs (les 6 couleurs sont les mêmes

famille	NInter	NProtH
Poxviridae	30	26
Herpesviridae	404	279
Hepadnaviridae	91	88
Adenoviridae	71	52
Parvoviridae	16	13
Flaviviridae	214	192
Coronaviridae	23	20
Paramyxoviridae	43	20
Orthomyxoviridae	47	30
Bunyaviridae	20	12
Retroviridae	470	343
Papillomaviridae	299	166
Polyomaviridae	40	32

NInter = Nbre d'interactions vers le protéome humain
NProtH = Nbre de protéines humaines touchées



Nbre de couleurs	Effectif	Fréquence
1	686	0,769
2	139	0,156
3	43	0,048
4	12	0,013
5	4	0,004
6	5	0,006
7	3	0,003
Total	892	

FIGURE 6.5 – En haut, la liste des familles virales utilisées. $NInter$ est le nombre d'interactions virus-humain, $NProtH$ est le nombre de protéines humaines touchées. En bas, le graphique et le tableau donnent la distribution des nombres de protéines humaines touchées en fonction du nombre de couleurs.

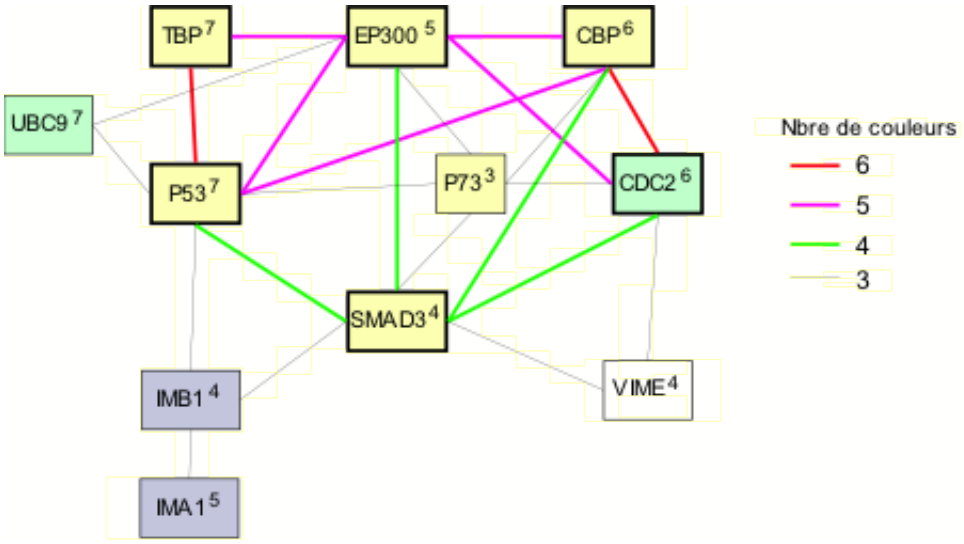


FIGURE 6.6 – Composante connexe de l'interactome humain touchée par 3 ou plus familles virales. En exposant des noms, nous donnons le nombre de couleurs pour le nœud. Concernant les arêtes, la couleur rouge relie les protéines de chacune des solutions à 6 couleurs. En ajoutant les arêtes mauves, nous obtenons les arêtes reliant les protéines de la solution à 5 couleurs. Avec les arêtes vertes en plus, nous retrouvons la solution à 4 couleurs. Enfin avec les arêtes noires, nous avons toutes les protéines de la figure, qui forment exactement la solution à 3 couleurs. Pour les nœuds, la coloration en jaune indique qu'il s'agit de régulateurs transcriptionnels, en vert nous avons les protéine intervenant dans le cycle cellulaire, et enfin en gris, celles qui ont un rôle de transport.

pour les 2 solutions). Les autres solutions maximales sont uniques. L'analyse de ces solutions montre qu'elles sont, en réalité, incluses les unes dans les autres. C'est-à-dire que les deux solutions à 6 couleurs fusionnent avec une 5ème protéine pour former la solution à 5 couleurs, puis une autre protéine s'aggrège à cette solution pour former la solution à 4 couleurs, enfin 5 autres protéines se rajoutent pour former la solution de taille 11 à 3 couleurs.

Ces graphes emboîtés sont représentés sur la Figure 6.6 et la liste des 11 protéines concernées est donnée dans la Figure 6.8.

L'examen des annotations (*Swiss-Prot*) associées à ces 11 protéines montre (Figure 6.8) que :

