Décomposition et Visualisation de graphes : Applications aux Données Biologiques

Romain Bourqui

N^o d'ordre : 3630

THÈSE

présentée à

L'UNIVERSITÉ BORDEAUX I

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET INFORMATIQUE

par Romain BOURQUI

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : Informatique

Décomposition et Visualisation de graphes : Applications aux Données Biologiques

---

Soutenue le : 24 Octobre 2008

Après avis de :

MMe.	Pascale Kuntz-Cosperec	Professeure	**Rapportrice**
M.	Alexandru Telea ...	Professeur	**Rapporteur**

Devant la Commission d'Examen composée de :

M.	Éric Sopena ...	Professeur	**Président**
MMe.	Marie-France Sagot	Directrice de Recheche	**Examinatrice invitée**
MMe.	Maylis Delest ..	Professeure	**Directrice de thèse**
M.	David Auber ...	Maître de Conférences	**Co-directeur de thèse**

- 2008 -

Décomposition et Visualisation de graphes: Applications aux Données Biologiques

Romain Bourqui

Remerciements

Mes remerciements s'adressent tout d'abord à la région Aquitaine pour avoir financé mes travaux de thèse ainsi qu'à l'Université Bordeaux 1 et au LaBRI pour m'avoir permis de la réaliser dans de bonnes conditions. Je souhaite ensuite remercier tous les membres de mon jury de thèse, Pascale Kuntz-Cosperec, Marie-France Sagot, Eric Sopena et Alexandru Telea pour avoir accepté de relire et permis d'améliorer par leurs remarques mon manuscrit de thèse.

Je tiens aussi à remercier l'ensemble des membres du thème « Visualisation de grandes masses de données » pour leurs qualités humaines indéniables, leur bonne humeur quotidienne ainsi bien sûr que leur humour sans égal (merci Jean-Mi). En particulier, je voudrais remercier Maylis et David qui ont su me donner la chance de réaliser cette thèse avec eux. J'en profite pour signifier toute ma gratitude et mon amitié à David pour le (bon) temps qu'il m'a consacré et l'enthousiasme pour la recherche qu'il m'a transmis.

Je remercie d'autre part tous les membres de ma famille pour le soutien qu'ils m'ont apporté. Je remercie tout particulièrement ma mère qui a toujours su nous pousser à donner le meilleur de nous-mêmes. Je pense d'autre part à mon défunt grand-père Pierre qui m'a donné sa passion pour la culture et la connaissance.

Je tiens enfin à remercier mon amie Sabrina pour m'avoir supporté (dans tous les sens du terme) notamment durant ces derniers mois. Son soutien moral ainsi que sa présence m'ont permis de surmonter nombre de difficultés au cours de ces trois années.

Décomposition et Visualisation de graphes : Applications aux Données Biologiques

Résumé : La quantité d'informations stockée dans les bases de données est en constante augmentation rendant ainsi nécessaire la mise au point de systèmes d'analyse et de visualisation. Nous nous intéressons dans cette thèse aux données relationnelles et plus particulièrement aux données biologiques. Cette thèse s'oriente autour de trois axes principaux : tout d'abord, la décomposition de graphes en groupes d'éléments "similaires" afin de détecter d'éventuelles structures de communauté ; le deuxième aspect consiste à mettre en évidence ces structures dans un système de visualisation, et dans un dernier temps, nous nous intéressons à l'utilisabilité de l'un de ces systèmes de visualisation via une évaluation expérimentale.

Les travaux de cette thèse ont été appliqués sur des données réelles provenant de deux domaines de la biologie : les réseaux métaboliques et les réseaux d'interactions gènes-protéines.

Mots-clef : Visualisation de graphe, décomposition de graphe, évaluation, bioinformatique
Discipline : Informatique

Graph Clustering and Visualization : Application to Biological Data

Abstract : The amount of information stored in databases is constantly increasing making necessary to develop systems for analysis and visualization. In this thesis, we are interested in relational data and in particular, in biological data. This thesis focuses on three main axes : firstly, the decomposition of graph into clusters of "similar" elements in order to detect the community structures ; the second aspect is to highlight these structures in a visualization system ; and thirdly, we are interested in the usability of one of these visualization systems through an experimental evaluation.

The work presented in this thesis was applied on real data from two fields of biology : the metabolic networks and the gene-protein interaction networks.

Keywords : Graph visualization, graph decomposition, evaluation, bioinformatic
Field : Computer Science

Table des matières

Résumé / Abstract **iii**

Table des matières **v**

Table des figures **xi**

Liste des tableaux **xxi**

1 Introduction **1**

1.1 Motivations 1

1.2 Extraction, Décomposition et Filtrage 3

1.3 Plongement Visuel et Rendu 4

1.4 Utilisation par l'expert 4

1.5 Mise en oeuvre d'un système de visualisation d'information 5

1.6 Organisation du mémoire 7

2 Définitions et Notations **11**

2.1 Ensembles 11

2.2 Graphes 11

2.2.1 Définitions 11

2.2.2 Dessin de graphe 19

2.3 Réseaux biologiques 20

2.4 Autres définitions 20

3 Décomposition et Visualisation : Etat de l'art **23**

3.1 Décomposition de graphes 23

3.1.1 Qualité d'une décomposition 24

3.1.2 Approches divisives 25

3.1.3 Approches agglomératives 28

3.1.4 Approches topologiques 28

3.2 Visualisation de graphe 29

3.2.1 Algorithmes de dessin de graphe planaire 29

3.2.2 Algorithmes de dessin de graphe orienté acyclique 31

3.2.3 Algorithmes par analogie physique 33

3.2.4 Approches par échantillonnage et modèle de forces 38

3.2.5 Approches basées sur les matrices 40

3.2.6 Approches basées sur le partitionnement 42

3.3 Visualisation de graphe partitionné 44

3.4 Visualisation de données biologiques : Cas particulier du métabolisme . . 49

3.4.1 Visualisation de voies métaboliques 50

3.4.2 Visualisation de réseaux métaboliques 52

4 Décomposition de graphe **55**

4.1 Introduction 55

4.2 Etat de l'art 56

4.3 Décomposition en composantes hautement connexes 58

4.3.1 Décompositions en composantes 2- et 3-connexes 59

4.3.2 Décomposition en composantes 4-connexes 60

4.4 Evaluer la qualité d'une décomposition chevauchante 62

4.4.1 Densité 62

4.4.2 Modularité 62

4.4.3 Généralisation de MQ 63

4.5 Evaluation de la qualité de la décomposition 64

4.5.1 Données et protocole 64

4.5.2 Résultats et Analyse 65
4.6 Conclusion 71

5 Visualisation de partitionnement de graphes arête-valués : application aux réseaux d'interactions 73

5.1 Problématique 73
5.2 Etat de l'art 74
5.3 Vue d'ensemble de la méthode 75
5.4 Prérequis 77
5.4.1 Algorithme GRIP 77
5.4.2 Diagramme de Voronoï 80
5.5 Algorithme 81
5.5.1 GRIP valué 81
5.5.2 Processus de dessin 86
5.5.3 Post-traitement : Regroupement d'arêtes 88
5.5.4 Résultats 90
5.6 Conclusion 91

6 Visualisation de décompositions de graphes : application aux réseaux métaboliques 93

6.1 Problématique 93
6.2 Contraintes et objectifs spécifiques aux réseaux métaboliques 95
6.3 Outils existants de visualisation 97
6.4 Modèle et données 98
6.4.1 Modèle 98
6.4.2 Données 99
6.5 Méthode 99
6.5.1 Partitionnement multi-échelles 100
6.5.2 Paramétrer les centres d'intérêts 104
6.5.3 Algorithmes de dessin 106
6.6 Résultats : comparaison aux méthodes existantes 111

6.6.1 Temps de calcul 112
6.6.2 Visualisation du réseau entier 113
6.6.3 Visualisation de voies métaboliques 115
6.6.4 Visualisation d'une voie métabolique dans son contexte 116
6.7 Conclusion 117

7 Algorithmes génériques ou dédié : évaluation sur une tâche biologique 121
7.1 Problématique et motivations 121
7.2 Modèle et algorithmes de dessins 122
7.2.1 Pré-traitements 122
7.2.2 Algorithme par modèle de forces 122
7.2.3 Algorithme Hiérarchique 125
7.2.4 Algorithme MetaViz 125
7.3 Méthodologie 126
7.3.1 Tâche et données biologiques 126
7.3.2 Hypothèse *a priori* 127
7.3.3 Protocole expérimental 129
7.3.4 Outils d'évaluation 130
7.3.5 Processus expérimental 132
7.4 Résultats et analyse 132
7.4.1 Résultats quantitatifs 133
7.4.2 Résultats qualitatifs 135
7.5 Discussion 136
7.6 Conclusion 137

8 Conclusion et perspectives 139
8.1 Perspectives 140
8.1.1 Décompositions 140
8.1.2 Visualisation de réseaux métaboliques 140
8.1.3 Autre perspective 141

Bibliographie **143**

Résumé / Abstract

x

Table des figures

1.1 Adaptation du *Visualization pipeline* de dos Santos et Brodlie [37]. 2

1.2 Processus de mise en place d'un système de visualisation. Ce *pipeline* est étroitement lié au *visualization pipeline* adaptée de [37]. 5

1.3 (a) Sous-réseau du « graphe d'Hollywood » où les sommets sont des acteurs et une arête indique que les acteurs correspondants aux extrémités de l'arête ont joué dans un même film ; (b), (c) et (d) Les décompositions successives calculées par notre algorithme où chaque groupe est entouré par une enveloppe convexe. 8

1.4 Sous-réseau du réseau d'interactions de la mouche. Chaque groupe et sous-groupe est entouré par une enveloppe convexe. 8

1.5 Réseau métabolique de Escherichia Coli dessiné en utilisant notre approche. En haut à droite, agrandissement de la voie métabolique *superpathway of histidine, purine and pyrimidine biosynthesis*, cette voie contient un cycle de 17 réactions. 9

1.6 Temps moyen de réponse pour les trois algorithmes de dessin, tous réseaux et tous motifs confondus. Les seules différences statistiquement significatives sont celles entre l'algorithme hiérarchique et GEM, et entre l'algorithme hiérarchique et MetaViz. 10

2.1 Exemple de décomposition de graphe $\{G, H\}$. La figure (a) montre le graphe G et la figure (b) le DAG de décomposition H. Le sommet bleu de H représente l'ensemble des sommets du graphe G (feuilles dans H) que l'on peut atteindre dans le DAG depuis le sommet bleu. 17

2.2 Partition d'un graphe en deux groupes (en rose dans le dessin). Les arêtes vertes sont les arêtes intra-groupes tandis que les arêtes rouges sont les arêtes inter-groupes. 17

2.3 Les segments verts représentent le diagramme de Voronoï des 7 points du plan (en bleu, vert foncé, jaune, rose, marron, gris et rouge). Les cellules de chaque site sont indiquées par le code de couleurs. 21

3.1 La métrique Strength de l'arête $e = (u, v)$ est calculée en comparant le nombre de cycles de tailles 3 et 4 passant par cette arête au nombre maximal de tels cycles. 27