- la plupart d'entre elles (8) sont annotées comme impliquées dans des interactions hôte-virus,
- 6 sont annotées comme des régulateurs de la transcription - parmi elles on

id	Swiss-Prot	description	deg.	Nbre de couleurs
ENSG00000100393	EP300	Histone acetyltransferase p300 (EC 2.3.1.48) (E1A-associated protein p300). [Source:Uniprot/SWISSPROT,Acc:Q09472]	228	5
ENSG00000170312	CDC2	Cell division control protein 2 homolog (EC 2.7.11.22) (EC 2.7.11.23) (p34 protein kinase) (Cyclin-dependent kinase 1) (CDK1). [Source:Uniprot/SWISSPROT,Acc:P06493]	617	6
ENSG00000005339	CBP	CREB-binding protein (EC 2.3.1.48). [Source:Uniprot/SWISSPROT,Acc:Q92793]	212	6
ENSG00000112592	TBP	TATA-box-binding protein (TATA-box factor) (TATA-binding factor) (TATA sequence-binding protein) (Transcription initiation factor TFIID TBP subunit). [Source:Uniprot/SWISSPROT,Acc:P20226]	114	7
ENSG00000141510	P53	Cellular tumor antigen p53 (Tumor suppressor p53) (Phosphoprotein p53) (Antigen NY-CO-13). [Source:Uniprot/SWISSPROT,Acc:P04637]	303	6
ENSG00000166949	SMAD3	Mothers against decapentaplegic homolog 3 (SMAD 3) (Mothers against DPP homolog 3) (Mad3) (hMAD-3) (JV15-2) (hSMAD3). [Source:Uniprot/SWISSPROT,Acc:P84022]	199	4
ENSG00000114030	IMA1	Importin alpha-1 subunit (Karyopherin alpha-1 subunit) (SRP1-beta) (RAG cohort protein 2) (Nucleoprotein interactor 1) (NPI-1). [Source:Uniprot/SWISSPROT,Acc:P52294]	24	5
ENSG00000108424	IMB1	Importin beta-1 subunit (Karyopherin beta-1 subunit) (Nuclear factor P97) (Importin 90). [Source:Uniprot/SWISSPROT,Acc:Q14974]	53	4
ENSG00000078900	P73	Tumor protein p73 (p53-like transcription factor) (p53-related protein). [Source:Uniprot/SWISSPROT,Acc:O15350]	41	3
ENSG00000103275	UBC9	SUMO-conjugating enzyme UBC9 (EC 6.3.2.-) (SUMO-protein ligase) (Ubiquitin-conjugating enzyme E2 I) (Ubiquitin-protein ligase I) (Ubiquitin carrier protein I) (Ubiquitin carrier protein 9) (p18). [Source:Uniprot/SWISSPROT,Acc:P63279]	126	7
ENSG00000026025	VIME	Vimentin. [Source:Uniprot/SWISSPROT,Acc:P08670]	128	4

FIGURE 6.7 – Liste des 11 protéines apparaissant dans toutes les solutions au Problème 4 lorsqu'on dépasse 3 couleurs. Pour une protéine donnée, on fournit son degré ($deg.$) et le nombre de familles virales qui interagissent avec elle ($Nbredecouleurs$).

trouve des protéines très étudiées comme la $P53$.

Nous avons représenté sur la Figure 6.8 ces annotations par un code couleur sur les nœuds. On constate que le "noyau dur" des 5 protéines solutions à $q = 6, 5, 4$ (boîtes en gras) est essentiellement constitué de ces régulateurs, les autres protéines "périphériques" étant généralement porteuses des autres annotations (cycle cellulaire, transport).

Enfin, une dernière remarque porte sur l'arité de ces protéines qui montre qu'il s'agit généralement de nœuds très connectés. Cette connection peut refléter une réalité biologique ou simplement le fait que ces protéines ont été très étudiées ($P53$).

Cette interaction massive des protéines virales avec ce groupe de protéines fortement impliquées dans la régulation transcriptionnelle évoque au biologiste avec qui nous avons discuté (V. Navratil) un phénomène bien connu de l'interaction hôte-virus : une tendance des virus à moduler la réponse transcriptionnelle de la cellule au cours de l'infection en lien plus ou moins direct avec l'échappement des virus au processus cellulaire de lutte antivirale et avec pour conséquence collatérale un impact de ces interactions dans la mise en place de cancers (*Pappillomavirus*, *Retrovirus*, *Adenovirus*).

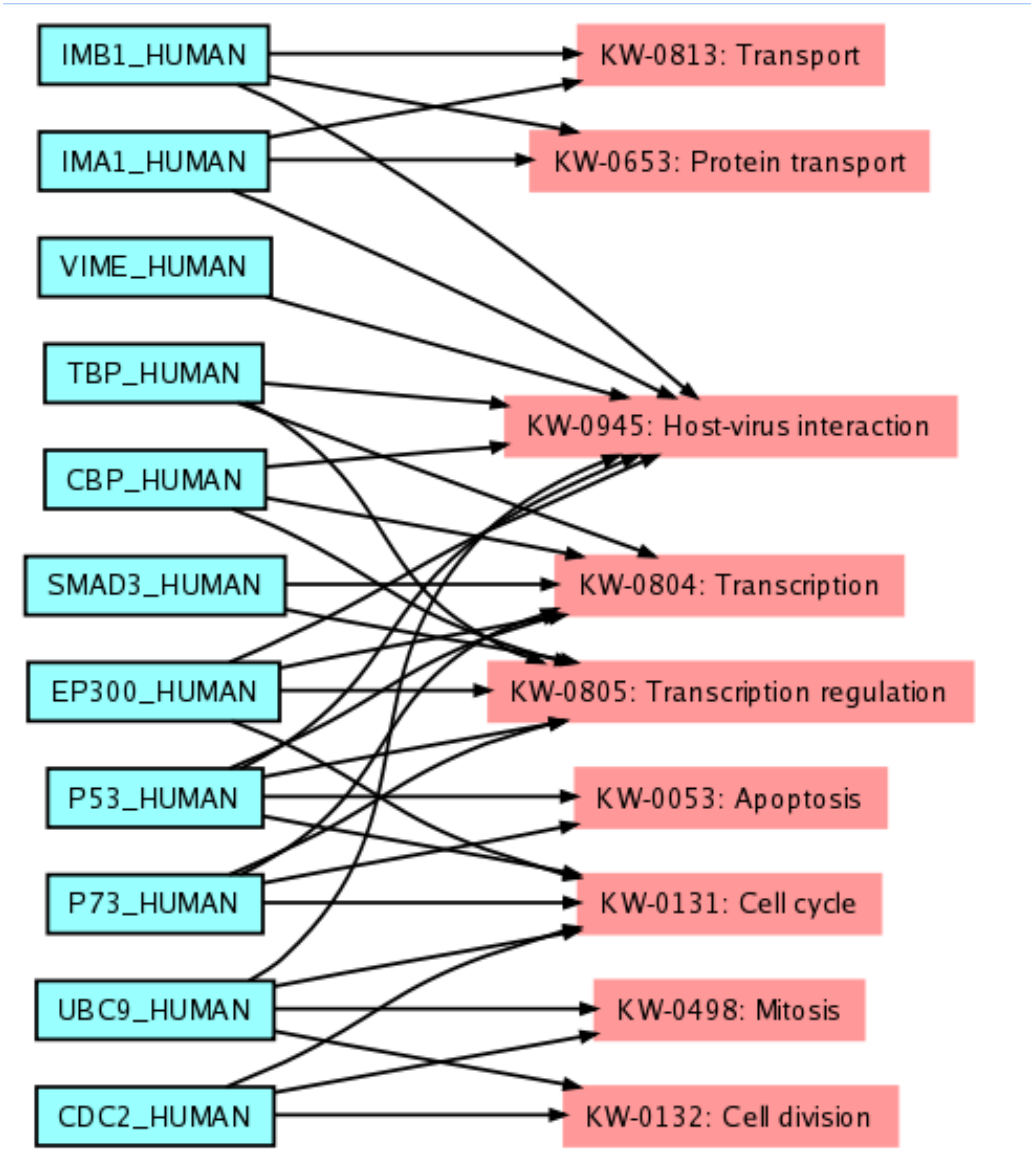


FIGURE 6.8 – Annotations *Swiss-Prot* (*keywords*) associées aux 11 protéines apparaissant dans toutes les solutions lorsqu'on dépasse 3 couleurs.

Chapitre 7

Conclusion et perspectives

Nous avons présenté dans cette thèse une approche exacte et générique d'alignement local multiple de réseaux biologiques.

Dans un premier temps, nous avons établi le formalisme du graphe de données stratifié, puis celui du multigraphe d'alignement (MGA) inspiré des travaux de Sharan [KBS08], Ogata [OFGK00] et Boyer [BML$^+$05]. Nous avons défini la notion d'alignement local dans le graphe de données stratifié, en introduisant des choix à plusieurs niveaux :

- choix des empilements (option *aggregator*), ce qui revient à décider quand n nœuds - un pris dans chaque réseau primaire - sont alignés,
- choix de la taille des trous autorisés (option *deltagap*), c'est-à-dire du nombre de nœuds qui peuvent être manquants sur un réseau sans briser l'alignement,
- choix de la taille maximale des réarrangements autorisés (option $deltashuf$), ce qui permet de forcer une conservation partielle de la topologie sur les réseaux. Pour les valeurs extrêmes, on retrouve donc d'un côté les alignements qui autorisent tous les réarrangements, et de l'autre ceux qui forcent une conservation totale de la topologie.

Nous avons montré que les alignements locaux du graphe de données stratifié correspondent en pratique à des sous-graphes du MGA que nous avons appelés connectons.

Constatant que ces connectons forment une partition des nœuds du MGA, nous avons ensuite proposé un algorithme permettant de les calculer, qui procède par construction et partitionnement à la volée du MGA. Nous évitons donc la construction complète du MGA, ce qui nous permet de traiter de façon efficace l'alignement de grands nombres de réseaux.

Dans un second temps, nous avons proposé deux formalismes d'alignement lo-

caux partiels, qui imposent seulement la présence de q nœuds par empilement : les alignements partiels par strate et les alignements partiels par empilement. Nous avons détaillé les algorithmes associés, puis nous avons proposé différentes améliorations, et des variantes adaptées à des problèmes biologiques particuliers.