3.2 Un exemple de graphe planaire triangulé dessiné par l'algorithme de De Fraysseix *et al.* [51] (a) et par l'algorithme de Gutwenger et Mutzel (b). 30

3.3 Résultats des techniques d'augmentation utilisées par Sugiyama *et al.* [123] (a) et Auber [12] (b). Les sommets bleus sont les sommets fictifs ajoutés lors de cette étape. 32

3.4 (a) Hiérarchie construite par la méthode de Auber [12] ; (b) graphe de compaction correspondant [48]. 32

3.5 Illustration du croisement *inter-couches* de Kuntz *et al.* [93]. (a) et (b) sont les deux individus parents ; (c) et (d) sont les résultats du croisement inter-couches. L'individu (c) (resp. (d)) est constitué des couches entourées en bleu (resp. entourées en vert) de (a) et des couches entourées en rouge (resp. entourées en jaune) de (b). 33

3.6 Illustration du croisement *intra-couche* de Kuntz *et al.* [93]. (a) et (b) sont les deux individus parents ; (c) et (d) sont les résultats du croisement intra-couche. L'individu (c) est constitué des sommets de (b) entourés en rouge et des sommets de (a) non-entourés en rouge dans (b) (en jaune dans (a)). Inversement, l'individu (d) est constitué des sommets de (a) entourés en bleu et des sommets de (b) non-entourés en bleu dans (a) (en vert dans (b)). . . . 34

3.7 Sous-graphe du « graphe d'Hollywood » où les sommets représentent des acteurs et une arête relie deux sommets dont les acteurs correspondants ont joué dans (au moins) un film commun, (a) dessiné par l'algorithme de Fruchterman et Reingold [55] et (b) par l'algorithme de Frick *et al.* [54]. 35

3.8 Sous-graphe du « graphe d'Hollywood », (a) dessiné par l'algorithme GRIP [59, 58], (b) par l'algorithme de FM^3 [65] et (c) par LGL [5]. 38

3.9 Exemple de graphe sans-échelle contenant 2000 sommets et 9646 arêtes, (a) dessiné par l'algorithme HDE [68, 69] et (b) par l'algorithme de ACE [89]. . . 41

3.10 Capture d'écran du système de visualisation présenté par Abello *et al.* dans [4]. 43

3.11 *Représentation multi-niveaux* de Eades et Feng [43] : chaque plan représente un niveau de la hiérarchie. 45

3.12 Evolution de $S(t)$ en fonction t. La partie transitoire entre t' et t'' est donnée à titre d'exemple. 46

3.13 Capture d'écran de l'application présentée dans [3]. Sur la gauche, la vue du graphe quotient et sur la droite la vue sur l'arbre hiérarchie. 46

3.14 (a) Capture d'écran du système de visualisation présenté par Granitzer *et al.* dans [62] ; (b) Visualisation de données financières par la méthode de Kumar et Garland [92]. 47

3.15 Les flèches rouges et bleues représentent respectivement les forces d'attraction et de répulsion exercées sur le sommet u ; (a) dans le cas où u n'est pas contraint dans sa cellule (en vert) ; (b) dans le cas où u est contraint dans sa cellule. On remarque que ces forces n'ont ni la même intensité, ni le même sens, ni la même direction. 48

3.16 Interface du système de visualisation de graphe partitionné, Grouse [9]. La partie gauche montre l'arbre de partition et la vue de droite est une vue du graphe quotient dont certains métanoeuds ont été « ouverts ». 49

3.17 Algorithmes hiérarchiques ; (a) Dans l'algorithme de Sugiyama *et al.* [123], chaque sommet est sur une unique couche, e.g. le sommet u ; (b) dans la version de Schreiber, le sommet u est sur 5 couches. 50

3.18 Un nouveau problème de croisement d'arêtes ; (a) et (b) montrent ce graphe sous deux angles différents. Les croisements d'arêtes de strates différentes peuvent être « supprimés » par simple rotation du dessin. 51

3.19 Réseau métabolique de Escherichia Coli ; (a) dessiné par l'algorithme par modèle de forces de Cytoscape [118] et (b) par celui de SBMLviewer [1]. 52

3.20 Capture d'écran du logiciel Reactome [77]. Cette carte représente le réseau métabolique de Homo Sapiens. Les zones en rouge et en bleu du dessin représentent les cycles de vie du virus HIV. 53

3.21 Réseau métabolique complet de *E. Coli* dessiné par Pathway Tools cellular overview diagram 53

4.1 Exemple de décomposition de graphe $\{G, H\}$. La figure (a) montre le graphe G et la figure (b) le DAG de décomposition H. Le sommet bleu de H représente l'ensemble des sommets du graphe G (feuilles dans H) que l'on peut atteindre dans le DAG depuis le sommet bleu. 56

4.2 (a) Sous-graphe du « graphe d'Hollywood » ; (b) et (c) Résultats d'algorithmes de partitionnement, chaque groupe est entouré par une enveloppe convexe, dans chacun de ces deux cas, l'une des deux cliques maximales a été « scindée » ; (d) Chacune des deux cliques maximales a été détectée. 57

4.3 (a) Sous-réseau de IMDb ; (b) Les composantes biconnexes sont entourées par des enveloppes convexes ; (c) Les composantes triconnexes sont entourées par des enveloppes convexes ; (d) Les composantes 4-connexes sont entourées par des enveloppes convexes. 59

4.4 (a) Un exemple de graphe triconnexe ; (b) Après la suppression du sommet u, le graphe résultant peut être décomposé en composantes triconnexes ; (c) Si on ajoute le sommet u à chaque composante de (b), on obtient alors trois composantes C_1, C_2 et C_3 et il n'existe pas de *3-séparateurs* (cf définition dans la section 2.2) contenant u dans C_1, C_2 ou C_3. 61

4.5 Nous nous intéressons sur cet exemple au groupe C_i. On observe trois types d'arêtes : les arêtes internes à C_i notées $e_{i,i}$ (en rouge), les arêtes dont une extrémité est dans $C_i \cap C_j$ et l'autre dans $C_j \setminus C_i$ notées $e_{i\cap j,j\setminus i}$ (en vert), et les arêtes dont une extrémité est dans $C_i \setminus C_j$ et l'autre dans $C_j \setminus C_i$ notées $e_{i\setminus j,j\setminus i}$ (en bleu). Seules les arêtes vertes et bleues doivent compter dans la cohésion externe du groupe C_i 63

4.6 (a) Composante connexe principale des données du concours InfoVis 2007 : ce graphe contient 117948 sommets et 1917841 arêtes ; Les autres graphes sont issus de la collection de sous-graphes extraits du graphe (a). On peut remarquer dans ces sous-graphes des parties ayant une haute densité, ce sont les structures de communauté. La couleur des sous-graphes est calculée en fonction de la valeur de MQ_{Over} de la décomposition en composantes 4-connexes du jaune pour les valeurs faibles au bleu pour les plus fortes (de 0.81 à 0.95). 65

4.7 Résultats qualitatifs de chacun des deux algorithmes agglomératifs Newman [104] et MQ_agglo ainsi que de la décomposition en composantes 4-connexes sur la collection de sous-graphes extraits des données du concours InfoVis 2007. Ce graphique montre les valeurs moyennes de Q, MQ_{Over} et de la densité moyenne des groupes ainsi que leurs écarts types respectifs. 66

4.8 Temps de calcul moyen en secondes ainsi que l'écart type de chacun des deux algorithmes agglomératifs Newman [104] et MQ_agglo ainsi que de la décomposition en composantes 4-connexes sur la collection de sous-graphes extraits des données du concours InfoVis 2007. 67

4.9 Résultats qualitatifs de chacun des deux algorithmes divisifs Newman et Girvan [105] et Auber *et al.* [13] ainsi que de la décomposition en composantes 4-connexes sur la collection de sous-graphes extraits des données du concours InfoVis 2007. Ce graphique montre les valeurs moyennes de Q, MQ_{Over} et de la densité moyenne des groupes ainsi que leurs écarts types respectifs. 68

4.10 Temps de calcul moyen en secondes ainsi que l'écart type de chacun des algorithmes divisifs de Newman et Girvan [105] et de Auber *et al.* [13] ainsi que de la décomposition en composantes 4-connexes sur la collection de sous-graphes extraits des données du concours InfoVis 2007. 69

4.11 Résultats de l'algorithme K-cores et de la décomposition en composantes 4-connexes sur la collection de sous-graphes extraits des données du concours InfoVis 2007. Ce graphique montre les valeurs moyennes de Q, MQ_{Over} et de la densité moyenne des groupes ainsi que leurs écarts types respectifs. 70

4.12 Temps de calcul moyen en secondes ainsi que l'écart type de l'algorithme K-cores et de la décomposition en composantes 4-connexes sur la collection de sous-graphes extraits des données du concours InfoVis 2007. 70

5.1 Une vue d'ensemble illustrant toutes les étapes de notre algorithme : du graphe arête-valué partitionné au dessin respectant les contraintes de la section 5.1 . 75

5.2 (a)Un exemple de graphe ; (b) Un exemple d'arbre de partition du graphe montré en (a), dans cet exemple la *hauteur* de l'arbre est 3 ; (c) Dessin du graphe quotient Q_1 correspondant au niveau 1 du graphe partitionné présenté en (a) et (b) en utilisant GRIP valué ; (d) Diagramme de Voronoï des positions des sommets du graphe Q_1 ; (e) Dessin du graphe quotient Q_2 ainsi que le diagramme correspondant à ce niveau ; (f) Résultat final. 76

5.3 (a) Un exemple de graphe contenant 70 sommets et 123 arêtes ; (b) Les sommets en rouge représentent l'ensemble maximal V_1 de sommets à distance au moins 2 ; (c) Les sommets en rouge représentent l'ensemble maximal V_2 de sommets à distance au moins $2^2 = 4$; (d) Les sommets en rouge représentent l'ensemble maximal V_3 de sommets à distance au moins $2^2 = 8$; (e) Les sommets en rouge représentent l'ensemble maximal V_4 de sommets à distance au moins $2^2 = 16$. 77

5.4 Positionnement initial d'un sommet u de V_i, $1 \leq i < k$. Les sommets v_1, v_2 et v_3 sont ses trois plus proches voisins déjà positionnés. Les points u_1^-, u_1^+, u_2^-, u_2^+, u_3^- et u_3^+ correspondent aux solutions des systèmes d'équations ci-dessous. 79

5.5 (a) En vert, le diagramme de Voronoï des positions des sommets du graphe Q_1 de la figure 5.2.(c) ; (b) Résultat après le dessin du sous-graphe correspondant au métanoeud 1 de la figure (a). 81

5.6 Illustration de l'algorithme de Fortune [50] sur un ensemble de trois points du plan (en rouge). (a) Les cônes d'influences des trois points du plan, en vert les intersections de ces cônes ; (b) Projection sur le plan (Oxy), les demi-droites en vert représentent le diagramme de Voronoï des trois points du plan. 81