Concernant les perspectives de ce travail, une partie des améliorations proposées dans cette thèse avaient pour objectif de limiter autant que possible la taille du multigraphe d'alignement courant, mais le problème n'est pas entièrement résolu. Il reste notamment un cas pratique "pathologique" dans cette approche : le cas où S est une relation d'équivalence différente de l'identité. Dans ce cas très particulier, en effet, le nombre d'empilements comme le nombre de connectons peut devenir extrêmement grand. Une première piste serait une modification des compresseurs : utiliser des conditions de compression moins contraintes, et comprimer encore plus, au lieu d'éviter seulement la construction des arêtes liées aux nœuds comprimés nous devons pouvoir éviter aussi leur prolongement. Cela permettrait de résoudre le premier problème mais pas l'augmentation exponentielle du nombre de solutions. Une solution serait de recourir à une compression des multigraphes d'alignements eux-mêmes, qui interviendrait après la procédure *Partitionner*.

Un second travail qui nous semble intéressant est le développement de méthodes d'analyse statistique de la pertinence des résultats obtenus en fonction du graphe de données stratifié (plus principalement de la relation de correspondance).

Une autre idée serait, au lieu d'appliquer une analyse statistique aux résultats de notre formalisme combinatoire, d'introduire ces statistiques directement dans le formalisme, ce qui nous permettrait au cours de l'algorithme de ne pas explorer les branches dont on sait déjà qu'elles produiront des connectons non-pertinents.

Enfin, nous nous sommes, dans ce travail, essentiellement consacrés à l'aspect algorithmique au détriment des applications. Pour développer cet aspect, nous sommes en relation avec F. Boyer pour une application à l'alignement local multiple avec quorum de données de co-expression (le problème détaillé en Section 6.2), une autre possibilité serait de développer l'approche dans un cas particulier de réseaux primaires, qui seraient des arbres, avec comme objectif d'adapter notre méthode à l'alignement de structures d'ARN.

Chapitre 8

Annexe

8.1 Implémentation en Java

L'ensemble des algorithmes décrits dans cette thèse ont été implémentés en Java. Le package est distribué avec les sources et une documentation sous licence libre CECIL. Il est composé d'environ 170 classes et 18000 lignes de code. Il peut être employé soit sous la forme d'une bibliothèque Java, soit sous la forme d'un programme exécutable (dénommé $Isofun$) muni d'options de réglage.

On trouvera ci-après deux tableaux récapitulant les options du programme principal avec pour chacune une référence aux paragraphes du document dans lesquels la fonctionnalité correspondante est décrite.

8.2 Publications

Le travail des Chapitres 3 et 4 a fait l'objet des publications suivantes :
- $JOBIM$: Recovering isofunctional genes : a synteny-based approach [DBSV08],
- CPM : Multiple Alignment of Biological Networks : A Flexible Approach [DBVS09].

Nous sommes en train de finir de rédiger un article qui recouvre les sections 5.2 (pour l'algorithme) et 5.4 (pour l'application), qui doit être soumis à $BMC\ Bioinformatics$.

Option	Values	Description	Default	Ref.
Options for all algorithms				
algo	bfs dfs otf otfq otfqg	select algorithm otfq : partiel par empilement otfqg : partiel par strate	otf	4.1 4.1 4.2 5.3 5.2
minsize	<integer>	ccc min size	2	3.6
mineltsize	<integer>	ccc min element size	2	3.6
aggregator	clique center cc dense[[absolute|relative]<integer>]	choose node aggregator dense absolute <min_links> dense relative <%link>	clique	4.2.2 5.1.1
deltagap	<integer>+	delta for gaps on each color. use 0 for no gap. if you specify a single value, it will apply on all colors	0 (no gap)	3.8.1
deltashuf	<integer>+	delta for shuffle on each color. use 0 for no shuffling, use "inf(inity)?\for infinite shuffling. If you specify a single value, it will apply on all colors	infinity (i.e. infinite shuffling on all colors)	3.8.2
colors	<integer>+	keep only specified colors for processing	keep all colors	
optimizer	off local[min|max] global[min|max] user<integer>+	select colors permutation scheduler (user <colors>+ : user specified order)	local min	4.3.2
compressor	off ident insetcolor inset	select graph compression mode	Off	6.1.2

Option	Values	Description	Default	Ref.
Options for OTF quorum algorithms				
maxstar	<integer>	set the maximum number of jokers (otfql and otfqg algo)	0	5.2.3
mincore	<integer>	set the minimum number of core nodes (otfql and otfqg algo)	0	5.3.3
lookahead	terminals allofcolor colorpath allpath oneway twoway	set lookahead connector (otfql algo) prolongement des empilements vides dans un multigraphe courant non-vide	oneway	5.3.3
Preprocessing options				
check	on off	check datagraph before processing	On	
cleanup	on off	cleanup datagraph before processing	On	4.3.1
tandem	off on[<integer> [<integer>]]	cleanup tandems before processing on [minfrac [minsize]]	Off	6.1
Postprocessing and reporting options				
reportgaps	on off	report on gene contiguity and gaps in syntons	Off	
memdog	on off	report memory used	Ff	
Misc options (for developpers)				
debug	on off	turn on/off debugger	Off	
verbose	on off	turn on/off verbosity	Off	
deltaeps	<integer>+	delta closure for epsilon	0 (no closure)	
ismulti	on off	tell input graph is already a multigraph (xfs algo only)	Off	
nbproc	<integer>	nb of concurrent processors to use (1=no-concurrency 0=max-available	1	

Bibliographie

[AA03] E. Alm et A.P. Arkin : Biological networks. *Curr. Opin. Struct. Biol.*, 13(2):193–202, Avril 2003.

[ABB+00] M. Ashburner, C.A. Ball, J.A. Blake, D. Botstein, H. Butler, J.M. Cherry, A.P Davis, K. Dolinski, S.S. Dwight, J.T. Eppig, M.A Harris, D.P. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J.C. Matese, J.E. Richardson, M. Ringwald, G.M. Rubin et G. Sherlock : Gene ontology : tool for the unification of biology. the gene ontology consortium. *Nature Genetics*, 25(1):25–29, Mai 2000.

[AGM+90] S.F. Altschul, W. Gish, W. Miller, E.W. Myers et D.J. Lipman : Basic local alignment search tool. *J. Mol. Biol.*, 215(3):403–410, Octobre 1990.

[AKdCL09] F. Ay, T. Kahveci et V. de Crécy-Lagard : A fast and accurate algorithm for comparative analysis of metabolic pathways. *J. Bioinform. Comput. Biol.*, 7(3):389–428, 2009.

[BCR02] A. Bergeron, S. Corteel et M. Raffinot : The algorithmic of gene teams. *In WABI : International Workshop on Algorithms in Bioinformatics*, volume 2452 de *Lecture Notes in Computer Science*, pages 464–476, 2002.

[BDC+03] M. Brudno, C. B. Do, G. M. Cooper, M. F. Kim, E. Davydov, E. D. Green, A. Sidow et S. Batzoglou : Lagan and multilagan : efficient tools for large-scale multiple alignment of genomic DNA. *Genome Res.*, 13(4):721–731, Avril 2003.

[BDP03] N. Bray, I. Dubchak et L. Pachter : Avid : A global alignment program. *Genome Res.*, 13(1):97–102, Janvier 2003.

[Ber01] A. Bergeron : A very elementary presentation of the hannenhalli-pevzner theory. *In CPM : 12th Symposium on Combinatorial Pattern Matching*, volume 2089 de *Lecture Notes in Computer Science*, pages 106–117, 2001.

[BML+05] F. Boyer, A. Morgat, L. Labarre, J. Pothier et A. Viari : Syntons, metabolons and interactons : an exact graph-theoretical approach for exploring neighbourhood between genomic and functional data. *Bioinformatics*, 21(23):4209–4215, 2005.

[BPM+00] S. Batzoglou, L. Pachter, J.P. Mesirov, B. Berger et E.S. Lander : Human and mouse gene structure : Comparative analysis and application to exon prediction. *Genome Res.*, 10(7):950–958, Juillet 2000.

[CCV03] P.P. Calabrese, S. Chakravarty et T.J. Vision : Fast identification and statistical evaluation of segmental homologies in comparative maps. *In ISMB (Supplement of Bioinformatics)*, volume 19, pages 74–80, 2003.

[CFF+06] R. Caspi, H. Foerster, C.A. Fulcher, R. Hopkinson, J. Ingraham, P. Kaipa, M. Krummenacker, S. Paley, J. Pick, S.Y. Rhee, Tissier C., P. Zhang et P.D. Karp : Metacyc : a multiorganism database of metabolic pathways and enzymes. *Nucl. Acids Res.*, 34(Database issue):D623–D631, Janvier 2006.

[CLM+06] D. Che, G. Li, F. Mao, H. Wu et Y. Xu : Detecting uber-operons in prokaryotic genomes. *Nucl. Acids Res.*, 34(8):2418–2427, Mai 2006.

[CSD+04] X. Chen, Z. Su, P. Dam, B. Palenik, Y. Xu et T. Jiang : Operon prediction by comparative genomics : an application to the synechococcus sp. wh8102 genome. *Nucl. Acids Res.*, 32(7):2147–2157, 2004.

[Dat08] S.V. Date : The rosetta stone method. *Bioinformatics*, 453:169–180, 2008.

[DBSV08] Y.-P. Denielou, F. Boyer, M.-F. Sagot et A. Viari : Recovering isofunctional genes : a synteny-based approach. *In Actes des Journes Ouvertes de Biologie, Informatique et Mathématiques JOBIM 2008*, pages 11–16, 2008.

[DBVS09] Y.-P. Denielou, F. Boyer, A. Viari et M.-F. Sagot : Multiple alignment of biological networks : A flexible approach. *In CPM : 20th Symposium on Combinatorial Pattern Matching*, volume 5577 de *Lecture Notes in Computer Science*, pages 263–273, 2009.

[Did03] G. Didier : Common intervals of two sequences. *In Algorithms in Bioinformatics Proceedings*, volume 2812 de *Lecture Notes in Bioinformatics*, pages 17–24, 2003.

[DKF+99] A. L. Delcher, S. Kasif, R. D. Fleischmann, J. Peterson, O. White et S. L. Salzberg : Alignment of whole genomes. *Nucl. Acids Res.*, 27(11):2369–2376, Juin 1999.

[DMBP04] A.C.E. Darling, B. Mau, F.R. Blattner et N. T. Perna : Mauve : Multiple alignment of conserved genomic sequence with rearrangements. *Genome Res.*, 14(7):1394–1403, Juillet 2004.