5.7 (a) Graphe de la figure 5.3 pour lequel on considère maintenant les valuations des arêtes. Ces valuations sont toutes égales à 0.01 ; (b) Le *MISF*, $\mathcal{V}$ a été calculé sur le graphe en utilisant les distances valuées. L'ensemble V_1 de $\mathcal{V}$ ne contient qu'un seul sommet, en rouge dans le dessin. 82

5.8 Exemple de graphe arête-valué, la partie « gauche » du graphe (en jaune) ne contient que des arêtes faiblement valuées tandis que la partie « droite » (en bleu) ne contient que des arêtes fortement valuées. Pour obtenir un bon échantillonnage des sommets de ce graphe, il ne faudrait pas utiliser la même distance pour les deux parties du graphe. 82

5.9 (a) Exemple du graphe arête-valué de la figure 5.8 ; (b) Arbre couvrant calculé en utilisant notre approche ; (c), (d), (e) et (f) Les sommets en rouge indiquent respectivement les ensembles V_1, V_2, V_3 et V_4. 84

5.10 (a) Dessin du sous-graphe $Q_i[S]$ situé dans la cellule centrale à l'itération t, u_t indique la position du noeud u à l'itération t ; (b) Pour calculer la force totale exercée sur u à l'itération t, un nombre constant de voisins dans Q_i sont pris en compte. u'_{t+1} indique la position que devrait avoir u à l'itération $t+1$ si l'on ne forçait pas les sommets du graphe $Q_i[S]$ à rester dans sa cellule. La position de u_{t+1} est trouvée en calculant l'intersection entre la cellule et le segment $[u_t u'_{t+1}]$. 87

5.11 (a) En bleu, le diagramme de Voronoï du sous-graphe représenté par le métanoeud 1 dans la figure 5.2.(a) ; (b) Les régions des métanoeuds (ici, de simples sommets) du sous-graphe sont le résultat de l'intersection du diagramme de Voronoï et de la cellule du métanoeud 1. 88

5.12 (a) Positionnement final des sommets du graphe partitionné de la figure 5.2.(a) et (b) ; (b) Positions des sommets de l'arbre de partition de la figure 5.2.(b). Ces positions sont utilisées pour regrouper les arêtes en faisceaux. 89

5.13 Sous partie d'un réseau d'interactions gène-protéine dessiné par notre approche, avant le post-traitement (a) et après (b). 89

5.14 (a) Réseau d'interactions gène-protéine de la mouche dessiné par un algorithme par modèle de forces classique [54], l'effet « brouillon » de ce dessin provient de la nature sans-échelle de ce graphe ; (b) Un algorithme de partitionnement a été appliqué sur ce graphe, et chaque groupe est entouré par une enveloppe convexe ; (c) Résultat de notre technique sur ce graphe partitionné. 90

6.1 Exemple de décomposition de graphe : (a) et (b) la paire (G,H) comme défini dans la section 2.2. Dans (a) chaque groupe est entouré par une couleur différente. 94

6.2 Graphe de dépendance correspondant au graphe décomposé (G, H) de la figure 6.1. 94

6.3 Poster de Boehringer [102] représentant le réseau métabolique de l'*homo sapiens*. 95

6.4 Voies métaboliques de la Glycolyse et de la Gluconeogenèse comme elles sont données sur le site de KEGG [80]. 96

6.5 Représentation manuelle respectant les contraintes de représentation proposées dans [102] d'une sous-partie d'un réseau biologique correspondant au mécanisme de régulation de la biosynthèse du cholestérol. 97

6.6 Graphe biparti représentant deux réactions biochimiques. 98

6.7 Description d'une réaction dans un fichier SBML extrait de la base de donnée BioCyc [87] puis traitée afin d'intégrer à chaque réaction l'information d'appartenance à une ou plusieurs voies. 99

6.8 Exemple fictif de réseau métabolique contenant 5 voies métaboliques de couleurs distinctes (p_1, p_2, p_3, p_4 et p_5). 100

6.9 Résultat de la première étape de partitionnement. (a) reprend l'exemple de la figure 6.8 ; (b) résultat de la première étape sur le réseau de la figure (a). . . 102

6.10 . Supposons que la réaction $R1$ transforme le composé $C1$ en $C2$ et $C2'$ et que $R2$ transforme les composés $C2$ et $C2'$ en $C3$. Si les réactions $R1$ et $R2$ sont réversibles, alors une recherche « simple » de cycle le plus long trouverait le cycle $C2, R1, C2', R2$ bien que ce cycle ne soit pas biologiquement pertinent. 102

6.11 Résultat de la recherche de cycles dans P_{ind} et P'_{Nind}. (a) et (b) reprennent l'exemple de la figure 6.9 ; (c) les arêtes des cycles trouvés lors de cette étape apparaissent en bleu. 103

6.12 Résultats de la recherche de cycles et de cascades de réactions dans le graphe de la figure 6.11.(c). (a), (b) et (c) reprennent l'exemple de la figure 6.11 ; (d) les arêtes des cycles et cascades de réactions trouvés lors de cette étape apparaissent en bleu. 104

6.13 (a) Exemple de la figure 6.8 ; (b) Graphe de dépendance correspondant ; (c) Le sommet centre d'intérêt, ici le sommet correspondant à $p4$ (entouré en rouge dans la figure (b)), ainsi que ces voisins sont retirés du graphe de dépendance ; (d) L'ensemble indépendant de cet exemple est $P_{ind} = \{p4, p2\}$. 105

6.14 (a) Sous-graphe extrait du réseau de la figure 6.5 ; (b) Dessin obtenu pas l'algorithme Mixed-Model de Gutwenger et Mutzel [64] sur ce sous-graphe. . . . 106

6.15 Le graphe G_k d'un *Shelling order*. La partie rouge du dessin représente l'ensemble V_1 du *Shelling order*. Les parties bleue et verte représentent les deux cas possibles pour l'ensemble V_k. 108

6.16 Exemple de positionnement des in- et outpoints d'un sommet u ayant 9 inpoints et 9 outpoints. Les lignes en pointillé indique les directions des arêtes correspondantes. 109

6.17 (a) Positionnement des sommets de V_i et de V_{i+1} dans l'algorithme de [64] ; (b) Positionnement des sommets de V_i et de V_{i+1} lorsque ceux-ci ont des tailles variables. 109

6.18 Résultats de notre approche sur le graphe de la figure 6.8. (a), (b), (c) et (d) reprennent les figures 6.8, 6.9, 6.11 et 6.12 ; (e) Le résultat de notre algorithme de dessin sur le graphe de la figure (d). Chaque voie métabolique est entourée par une enveloppe de couleur. 110

6.19 Réseau métabolique complet de *E. Coli* dessiné par note approche. Les métanoeuds mauves représentent les voies métaboliques dessinées correctement. Les métanoeuds jaunes correspondent eux aux structures topologiques particulières (cycles et chaînes) détectées. 111

6.20 Temps de calcul en secondes de notre algorithme pour chacun des 27 organismes.112

6.21 Pourcentage de temps passé lors de chaque étape. 113

6.22 Réseau métabolique complet de *E. Coli* dessiné par Cytoscape 114

6.23 Réseau métabolique complet de *E. Coli* dessiné par Pathway Tools cellular overview diagram 114

6.24 La super-voie métabolique de « glycolysis, pyruvate dehydrogenase, TCA, and glyoxylate bypass ». (a) En utilisant notre méthode. Les sommets et les liens correspondants au cycle de Krebs sont entourés en rose ; (b) dans BioCyc [85] 116

6.25 Dessin de la voie de la biosynthèse de la valine en utilisant notre méthode et sans paramétrer l'algorithme de partitionnement. Les sommets correspondants sont entourés en rose. On peut voir qu'ils sont partagés sur trois métanoeuds. 117

6.26 Dessin de la voie de la biosynthèse de la valine en utilisant notre méthode et en guidant l'algorithme de partitionnement. 118

6.27 Dessin de la voie de la biosynthèse de la valine dans BioCyc. 119

6.28 Les sommets directement liés à la voie de la synthèse de la valine sont entourés en rose. 119

6.29 Liens entre la voie de la biosynthèse de la valine dans Pathway Tools cellular overview diagram. 120

7.1 Pour que GEM [54] prenne en compte la taille des sommets, la taille idéale de l'arête $e = \{u, v\}$ est la somme de *len* et des rayons des cercles englobant de u et de v. 123

7.2 Résultat de l'algorithme par modèle de forces sur le graphe quotient de Buchnera Aphidicola BP. 123

7.3 Résultat de l'algorithme hiérarchique sur le graphe quotient de Buchnera Aphidicola BP. 124

7.4 Résultat de l'algorithme MetaViz sur le graphe quotient de Buchnera Aphidicola BP. 125

7.5 (a) Exemple d'une occurrence du motif {1.2.1.12, 5.3.1.1, 4.1.2.13} ; (b) Ces trois ensembles représentent le même motif. Notez qu'un motif n'est pas nécessairement un chemin. 126

7.6 Vue agrandie du réseau métabolique de *Buchnera Aphidicola APS* dessiné par (a) l'algorithme GEM, (b) l'algorithme hiérarchique et (c) l'algorithme MetaViz. Les chemins reliant le sommet entouré en rouge et les sommets à distance inférieure ou égale à deux du sommet entouré en rouge sont en surbrillance rose. 128

7.7 Temps de réponse des participants en fonction des tâches précédemment effectuées. La courbe rouge est une régression logarithmique du temps moyen de réponse. Au delà de la douzième tâche, les participants ont acquis assez d'expérience pour commencer l'expérimentation « réelle ». 130

7.8 Capture d'écran du logiciel d'évaluation. Les boutons 1 et 2 permettent de sélectionner et de trouver les réactions pertinentes. Le bouton 2 est un interacteur permettant de mettre en surbrillance le voisinage à distance 2 d'un sommet, et le bouton 1 de sélectionner les réactions pertinentes. Le bouton 3 permet quant à lui de supprimer toute surbrillance. Le bouton 4 permet de valider la réponse et d'obtenir la tâche suivante. Enfin, 5 montre la zone où le retour sur les réponses précédentes est donné lors des cinq premières tâches d'apprentissage. 131

7.9 En utilisant l'interacteur du bouton 2, « cliquer » sur la réaction étiquetée 6.3.4.3 met en surbrillance les sommets à distance au plus 2 dans le graphe original ainsi que les arêtes (et méta-arêtes) les reliant. Ici, la réaction 6.3.4.3 appartient bien au motif recherché ($\{6.3.4.*, 3.5.4.9, 6.3.2.17\}$) puisque les autres réactions du motif (3.5.4.9 et 6.3.2.17) ont été mises en surbrillance rose. . . 132

7.10 Temps moyen de réponse pour les trois algorithmes de dessin, tout réseau et tout motif confondus. 133

7.11 Réponses des participants aux questions Q1, Q3 et Q5 (algorithmes préférés) données en pourcentages. 135

7.12 Réponses des participants aux questions Q2, Q4 et Q6 (algorithmes les moins appréciés) données en pourcentages. 136

Liste des tableaux

3.1 Complexités en temps et en espace des algorithmes de [42, 55, 54, 79]. 37

3.2 Complexités en temps et en espace des algorithmes de [59, 58, 65, 5]. 40

4.1 Avantages et Inconvénients des six algorithmes comparés dans ce chapitre. . 71

7.1 Nombre d'occurrences de chaque motif contenu dans chaque réseau métabolique (graphes A, B et C) et indique combien sont contenues dans un métanoeud (contenues) et combien sont partagées sur plusieurs métanoeuds (partagées). 127

7.2 Différences trouvées lors du test Tukey entre chaque paire de conditions. Dans cette étude la valeur attendue est $HSD(3.44, \alpha = 0.05) = 4.77$. 134

7.3 Nombre et résumé des commentaires positifs et négatifs sur les trois algorithmes de dessin. 136

Chapitre 1
Introduction

1.1 Motivations

Il existe de nos jours de nombreux systèmes d'information, tels que les média écrits, télévisuels, ou encore internet. Parallèlement au développement de ces systèmes d'information, les techniques d'acquisition de données permettent de générer toujours plus de données. La taille et la complexité de ces données soulèvent un nouveau problème : les outils classiques d'analyse de données (e.g. graphiques, tableurs, etc) permettent-ils de les explorer efficacement ? Il paraît évident que l'analyse d'une liste de plusieurs milliers (voire millions) de lignes requiert un effort non négligeable aux scientifiques. La constante augmentation de la quantité d'information stockée dans les bases de données mais aussi la complexité de ces données rendent donc cruciale la mise en place de système d'exploration et d'analyse permettant leur exploitation.