[DT07] J. Dutkowski et J. Tiuryn : Identification of functional modules from conserved ancestral protein protein interactions. *Bioinformatics*, 23(13), 2007.

[EVDO02] A.J. Enright, S. Van Dongen et C.A. Ouzounis : An efficient algorithm for large-scale detection of protein families. *Nucl. Acids Res.*, 30(7):1575–1584, Avril 2002.

[Fel89] J. Felsenstein : Phylip - phylogeny inference package (version 3.2). *Cladistics*, 5:164–166, 1989.

[Fit00] W. Fitch : Homology a personal view on some of the problems. *Trends in Genetics*, 16(5):227–231, Mai 2000.

[FNS+06] J. Flannick, A. Novak, B.S. Srinivasan, H.H. McAdams et S. Batzoglou : Graemlin : general and robust alignment of multiple large interaction networks. *Genome Res.*, 16(9):1169–1181, 2006.

[FTM+08] R. D. Finn, J. Tate, J. Mistry, P.C. Coggill, S.J. Sammut, H.-R. Hotz, G. Ceric, K. Forslund, S.R. Eddy, E.L.L. Sonnhammer et A. Bateman : The pfam protein families database. *Nucl. Acids Res.*, 36(suppl_1):D281–D288, Janvier 2008.

[GBK+02] A.-C. C. Gavin, M. Bösche, R. Krause, P. Grandi, M. Marzioch, A. Bauer, J. Schultz, J.M. Rick, A.-M. M. Michon, C.-M. M. Cruciat, M. Remor, C. Höfert, M. Schelder, M. Brajenovic, H. Ruffner, A. Merino, K. Klein, M. Hudak, D. Dickson, T. Rudi, V. Gnau, A. Bauch, S. Bastuck, B. Huhse, C. Leutwein, M.-A. A. Heurtier, R.R. Copley, A. Edelmann, E. Querfurth, V. Rybin, G. Drewes, M. Raida, T. Bouwmeester, P. Bork, B. Seraphin, B. Kuster, G. Neubauer et G. Superti-Furga : Functional organization of the yeast proteome by systematic analysis of protein complexes. *Nature*, 415(6868):141–147, Janvier 2002.

[GKM00] M.S. Gelfand, E.V. Koonin et A.A. Mironov : Prediction of transcription regulatory sites in archaea by a comparative genomic approach. *Nucl. Acids Res.*, 28(3):695–705, 2000.

[GVL96] G. H. Golub et C. Van Loan : *Matrix Computations, 3rd edition.* JHU Press, 1996.

[HDWS04] B.J. Haas, A.L. Delcher, J.L. Wortman et S. Salzberg : DAGchainer : a tool for mining segmental genome duplications and synteny. *Bioinformatics*, 20(18):3643–3646, 2004.

[HG05] X. He et M.H. Goldwasser : Identifying conserved gene clusters in the presence of homology families. *J. Comput. Biol.*, 12(6):638–656, 2005.

[HMGB03] S. Hampson, A. McLysaght, B. Gaut et P. Baldi : Lineup : statistical detection of chromosomal homology with application to plant comparative genomics. *Genome Res.*, 13(5):999–1010, Mai 2003.

[HP99] S. Hannenhalli et P. Pevzner : Transforming cabbage into turnip : Polynomial algorithm for sorting signed permutations by reversals. *JACM : Journal of the ACM*, 46:178–189, 1999.

[HPR04] M. Habib, C. Paul et M. Raffinot : Maximal common connected sets of interval graphs. *In CPM : 15th Symposium on Combinatorial Pattern Matching*, volume 3109 de *Lecture Notes in Computer Science*, pages 347–358, 2004.

[HS01] S. Heber et J. Stoye : Algorithms for finding gene clusters. *In WABI : International Workshop on Algorithms in Bioinformatics, LNCS*, 2001.

[ICO+01] T. Ito, T. Chiba, R. Ozawa, M. Yoshida, M. Hattori et Y. Sakaki : A comprehensive two-hybrid analysis to explore the yeast protein interactome. *Proc. Natl. Acad. Sci. USA*, 98(8):4569–4574, Avril 2001.

[KBS08] M. Kalaev, V. Bafna et R. Sharan : Fast and accurate alignment of multiple protein networks. *In International Conference on Research in Computational Molecular Biology RECOMB 2008*, volume 4955 de *Lecture Notes in Computer Science*, pages 246–256, 2008.

[KCVGC+05] I.M. Keseler, J. Collado-Vides, S. Gama-Castro, J. Ingraham, S. Paley, I.T. Paulsen, M. Peralta-Gil et P.D. Karp : Ecocyc : a comprehensive database resource for escherichia coli. *Nucl. Acids Res.*, 33(Database issue):D334–D337, Janvier 2005.

[KCY05] S. Kim, J.-H. Choi et J. Yang : Gene teams with relaxed proximity constraint. *In IEEE Computational Systems Bioinformatics Conference (CSB 2005)*, pages 44–55, 2005.

[KG00] M. Kanehisa et S. Goto : KEGG : kyoto encyclopedia of genes and genomes. *Nucl. Acids Res.*, 28(1):27–30, Janvier 2000.

[KKT+06] M. Koyutürk, Y. Kim, U. Topkara, S. Subramaniam, W. Szpankowski et A. Grama : Pairwise alignment of protein interaction networks. *J. Comput. Biol.*, 13(2):182–199, 2006.

[KSK+03] B.P. Kelley, R. Sharan, R.M. Karp, T. Sittler, D.E. Root, B.R. Stockwell et T. Ideker : Conserved pathways within bacteria and yeast as revealed by global protein network alignment. *Proc. Natl. Acad. Sci. USA*, 100(20):11394–11399, 2003.

[KZ00] W. J. Kent et A. M. Zahler : Conservation, regulation, synteny, and introns in a large-scale c. briggsae-c. elegans genomic alignment. *Genome Res.*, 10(8):1115–1125, Août 2000.

[LAB+04] S. Li, C.M. Armstrong, N. Bertin, H. Ge, S. Milstein, M. Boxem, P.O. Vidalain, J.D. Han, A. Chesneau, T. Hao, D.S. Goldberg, N. Li, M. Martinez, J.F. Rual, P. Lamesch, L. Xu, M. Tewari, S.L. Wong, L.V. Zhang, G.F. Berriz, L. Jacotot, P. Vaglio, J. Reboul, T. Hirozane-Kishikawa, Q. Li, H.W. Gabel, A. Elewa, B. Baumgartner, D.J. Rose, H. Yu, S. Bosak, R. Sequerra, A. Fraser, S.E. Mango, W.M. Saxton, S. Strome, S. Van Den Heuvel, F. Piano, J. Vandenhaute, C. Sardet, M. Gerstein, L. Doucette-Stamm, K.C. Gunsalus, J.W. Harper, M.E. Cusick, F.P. Roth, D.E. Hill et M. Vidal : A map of the interactome network of the metazoan c. elegans. *Science*, 303(5657):540–543, Janvier 2004.

[LCTS08] V. Lacroix, L. Cottret, P. Thébault et M.-F. F. Sagot : An introduction to metabolic networks and their structural analysis. *IEEE/ACM transactions on computational biology and bioinformatics*, 5(4):594–617, 2008.

[LdRdGR08] Y. Li, D. de Ridder, M.J.L. de Groot et M.J.T. Reinders : Metabolic pathway alignment (M-pal) reveals diversity and alternatives in conserved networks. *In Proceedings of the 6th Asia-Pacific Bioinformatics Conference, APBC 2008*, volume 6 de *Advances in Bioinformatics and Computational Biology*, pages 273–286, 2008.

[LHX09] X. Ling, X. He et D. Xin : Detecting gene clusters under evolutionary constraint in a large number of genomes. *Bioinformatics*, 25(5):571–577, 2009.

[LHXH08] X. Ling, X. He, D. Xin et J. Han : Efficiently identifying max-gap clusters in pairwise genome comparison. *J. Comput. Biol.*, 15(6):593–609, 2008.

[MHPF98] H.W. Mewes, H. Hani, F. Pfeiffer et D. Frishman : MIPS : a database for protein sequences and complete genomes. *Nucl. Acids Res.*, 26(1):33–37, 1998.

[NCM01] Z. Ning, A.J. Cox et J.C. Mullikin : SSAHA : A fast search method for large DNA databases. *Genome Res.*, 11(10):1725–1729, Octobre 2001.

[NdCM+08] V. Navratil, B. de Chassey, L. Meyniel, S. Delmotte, C. Gautier, P. Andre, V. Lotteau et C. Rabourdin-Combe : Virhostnet : a knowledge base for the management and the analysis of proteome-wide virus-host interaction networks. *Nucl. Acids Res.*, 37(Database issue):D661–D668, Novembre 2008.

[NW70] S.B. Needleman et C.D. Wunsch : A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.*, pages 443–453, 1970.

[OFGK00] H. Ogata, W. Fujibuchi, S. Goto et M. Kanehisa : A heuristic graph comparison algorithm and its application to detect functionally related enzyme clusters. *Nucl. Acids Res.*, 28(20):4021–4028, Octobre 2000.

[PBMW98] L. Page, S. Brin, R. Motwani et T. Winograd : The pagerank citation ranking : Bringing order to the web. *In Proceedings of the 7th International World Wide Web Conference*, pages 161–172, 1998.

[PBR+05] S. Pasek, A. Bergeron, J.L. Risler, A. Louis, E. Ollivier et M. Raffinot : Identification of genomic features using microsyntenies of domains : Domain teams. *Genome Res.*, 15(6):867–874, 2005.

[PHAA05] M. N. Price, K. H. Huang, E. J. Alm et A. P. Arkin : A novel method for accurate operon predictions in all sequenced prokaryotes. *Nucl. Acids Res.*, 33(3):880–892, 2005.

[PMT+99] M. Pellegrini, E.M. Marcotte, M.J. Thompson, D. Eisenberg et T.O. Yeates : Assigning protein functions by comparative genome analysis : protein phylogenetic profiles. *Proc. Natl. Acad. Sci. USA*, 96:4285–8, 1999.