Dans [130], Ware explique que 40% de nos activités corticales sont dédiées à la transmission et à l'analyse de signaux visuels. Utiliser un support visuel est donc intéressant pour permettre aux experts d'explorer leurs données. C'est dans ce contexte que s'inscrit la Visualisation d'Information. Selon Ware [130], un système de Visualisation d'Information doit permettre l'exploration visuelle de données complexes et/ou de grandes tailles et ainsi permettre leurs analyses. La Visualisation d'Information tente donc d'exploiter au mieux l'acuité visuelle de l'utilisateur afin de lui permettre d'analyser ses données.

Dans [37], dos Santos et Brodlie présentent leur *visualization pipeline* (cf figure 1.1). Ce *pipeline* met en exergue les différentes composantes intervenant généralement dans un processus de visualisation d'information :

1. Analyse des données
2. Filtrage
3. Plongement Visuel (en anglais, *Visual Mapping*)
4. Rendu (en anglais, *Rendering*)

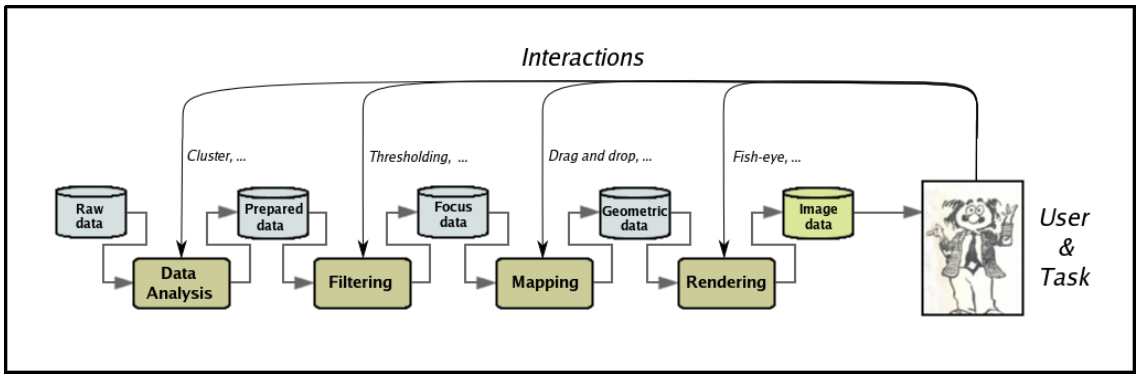

FIG. 1.1: Adaptation du *Visualization pipeline* de dos Santos et Brodlie [37].

La Visualisation d'Information est donc généralement modélisée par ces quatre étapes. La première étape consiste tout d'abord à construire une abstraction des données. Nous nous intéressons tout particulièrement dans cette thèse aux données relationnelles. Il existe de nombreux moyens pour visualiser et/ou représenter de telles données comme les matrices, les coordonnées parallèles ou encore les graphes. Nous avons choisi dans cette thèse de nous concentrer sur la visualisation de graphe. Dans ce cadre, cette étape consiste ici à construire un graphe correspondant à ces données.Plus précisément, cela correspond à choisir quels éléments sont représentés par des sommets dans ce graphe et quelle relation entre les éléments une arête du graphe représente. Puis dans le cas de données complexes, le graphe peut être décomposé en groupes d'éléments similaires. Cette étape de décomposition est très importante pour deux raisons. D'une part, elle permet une première analyse (automatique) des données. Cette analyse devra ensuite être validée et/ou affinée par l'expert via un système d'interactions. D'autre part, elle permet de construire une abstraction sans laquelle la représentation pourrait être confuse.

Lors de la deuxième étape du *pipeline*, les données produites lors de la première étape sont filtrées. Cette simplification consiste à retirer des éléments (sommets et/ou arêtes) dont la valeur d'un attribut est en deçà d'une borne fixée par l'expert. Cela permet ainsi de n'afficher que les éléments les plus « importants » et d'éliminer dans le cas de données expérimentales d'éventuels « bruits ».

La troisième étape du *pipeline* consiste à construire des données géométriques à partir de l'objet mathématique qu'est le graphe. Pour fabriquer ces données géométriques, deux composantes interviennent : le dessin de graphe et la mise en évidence de certains attributs. Le dessin de graphe consiste à attribuer des coordonnées aux sommets et arêtes du graphe. Le positionnement des sommets et arêtes doit respecter un certain nombre de contraintes esthétiques [16], par exemple minimiser le nombre de croisements d'arêtes, maximiser la résolution angulaire ou encore minimiser l'espace occupé par le dessin. Un algorithme

efficace de dessin de graphe doit prendre non seulement en compte les conventions propres aux domaines d'application (e.g. biologie, chimie, etc) mais aussi mettre visuellement en évidence l'éventuelle décomposition du graphe. Parallèlement au calcul des coordonnées, la couleur ou la taille permettent de mettre en évidence la valeur d'un attribut des sommets et/ou des arêtes.

Enfin, lors de la dernière étape, l'image est construite à partir des données géométriques par des techniques de *rendu* (en anglais, *Rendering*). Cette partie de la Visualisation d'Information fait intervenir des techniques proches de l'infographie. Par exemple, il peut s'agir de construire une image par un nuage de points (e.g. [111, 110]), ou encore de la mise en évidence de sous-parties du réseau par des enveloppes (e.g. [27]). Nous ne nous intéressons pas à l'étape de « Rendu » dans cette thèse. En effet, les travaux présentés dans ce manuscrit ont été réalisés sur la plateforme de visualisation de graphe Tulip [12]. La plateforme Tulip permet de dessiner et d'interagir sur de grands graphes (jusqu'à 1000000 de sommets) mais aussi et surtout d'afficher ces graphes. La partie *rendu* est donc réalisée par ce logiciel.

1.2 Extraction, Décomposition et Filtrage

Parmi ces trois étapes, nous nous sommes tout particulièrement intéressés, lors de cette thèse, à la décomposition de graphe. En effet, la décomposition de graphe en groupes d'éléments « similaires » est l'un des prétraitements les plus intéressants puisqu'elle offre deux atouts majeurs. Premièrement, le calcul automatique de ces sous-structures aide l'expert dans l'analyse du réseau. La similarité des éléments d'un même groupe permet à l'expert de réduire le nombre d'éléments qu'il doit examiner puisqu'il peut alors limiter son étude à certains groupes d'intérêt. Deuxièmement, cette décomposition en groupes permet de construire une abstraction des données et ainsi facilite la compréhension de l'organisation générale de celles-ci.

Comme mentionné ci-dessus, les algorithmes automatiques de décomposition de graphe recherchent des groupes d'éléments ayant une (ou plusieurs) propriété(s) commune(s). Le critère le plus largement admis pour qu'un ensemble de groupes forme une « bonne » décomposition du graphe est que la cohésion interne de chaque groupe soit aussi élevée que possible tout en ayant une cohésion faible entre ces groupes. Afin d'évaluer la qualité d'une décomposition, il existe de nombreuses mesures de qualité. Chacune permet de mesurer la qualité d'une décomposition selon certains critères et par conséquent ces mesures ne permettent pas d'évaluer les mêmes aspects d'une décomposition. Maximiser l'une de ces mesures ne semble pas suffisant pour trouver une « bonne » décomposition de graphe. Un

« bon » algorithme de décomposition doit essayer de maximiser simultanément plusieurs de ces mesures de qualité.

1.3 Plongement Visuel et Rendu

La deuxième partie du *visualization pipeline* consiste à afficher une image représentant les données. L'étape « Plongement Visuel » permet de construire des données géométriques à partir du graphe, puis l'étape de « Rendu » construit l'image à partir de ces données géométriques. Comme mentionné ci-dessus, nous ne nous intéressons pas dans cette thèse à l'aspect « Rendu » du *visualization pipeline*.

La composante principale du « Plongement Visuel » est le dessin de graphe. Le dessin de graphe consiste à positionner les sommets et les arêtes sur un plan (dessin en 2 dimensions) ou dans une espace à d dimensions (généralement $d = 3$). Bien que la définition du dessin de graphe soit relativement simple, le dessin de graphe relève généralement de la théorie des graphes [18] et de l'algorithmique géométrique [34]. Pour que l'expert puisse en extraire de l'information, le dessin du graphe doit prendre en compte le domaine d'application. En effet, chaque domaine à ses propres conventions et par conséquent ses propres contraintes. En particulier dans le domaine de la biologie, il faut prendre en compte les habitudes des biologistes. La biologie est un domaine dans lequel les représentations visuelles ont été largement utilisées. Au cours des années, les représentations manuelles des processus biologiques ont permis de mettre en place des conventions plus ou moins précises de dessin [102]. Une « bonne » visualisation doit respecter au mieux ces nombreuses conventions afin d'aider les biologistes à exploiter aisément leurs données.

1.4 Utilisation par l'expert

Une fois toutes les étapes du *visualization pipeline* réalisées, une image est présentée à l'expert. La visualisation offerte à l'expert doit non seulement lui permettre de vérifier ses connaissances *a priori*. Mais surtout, cette image doit lui permettre de trouver de nouvelles informations et donc de faire de nouvelles hypothèses. Afin de vérifier cette (ces) nouvelle(s) hypothèse(s), l'expert peut alors utiliser un système d'interactions. Grâce à celui-ci, l'expert peut modifier chacune des étapes du *visualization pipeline*. Par exemple, les techniques de *fisheye* modifient l'étape de « Rendu », ou encore, le *drag and drop* modifie l'étape de « Plongement Visuel ». En d'autres termes, le système d'interactions lui permet d'émettre de nouvelles spécifications engendrant ainsi une nouvelle visualisation. Ce processus cyclique se répète tant que l'expert ne peut confirmer ou infirmer son hypothèse.