[PRYLZU05a] R. Y. Pinter, O. Rokhlenko, E. Yeger-Lotem et M. Ziv-Ukelson : Alignment of metabolic pathways. *Bioinformatics*, 21(16):3401–3408, Août 2005.

[PRYLZU05b] R.Y. Pinter, O. Rokhlenko, E. Yeger-Lotem et M. Ziv-Ukelson : Alignment of metabolic pathways. *Bioinformatics*, 21(16):3401–3408, Août 2005.

[RD07] C. Rödelsperger et C. Dieterich : Two graph-based approaches for finding cross-species conserved gene orders. *In Proceedings of the German Conference on Bioinformatics, GCB 2007*, volume 115 de *Lecture Notes in Informatics*, pages 163–173, 2007.

[RD08] C. Rödelsperger et C. Dieterich : Syntenator : multiple gene order alignments with a gene-specific scoring function. *Algorithms for Molecular Biology*, 3:14, Novembre 2008.

[RD10] C. Rödelsperger et C. Dieterich : Cyntenator : progressive gene order alignment of 17 vertebrate genomes. *PloS one*, 5(1): e8861, Janvier 2010.

[SJSVdP08] C. Simillion, K. Janssens, L. Sterck et Y. Van de Peer : i-ADHoRe 2.0 : an improved tool to detect degenerated genomic homology using genomic profiles. *Bioinformatics*, 24(1):127–128, Janvier 2008.

[SM07] A.U. Sinha et J. Meller : Cinteny : flexible analysis and visualization of synteny and genome rearrangements in multiple organisms. *BMC Bioinformatics*, 8:82, Mars 2007.

[SNF+06] B.S. Srinivasan, A.F. Novak, J. Flannick, S. Batzoglou et H.H. McAdams : Integrated protein interaction networks for 11 microbes. *In International Conference on Research in Computational Molecular Biology RECOMB 2006*, volume 3909 de *Lecture Notes in Computer Science*, pages 1–14, 2006.

[SSK+05] R. Sharan, S. Suthram, R.M. Kelley, T. Kuhn, S. McCuine, P. Uetz, T. Sittler, R.M. Karp et T. Ideker : Conserved patterns of protein interaction in multiple species. *Proc. Natl. Acad. Sci. USA*, 102(6):1974–1979, 2005.

[SVSVdP04] C. Simillion, K. Vandepoele, Y. Saeys et Y. Van de Peer : Building genomic profiles for uncovering segmental homology in the twilight zone. *Genome Res.*, 14(6):1095–1106, Juin 2004.

[SW81] T.F. Smith et M.S. Waterman : Identification of common molecular subsequences. *J. Mol. Biol.*, 147:195–197, 1981.

[SXB07] R. Singh, J. Xu et B. Berger : Pairwise global alignment of protein interaction networks by matching neighborhood topology. *In International Conference on Research in Computational Molecular Biology RECOMB 2007*, volume 4453 de *Lecture Notes in Computer Science*, pages 16–31, 2007.

[SXB08] R. Singh, J. Xu et B. Berger : Global alignment of multiple protein interaction networks with application to functional orthology detection. *Proc. Natl. Acad. Sci.*, 105(35):12763–12768, 2008.

[Tes02] G. Tesler : GRIMM : genome rearrangements web server. *Bioinformatics*, 18(3):492–493, 2002.

[THG94] J.D. Thompson, D.G. Higgins et T.J. Gibson : CLUSTAL W : improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucl. Acids Res.*, 22:4673–4680, 1994.

[TMH00] Y. Tohsato, H. Matsuda et A. Hashimoto : A multiple alignment algorithm for metabolic pathway analysis using enzyme hierarchy. *In Proceedings of the 8th International Conference on Intelligent Systems for Molecular (ISMB 2000)*, pages 376–383, 2000.

[TN07] Y. Tohsato et Y. Nishimura : Metabolic pathway alignment based on similarity between chemical structures. *IPSJ Digital Courier*, 48(17):9–18, 2007.

[TS09] W. Tian et N.F. Samatova : Pairwise alignment of interaction networks by fast identification of maximal conserved patterns. *In Pacific Symposium on Biocomputing 2009*, volume 14, pages 99–110, 2009.

[UY00] T. Uno et M. Yagiura : Fast algorithms to enumerate all common intervals of two permutations. *Algorithmica*, 26, 2000.

[VSS+02] K. Vandepoele, Y. Saeys, C. Simillion, J. Raes et Y. Van De Peer : The automatic detection of homologous regions (adhore) and its application to microcolinearity between arabidopsis and rice. *Genome Res.*, 12(11):1792–1801, Novembre 2002.

[WR07] S. Wernicke et F. Rasche : Simple and fast alignment of metabolic pathways by exploiting local diversity. *Bioinformatics*, 23(15):1978–1985, 2007.

[ZPV+08] X. Zeng, J. Pei, I. Vergara, M.J. Nesbitt, K. Wang et N. Chen : Orthocluster : a new tool for mining synteny blocks and applications in comparative genomics. *In EDBT 2008 : Proceedings of the 11th International Conference on Extending Database Technology*, pages 656–667, 2008.