1.5 Mise en oeuvre d'un système de visualisation d'information

Le domaine d'application principal de cette thèse est la biologie. Dans ce domaine, les techniques d'acquisition génèrent un très grand nombre de données. D'une part ces données sont complexes et variées (e.g. séquences d'ADN et/ou d'ARN, réseaux de protéines, réseaux métaboliques, etc). D'autre part elles sont généralement relationnelles et par conséquent sont représentables par des graphes. Nous nous sommes intéressés lors de cette thèse à la visualisation orienté tâche, c'est-à-dire à la mise en place de systèmes de visualisation dédiés à la résolution d'une ou plusieurs tâches particulières.

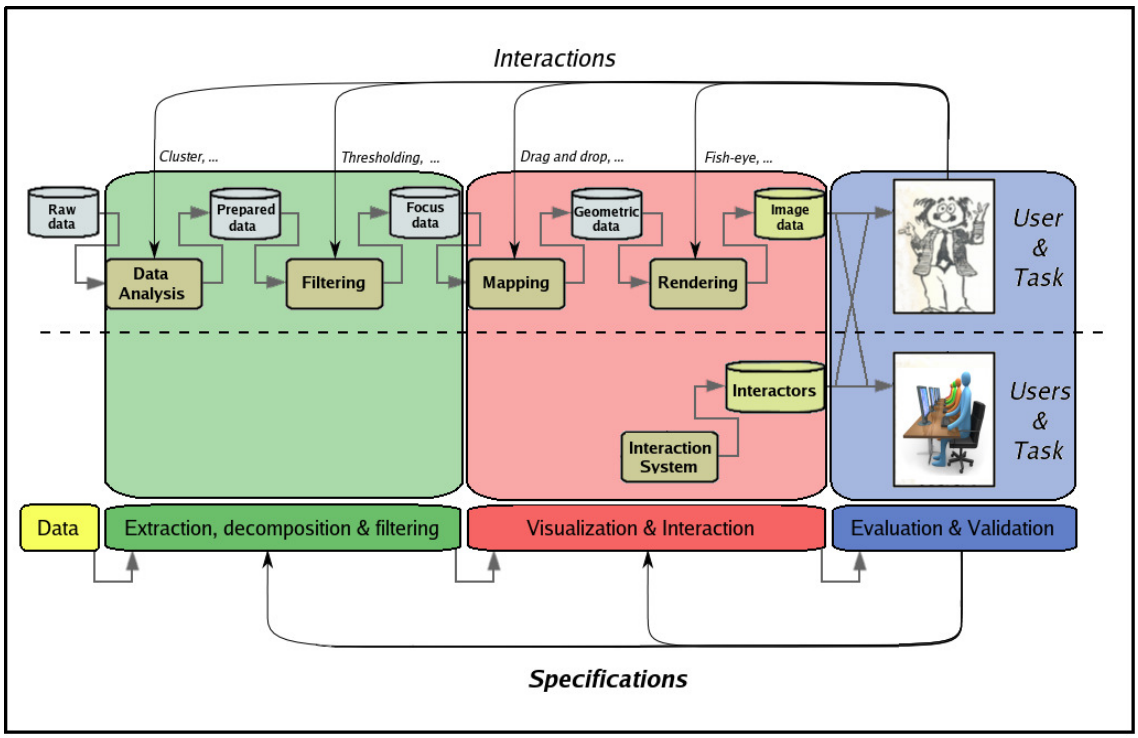

FIG. 1.2: Processus de mise en place d'un système de visualisation. Ce *pipeline* est étroitement lié au *visualization pipeline* adaptée de [37].

Grâce aux différentes étapes du *visualization pipeline*, la Visualisation d'Information doit permettre à un expert de résoudre ses tâches. Afin de respecter au mieux les attentes des experts, il semble donc intéressant de procéder de manière analogue lors de la mise en oeuvre d'un système de visualisation. La figure 1.2 illustre le processus de mise en place que nous avons utilisé lors de cette thèse. Ce processus comporte 3 étapes principales :

1. Extraction, Décomposition, Filtrage
2. Visualisation et Interaction
3. Validation et Evaluation

La première étape « Extraction, Décomposition, Filtrage » du processus correspond aux deux premières phases du *visualization pipeline*. L'étape « Visualisation » quant à elle, correspond aux phases « Plongement Visuel » et « Rendu » de ce *pipeline*. De plus, cette étape comporte la mise au point d'un système d'interactions permettant à l'expert de naviguer dans ces données. Ce système doit permettre de respecter le mantra de Shneiderman [119] :

« Overview first, zoom and filter, then details -on demand- »

Partant d'une vue globale des données, l'utilisateur doit pouvoir interagir sur la vue par des techniques d'agrandissement, éventuellement filtrer les éléments pour simplifier la vue et enfin demander des informations précises sur une sous-partie d'intérêt du graphe. Ces techniques d'interactions font partie des plus connues, il en existe cependant de nombreuses autres [109, 70], des plus simples aux plus évoluées. Parmi les plus simples se trouvent les translations de vue et les modifications de facteur d'agrandissement (*pan and zoom*) mais aussi les déplacements d'éléments du graphe (*drag and drop*). Et parmi les plus évoluées, on trouve une technique d'interaction permettant de calculer automatiquement une trajectoire lors d'un changement de vue [127] par une combinaison de *pan* et de *zoom*, le *fisheye* [56, 60], ou encore, dans le cas de graphes décomposés, la navigation dans la hiérarchie de groupes [14, 4, 9, 11]. L'utilisation de ces interacteurs permet de « revenir » à une étape précédente du *visualization pipeline* et ainsi de générer une nouvelle image correspondant à la requête émise. Lors de la mise au point des interacteurs, les choix faits sont donc étroitement liés à l'algorithme de dessin et à la technique de rendu utilisés.

Enfin, l'étape « Validation et Evaluation » correspond à l'évaluation de la qualité du système par des utilisateurs. On peut distinguer deux types d'évaluation : l'évaluation par un (ou plusieurs) expert(s) du système de visualisation lors de sa mise en place ; et l'évaluation expérimentale *a posteriori* par un ensemble d'utilisateurs (non nécessairement experts du domaine d'application) pour valider l'approche utilisée. Faire intervenir l'utilisateur, ici l'expert, dans le processus de mise en place du système de visualisation permet d'améliorer la visualisation et/ou l'interaction. En effet, lors de l'utilisation, l'expert met en évidence un certains nombre de nouvelles spécifications permettant d'améliorer la qualité du système. Ces améliorations peuvent porter sur chacune des étapes d'« Extraction, Décomposition, Filtrage » et de « Visualisation », créant ainsi un cycle dans le processus. Ces cycles sont nécessaires à la mise en place d'un système de visualisation efficace, cependant ils ne nous permettent d'obtenir qu'un avis purement subjectif (d'un ou plusieurs experts) sur la qualité du système. Pour vérifier que la visualisation permet effectivement de résoudre une tâche efficacement, il faut alors mettre en place une évaluation expérimentale. Lors de cette évaluation, la qualité du système est mesurée par les performances

d'un échantillon de personnes sur une tâche donnée. Evaluer les performances d'utilisateurs (non nécessairement experts) sur le système de visualisation permet de connaître l'efficacité réelle du système. La qualité du système est alors mesurée par deux facteurs : le taux d'erreurs et le temps pris pour réaliser la tâche. Lors de ce type d'évaluation, plusieurs systèmes de visualisation sont généralement comparés et les résultats de chaque système sont alors analysés par une étude statistique. Cela permet de valider avec un taux de confiance donné les qualités de chacune des visualisations offertes.

1.6 Organisation du mémoire

Nous présentons dans cette section l'organisation de ce mémoire, et pour chacun des chapitres 4, 5, 6 et 7, nous donnons les méthodes utilisées ainsi que les contributions majeures de ces méthodes.

Dans le chapitre 2, nous donnons tout d'abord les définitions et les notations utilisées tout au long de ce manuscrit. Certaines notions spécifiques aux domaines d'applications et peu utilisées seront toutefois définies dans les chapitres suivants, lors de leur utilisation.

Le chapitre 3 porte sur l'*état de l'art* des domaines abordés dans cette thèse, c'est-à-dire, la décomposition, la visualisation de graphe et la visualisation en biologie et en particulier la visualisation de métabolismes.

Le chapitre 4 s'intéresse à la décomposition de graphe et plus particulièrement à une décomposition en groupes chevauchants. Il existe de nombreuses applications de la décomposition de graphe en biologie comme la détection automatique de familles de protéines ou encore la décomposition en voies métaboliques d'un réseau métabolique. La décomposition étudiée dans ce chapitre est basée sur la décomposition en composantes 4-connexes et est réalisée en temps $O(|V| \cdot |E|)$ (cf figure 4.1). Afin de valider cette approche, nous comparons dans ce chapitre le temps de calcul et la qualité des résultats obtenus à celle des résultats obtenus par des algorithmes connus et utilisés en visualisation d'information. Pour ce faire, nous utilisons trois mesures « standards » de qualité : la densité moyenne des groupes, la modularité [105] et la mesure MQ [99]. L'une des contributions majeures de ce chapitre est la généralisation de la mesure de qualité MQ au cas des décompositions en groupes chevauchants. De plus, nous montrons que notre décomposition permet d'obtenir de bonnes valeurs pour chacune des trois mesures contrairement aux autres approches évaluées. D'autre part, les temps de calcul offerts par notre décomposition sont tout à fait acceptables bien que sa complexité théorique soit élevée.

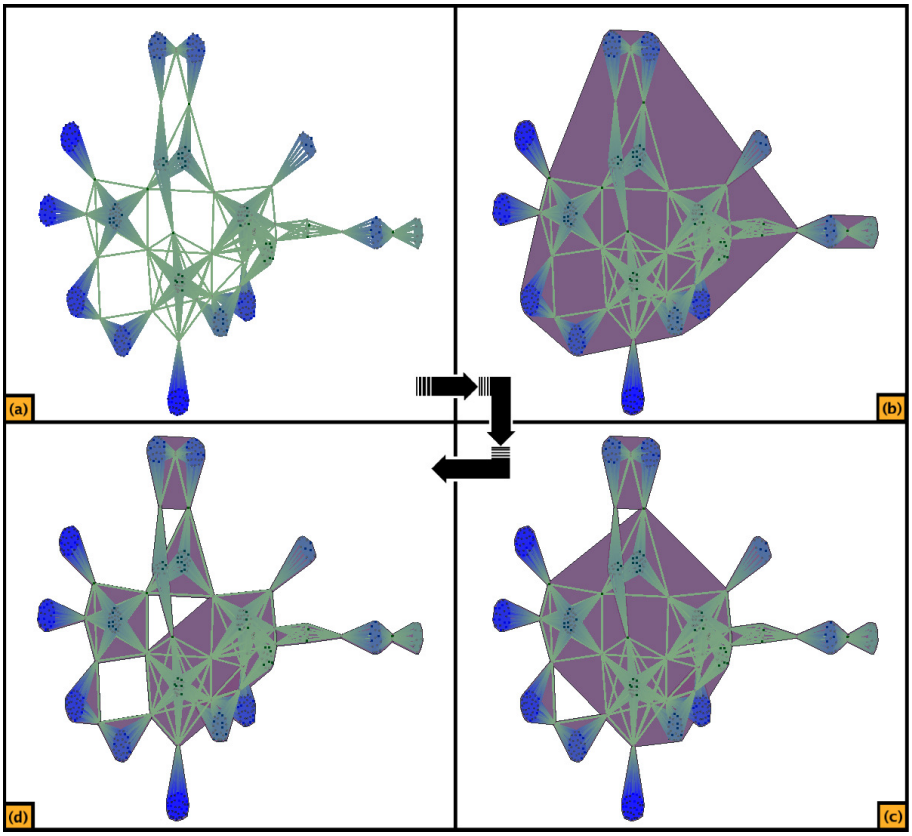

FIG. 1.3: (a) Sous-réseau du « graphe d'Hollywood » où les sommets sont des acteurs et une arête indique que les acteurs correspondants aux extrémités de l'arête ont joué dans un même film ; (b), (c) et (d) Les décompositions successives calculées par notre algorithme où chaque groupe est entouré par une enveloppe convexe.

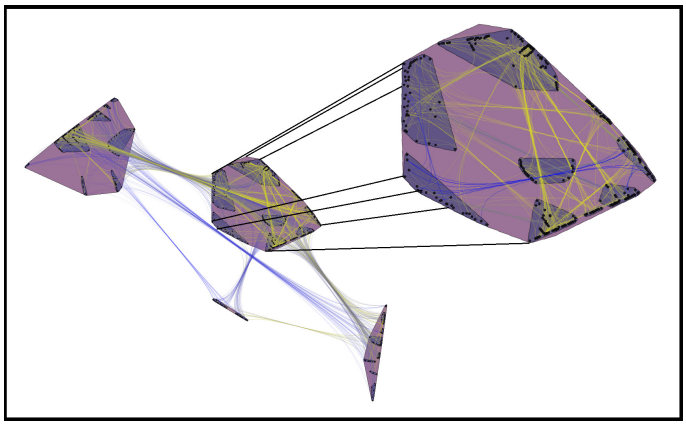

FIG. 1.4: Sous-réseau du réseau d'interactions de la mouche. Chaque groupe et sous-groupe est entouré par une enveloppe convexe.

Dans le chapitre 5, nous abordons le problème de la visualisation d'une partition[1] de graphe arête-valué. Plus précisément, il s'agit dans ce travail de représenter des réseaux d'interactions protéine-protéine ou gène-protéine décomposés dont les arêtes sont valuées. Cette visualisation doit prendre en compte un certain nombre de contraintes :

1. interdire le chevauchement de groupes
2. préserver les inclusions des groupes dans le dessin
3. obtenir un dessin dans lequel chaque groupe peut être entouré par une enveloppe convexe, et
4. respecter les distances valuées dans le graphe en utilisant une fonction de minimisation d'énergie.

Les contraintes 1 et 3 permettent une identification plus aisée des groupes de la partition puisqu'il est notamment plus facile de suivre le contour d'une enveloppe convexe que d'une enveloppe quelconque. De plus, l'inclusion visuelle des sous-groupes dans les groupes qui les contiennent nous permet de visualiser la hiérarchie produite par l'algorithme de partitionnement. Enfin, une fonction de minimisation d'énergie nous permet de rendre visuellement compte de la valuation des arêtes. Pour respecter ces contraintes, nous avons mis au point une approche descendante (i.e. nous dessinons chaque niveau de la hiérarchie du plus haut au plus bas) utilisant un algorithme par modèle de forces, GRIP [59, 58] et les diagrammes de Voronoï. Cette approche permet de respecter au mieux les contraintes 1, 2 et 3. Pour respecter la contrainte 4, nous avons adapté l'algorithme GRIP afin qu'il puisse prendre en compte la pondération des arêtes, et rendre ainsi visuellement compte de la force des interactions entre les protéines (cf figure 1.4).

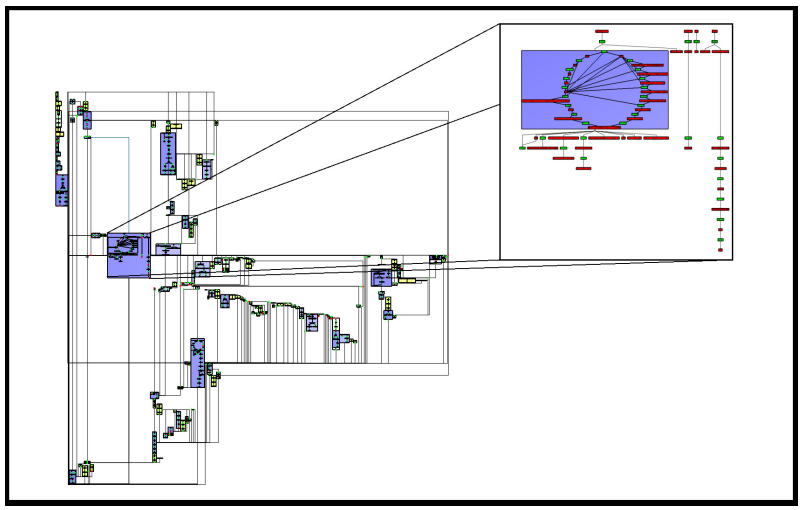

FIG. 1.5: Réseau métabolique de Escherichia Coli dessiné en utilisant notre approche. En haut à droite, agrandissement de la voie métabolique *superpathway of histidine, purine and pyrimidine biosynthesis*, cette voie contient un cycle de 17 réactions.

Le chapitre 6 concerne la visualisation de réseaux métaboliques (i.e. réseau de l'ensemble des réactions biochimiques ayant lieu dans un organisme). Ce type de données offre

[1] Une partition est un cas partitculier de décomposition où les groupes de la décomposition ne partagent aucun sommet.

un problème intéressant puisque la visualisation produite doit permettre de visualiser les voies métaboliques (i.e. sous-parties du réseau ayant des fonctionnalités connues) tout en conservant la vue globale du réseau. Les outils existant de visualisation du métabolisme portent soit sur la visualisation de voies métaboliques (e.g. [17, 115, 131]), soit sur la visualisation de réseaux complets mais autorisent la duplication de sommets (e.g. [77, 108]). La contribution majeure de notre travail est donc la représentation de réseaux métaboliques complets tout en interdisant la duplication de sommets (cf figure 1.5). D'autre part, notre algorithme de partitionnement permet de conserver au mieux l'information liée à la décomposition en voies métaboliques.

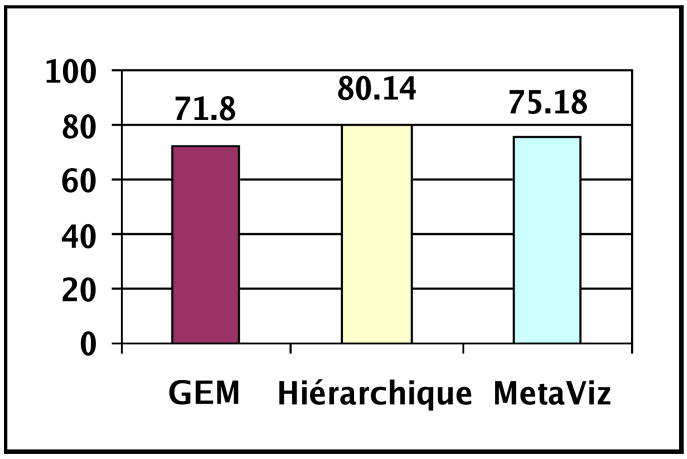

FIG. 1.6: Temps moyen de réponse pour les trois algorithmes de dessin, tous réseaux et tous motifs confondus. Les seules différences statistiquement significatives sont celles entre l'algorithme hiérarchique et GEM, et entre l'algorithme hiérarchique et MetaViz.

Dans le chapitre 7, nous évaluons la qualité de l'algorithme de dessin utilisé dans [19, 22] (cf chapitre 6). Pour cela, nous le comparons à deux algorithmes : un algorithme par modèle de forces, GEM [54] et un algorithme hiérarchique [12]. Nous avons enregistré le taux d'erreurs ainsi que le temps nécessaire à chaque participant de l'évaluation pour résoudre une tâche biologique liée à la connectivité. La figure 1.6 montre les résultats moyens obtenus par les utilisateurs pour chaque algorithme de dessin. Bien qu'il semble que les résultats de l'algorithme de dessin GEM soient meilleurs que ceux de MetaViz (i.e. algorithme utilisé dans le chapitre 6), cette différence n'est pas significative du point de vue statistique. Il semble donc que respecter les conventions biologiques permette d'obtenir une visualisation efficace même si la tâche devrait favoriser un algorithme par modèle de forces.

Enfin, nous concluons sur les travaux effectués au cours de cette thèse ainsi que les perspectives de recherche qu'ils offrent dans le chapitre 8.

Chapitre 2
Définitions et Notations

Nous présentons dans ce chapitre les principales notions et notations nécessaires à la compréhension de ce manuscrit. Nous donnons tout d'abord quelques notions ensemblistes dans la section 2.1, puis nous définissons certaines notions portant sur les graphes dans la section 2.2. Dans 2.3, nous donnons les définitions des réseaux biologiques sur lesquels portent certains de nos travaux. Enfin, dans la section 2.4, nous présentons certaines autres définitions importantes dans le cadre de cette thèse.

2.1 Ensembles

Définition 2.1 (Décomposition d'ensemble) *Soient E un ensemble et $P = \{E_i\}_{1 \leq i \leq p}$ un ensemble d'ensembles E_i tels que $\forall i \in [1..p], E_i \subseteq E$. P est une décomposition de E si et seulement si $\bigcup_{i=1}^{p} E_i = E$.*

Définition 2.2 (Partition d'ensemble) *Soient E un ensemble et $P = \{E_1, E_2, ..., E_p\}$ une décomposition de E. P est une partition de E si et seulement si $\forall i \in [1..p], j \in [1..p], i \neq j$, on a $E_i \cap E_j = \emptyset$.*

Définition 2.3 (Paire d'éléments) *Soit E un ensemble. On appelle paire d'éléments de E tout ensemble non-ordonné de deux éléments $\{u, v\}$ tel que $\{u, v\} \subseteq E$.*

Définition 2.4 (Couple d'éléments) *Soient E un ensemble et u et v deux éléments de E. On appelle couple des éléments u et v, la donnée de u et v dans un ordre déterminé, noté (u, v).*

2.2 Graphes

2.2.1 Définitions

Définition 2.5 (Graphe non-orienté) *Soient V et E deux ensembles, tels que $E \subseteq \{\{u, v\} \mid u \in V, v \in V\}$. On appelle graphe, noté $G = (V, E)$, la structure $< V, E >$. Les*

éléments de V (resp. de E) sont appelés des sommets (resp. des arêtes). Soit $e = \{u, v\}$ une arête, les sommets u et v sont appelés les extrémités de l'arête e.

Définition 2.6 (Graphe orienté) *Soient V et A deux ensembles, tels que $A \subseteq \{(u, v) \mid u \in V, v \in V\}$. On appelle graphe orienté, noté $G = (V, A)$, la structure $< V, A >$. Les éléments de V (resp. de A) sont appelés des sommets (resp. des arcs). Soit $a = (u, v)$ un arc, on appelle le sommet u (resp. v) la source (resp. la destination) de l'arc a.*

Dans certains réseaux biologiques, les liens entre les sommets peuvent être soit orientés (arcs) soit non-orientés (arêtes). Afin de modéliser les relations de ce type, nous devons donc définir les graphes mixtes.

Définition 2.7 (Graphe mixte) *Soient V, E et A trois ensembles, tels que $E \subseteq \{\{u, v\} \mid u \in V, v \in V\}$ et $A \subseteq \{(u, v) \mid u \in V, v \in V\}$. On appelle graphe mixte, noté $G = (V, E, A)$, la structure $< V, E, A >$. Les éléments de V (resp. de E et de A) sont appelés des sommets (resp. des arêtes et des arcs).*

Remarque : Un graphe non-orienté (resp. orienté) est un cas particulier de graphe mixte tel que $A = \emptyset$ (resp. $E = \emptyset$).

Les définitions suivantes données pour les graphes mixtes peuvent trivialement être simplifiées dans les cas particuliers des graphes non-orientés et des graphes orientés.

Définition 2.8 (Sous-graphe) *Soit $G = (V, E, A)$ un graphe mixte. On appelle $G' = (V', E', A')$ sous-graphe de G si et seulement si les quatre conditions suivantes sont vérifiées :*
- *G' est un graphe mixte.*
- *$V' \subseteq V$.*
- *$E' \subseteq E$*
- *$A' \subseteq A$.*

Définition 2.9 (Sous-graphe induit) *Soit $G = (V, E, A)$ un graphe mixte. On appelle $G' = (V', E', A')$ sous-graphe induit de G par V' si et seulement si les conditions suivantes sont vérifiées :*
- *G' est un sous-graphe de G.*
- *$E' = \{\{u, v\} \mid \{u, v\} \in E, u \in V' \text{ et } v \in V'\}$.*
- *$A' = \{(u, v) \mid (u, v) \in A, u \in V' \text{ et } v \in V'\}$.*

Le sous-graphe G' de G induit par V' est noté $G[V']$.

Remarque : Pour tout ensemble $V' \subseteq V$, on peut construire le sous-graphe induit G' de G par V'.

Définition 2.10 (Voisinage d'un sommet) *Soient $G = (V, E, A)$ un graphe mixte et $u \in V$. on appelle voisinage de u dans G, noté $N_G(u)$ l'ensemble $\{v \mid \{u, v\} \in E\} \cup \{v \mid (u, v) \in A \text{ ou } (v, u) \in A\}$.*

Remarque : Il est clair que $N_G(u) \subseteq V$.

Définition 2.11 (Adjacence d'un sommet) *Soient $G = (V, E, A)$ un graphe mixte et $u \in V$. On appelle adjacence de u dans G, noté $adj_G(u)$ l'ensemble $\{\{u, v\} \mid \{u, v\} \in E\} \cup \{(u, v) \mid (u, v) \in A\} \cup \{(v, u) \mid (v, u) \in A\}$.*

Remarque : Il est clair que $adj_G(u) \subseteq E \cup A$.

Définition 2.12 (Adjacence entrante et sortante d'un sommet) *Soient $G = (V, A)$ un graphe orienté et $u \in V$. On appelle adjacence entrante (resp. sortante) de u dans G, notée $adj_G^-(u)$ (resp. $adj_G^+(u)$), l'ensemble $\{(v, u) \mid (v, u) \in A\}$ (resp. $\{(u, v) \mid (u, v) \in A\}$).*

Définition 2.13 (Degré d'un sommet) *Soient $G = (V, E, A)$ un graphe mixte et $u \in V$. On appelle degré de u dans G, noté $deg_G(u)$ la quantité $|adj_G(u)|$.*

Définition 2.14 (Degrés entrant et sortant d'un sommet) *Soient $G = (V, A)$ un graphe orienté et $u \in V$. On définit les degré entrant et sortant de u dans G, respectivement notés $deg_G^-(u)$ et $deg_G^+(u)$ comme suit :*

$$deg_G^-(u) = |adj_G^-(u)| \text{ et } deg_G^+(u) = |adj_G^+(u)| \ .$$

Remarque : Le degré d'un sommet u dans un graphe orienté $G = (V, A)$ est $deg_G(u) = deg_G^-(u) + deg_G^+(u)$.

Définition 2.15 (Degré moyen d'un graphe) *Soit $G = (V, E, A)$ un graphe mixte. Le degré moyen de G, noté $deg_{avg}(G)$ est défini comme suit :*

$$deg_{avg}(G) = \frac{1}{|V|} \sum_{u \in V} deg_G(u).$$

Il est clair que $deg_{avg}(G) = \frac{2 \cdot |E|}{|V|}$.

Définition 2.16 (Valuation) *Soient $G = (V, E, A)$ un graphe mixte et K un ensemble. On appelle valuation des sommets (resp. des arcs et arêtes) du graphe toute application $f : V \to K$ (resp. $f : E \cup A \to K$). On dit alors que G est sommet-valué (resp. arc-arête-valué) et on le note $G = (V, E, A, f)$.*

Remarque : Si G est un graphe orienté (resp. non-orienté), on parle alors de graphe orienté arc-valué (resp. graphe arête-valué).

Définition 2.17 (Chemin non-orienté) *Soit $G = (V, E)$ un graphe non-orienté. On appelle chemin non-orienté (ou chemin) dans G, une séquence $(v_1, e_1, v_2, e_2, ..., e_{k-1}, v_k)$ avec :*

- $\forall i \in [1..k]$, $v_i \in V$
- $\forall i \in [1..k-1]$, $e_i = \{v_i, v_{i+1}\} \in E$

- $\forall i, j \in [1..k[,\ i \neq j,\ v_i \neq v_j$ *et*
- $\forall i, j \in [1..k],\ i \neq j,\ e_i \neq e_j$.

Définition 2.18 (Chemin orienté) *Soit $G = (V, A)$ un graphe orienté. On appelle chemin orienté dans G, une séquence $(v_1, e_1, v_2, e_2, ..., e_{k-1}, v_k)$ avec :*
- $\forall i \in [1..k],\ v_i \in V$
- $\forall i \in [1..k-1],$ *ou* $e_i = (v_i, v_{i+1}) \in A$
- $\forall i, j \in [1..k],\ i \neq j,\ v_i \neq v_j$ *et* $e_i \neq e_j$.

Remarque : Dans un graphe mixte, un chemin mixte est une combinaison de chemins non-orientés et orientés.
Remarque : Un chemin ne passe pas deux fois par le même sommet, le même arc ou la même arête.
Remarque : On dit que l'on peut atteindre un sommet v depuis un sommet u si et seulement si il existe un chemin (non-orienté, orienté ou mixte) tel que $u = v_1$ et $v = v_k$.

Définition 2.19 (Longueur valuée d'un chemin mixte) *Soient $G = (V, E, A, w)$ un graphe mixte arc-arête-valué, $u \in V$, $v \in V$ et $p = (u = v_1, e_1, v_2, e_2, ..., e_{k-1}, v_k = v)$ un chemin mixte de u à v. La longueur valuée du chemin p est $\sum_{i=1}^{k-1} w(e_i)$.*

Remarque : La longueur (non-valuée) du chemin p est le nombre d'arcs et arêtes qu'il contient.
Remarque : On définit de manière analogue les longueurs (valuées ou non) des chemins orientés et non-orientés.

Définition 2.20 (Distance dans un graphe) *Soient $G = (V, E, A)$ un graphe mixte, $u \in V$ et $v \in V$. On appelle distance dans G entre u et v, notée $d_G(u, v)$, la longueur du plus court chemin mixte dans G de u à v.*

Définition 2.21 (Distance valuée dans un graphe) *Soient $G = (V, E, A, w)$ un graphe mixte arc-arête-valué, $u \in V$ et $v \in V$. Soit $P = \{p_1, p_2, ..., p_l\}$ l'ensemble de tous les chemins mixtes dans G de u à v. On appelle distance valuée dans G entre u et v, notée $d_{G,w}(u, v)$, la plus petite longueur valuée des chemins de P.*

Définition 2.22 (Cycle et Graphe mixte acyclique) *Soient $G = (V, E, A)$ un graphe mixte et $u \in V$. On appelle cycle tout chemin de u à u. S'il n'existe pas de tel chemin dans G, alors le graphe G est un graphe mixte acyclique, noté **GMA**.*

Remarque : Si $E = \emptyset$ (resp. $A = \emptyset$), alors G est un graphe orienté acyclique, noté **DAG** pour *Directed Acyclic Graph* (resp. un graphe acyclique, noté **GA**).

Définition 2.23 (Arbre Enraciné) *Soit $G = (V, A)$ un graphe orienté. On dit que G est un arbre enraciné si et seulement si :*
- *G est un DAG.*

- $|A| = |V| - 1$.
- $\exists! r \in V, deg^-_G(r) = 0$.
- $\forall v \in V \setminus \{r\}, deg^-_G(v) = 1$.

Remarque : Le sommet r est appelé racine de l'arbre.
Remarque : L'ensemble des sommets u tels que $deg^+_G(u) = 0$, noté $feuilles(G)$, est appelé feuilles de l'arbre.
Remarque : Un arbre enraciné est un cas particulier de DAG enraciné.

Définition 2.24 (DAG enraciné) *Soit $G = (V, A)$ un DAG. On dit que G est un DAG enraciné si et seulement si $\exists! r \in V \mid deg^-_G(r) = 0$.*

Remarque : Le sommet r est appelé racine du DAG.
Remarque : L'ensemble des sommets u tels que $deg^+_G(u) = 0$, noté $feuilles(G)$, est appelé feuilles de G.

Définition 2.25 (Arbre libre) *Soit $G = (V, E)$ un graphe non-orienté. On dit que G est un arbre libre si et seulement si :*
- *G est un GA.*
- *$|E| = |V| - 1$.*

Définition 2.26 (Profondeur d'un sommet) *Soient $G = (V, A)$ un DAG enraciné de racine r et $u \in V$. On définit la profondeur de u dans G, notée $prof_G(u)$, comme suit :*

$$prof_G(u) = d_G(r, u)$$

Définition 2.27 (Graphe couvrant) *Soient $G = (V, E, A)$ et $H = (V_H, E_H, A_H)$ deux graphes mixtes. H est un graphe couvrant de G si et seulement si $V_H = V$, $E_H \subseteq E$ et $A_H \subseteq A$.*

Définition 2.28 (Arbre couvrant) *Soient $G = (V, A)$ un graphe orienté et $T = (V_T, A_T)$ un arbre. T est un arbre couvrant de G si et seulement si $V_T = V$ et $E_T \subseteq E$.*

Définition 2.29 (Graphe k-couvrant) *Soient $G = (V, E, A)$, $k \in \mathbb{N}^*$ et $H = (V_H, E_H, A_H)$. H est un graphe k-couvrant de G si et seulement si les deux conditions suivantes sont vérifiées :*
- *H est un graphe couvrant de G, et*
- *$\forall u, v \in V : d_H(u, v) \leq k \times d_G(u, v)$*

Définition 2.30 (Arbre k-couvrant) *Soient $G = (V, A)$, $k \in \mathbb{N}^*$ et $T = (V_T, A_T)$. T est un arbre k-couvrant de G si et seulement si les deux conditions suivantes sont vérifiées :*
- *T est un arbre couvrant de G, et*
- *$\forall u, v \in V : d_T(u, v) \leq k \times d_G(u, v)$*

Définition 2.31 (Graphe connexe) *Soit $G = (V, E)$ un graphe non-orienté. Le graphe G est connexe s'il existe un chemin (non-orienté) entre toute paire de sommets dans G.*

Définition 2.32 (Forêt) *Soit $G = (V, E)$ un graphe non-orienté. Le graphe G est une forêt si et seulement chaque composante connexe est un arbre libre.*

Remarque : On peut étendre ces trois définitions aux graphes orientés et mixtes en considérant leurs arcs comme des arêtes.

Définition 2.33 (Graphe k-connexe) *Soit $G = (V, E)$ un graphe non-orienté. On dit que G est k-connexe si et seulement s'il existe au moins k chemins sommets-disjoints entre chaque paire de sommets.*

Remarque : Un graphe 2-connexe (resp. 3-connexe) est dit *biconnexe* (resp. *triconnexe*).

Définition 2.34 (k-séparateurs) *Soit $G = (V, E)$ un graphe non-orienté. Pour tout $k \in \mathbb{N}^*$, les sommets $u_1, u_2, ..., u_k$ de G sont des k-séparateurs de G si et seulement si le graphe $G[V \setminus \{u_1, ..., u_k\}]$ n'est pas connexe.*

Définition 2.35 (Graphe complet) *Soit $G = (V, E)$ un graphe, on dit que le graphe G est un graphe complet si et seulement si $\forall u \in V$ et $\forall v \in V$, $\{u, v\} \in E$.*

Remarque : Un graphe complet à n sommets, noté K_n, possède $\frac{n(n-1)}{2}$ arêtes.

Définition 2.36 (Clique) *Soient $G = V, E)$ un graphe et $G' = (V', E')$ un sous-graphe de G. G' est une clique si et seulement si G' est un graphe complet à $|V'|$ sommets.*

Définition 2.37 (Stable) *Soit $G = (V, E, A)$ un graphe. Un stable (ou ensemble indépendant) est un sous-ensemble $V' \subseteq V$ tel que $G[V']$ est un graphe sans arête ni arc.*

Remarque : On dit alors que $G[V']$ est un graphe stable.

Définition 2.38 (Graphe biparti) *Soit $G = (V, E, A)$ un graphe. On dit que G est un graphe biparti si et seulement il existe V_1 et V_2 tels que $V_1 \subset V$ et $V_2 \subset V$, $V_1 \cap V_2 = \emptyset$, $V_1 \cup V_2 = V$ et V_1 et V_2 sont des stables.*

Définition 2.39 (Graphe planaire) *Soit $G = (V, E)$ un graphe. On dit que G est un graphe* planaire *si et seulement si G peut être dessiné sur une surface de* genre 0 *(e.g. un plan ou une sphère) sans croisement d'arêtes.*

Définition 2.40 (Graphe décomposé) *Soient $G = (V, E, A)$ un graphe mixte et $H = (V_H, A_H)$ un DAG enraciné tel que $feuilles(H) = V$. Le graphe décomposé $\{G, H\}$ est défini comme suit : tout sommet $u \in V_H \setminus feuilles(H)$ représente un groupe C_u de sommets de G tel que $C_u = \{v \in V \mid deg^+_H(v) = 0$ et il existe un chemin de u à v dans $H\}$ (cf figure 2.1).*

Remarque : Le graphe H est appelé DAG de décomposition.
Remarque : On appelle niveau i de (G, H), l'ensemble des sommets u de H tels que $prof_H(u) = i$.

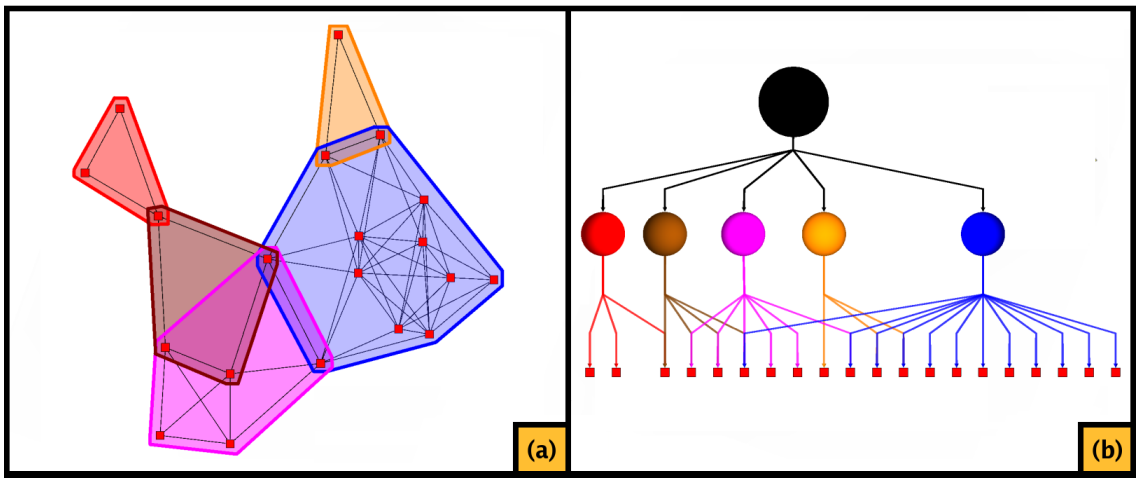

FIG. 2.1: Exemple de décomposition de graphe $\{G, H\}$. La figure (a) montre le graphe G et la figure (b) le DAG de décomposition H. Le sommet bleu de H représente l'ensemble des sommets du graphe G (feuilles dans H) que l'on peut atteindre dans le DAG depuis le sommet bleu.

Définition 2.41 (Graphe partitionné) *Soit (G, H) un graphe décomposé. (G, H) est un graphe partitionné si et seulement si H est un arbre enraciné.*

Remarque : Le graphe H est appelé arbre de partition.
Remarque : Lorsque le DAG de décomposition n'est pas un arbre, on parle alors de décomposition chevauchante.

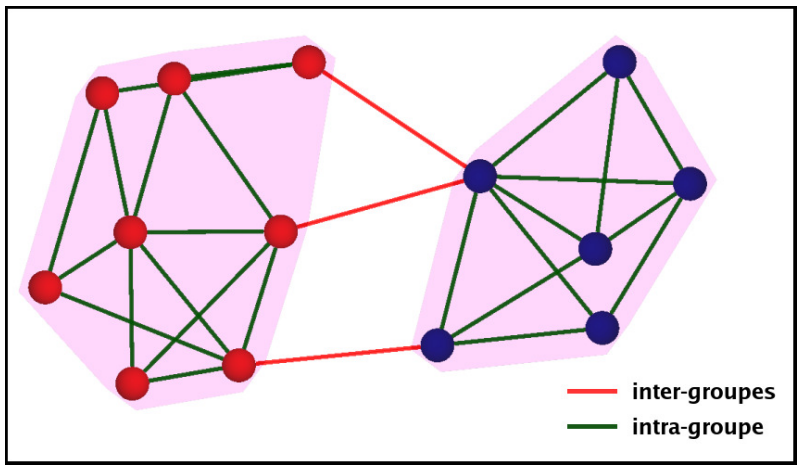

FIG. 2.2: Partition d'un graphe en deux groupes (en rose dans le dessin). Les arêtes vertes sont les arêtes intra-groupes tandis que les arêtes rouges sont les arêtes inter-groupes.

Définition 2.42 (Arête inter-groupes / intra-groupe) *Soient $G = (V, E)$ un graphe et $C = \{C_1, C_2, ..., C_k\}$ un décomposition des sommets de G. On dit que $e = \{u, v\} \in E$*

est une arête inter-groupes (resp. intra-groupe) s'il existe C_i et C_j tels que $u \in C_i$ et $v \in C_j$ (resp. s'il existe C_i tel que $u, v \in C_i$) (cf figure 2.2).

Remarque : On note $E(C_i, C_j)$ l'ensemble des arêtes reliant un sommet de C_i et un sommet de C_j.

Dans le cas de graphes décomposés, il est intéressant de représenter les sommets d'un même groupe par un sommet unique. Cela permet de simplifier la représentation tout en conservant l'organisation générale du graphe.

Définition 2.43 (Graphe quotient) *Soient (G, H) un graphe décomposé, $\{u_1, ..., u_p\}$ le niveau i de (G, H) et $C = \{C_1, C_2, ..., C_p\}$ l'ensemble des groupes correspondant dans G. Le graphe quotient $Q_i = (V_{Q_i}, E_{Q_i}, A_{Q_i})$ du niveau i de (G, H) est défini comme suit :*

- $V_{Q_i} = \{u_1, u_2, ..., u_p\}$.
- $e = \{u_j, u_k\} \in E_{Q_i}$ *si et seulement si* $j \neq k$ *et l'une des deux conditions suivantes est respectée :*
  - $\exists u \in C_j, v \in C_k$ *tels que* $\{u, v\} \in E$, *ou*
  - $C_j \cap C_k \neq \emptyset$
- $a = (u_j, u_k) \in A_{Q_i}$ *si et seulement si* $j \neq k$ *et* $\exists u \in C_j, v \in C_k$ *tels que* $(u, v) \in A$

Remarque : Un sommet (resp. arête et arc) de Q_G est appelé **métanoeud** (resp. **méta-arête** et **méta-arc**).

Définition 2.44 (Graphe quotient valué) *Soient $Q = (V_Q, E_Q, A_Q, w)$ un graphe mixte arête-arc valué. On dit que Q est un graphe quotient valué du niveau i d'un graphe mixte arc-arête-valué décomposé (G, H) si le graphe $Q' = (V_Q, E_Q, A_Q)$ est le graphe quotient du niveau i de (G, H).*

Remarque : Pour valuer une méta-arête ou un méta-arc, il existe plusieurs méthodes, notamment la valeur minimale, maximale, médiane ou encore moyenne des arêtes ou arcs qu'il ou elle représente.

Définition 2.45 (Graphe de dépendance) *Soient (G, H) un graphe décomposé, $\{u_1, u_2, ..., u_p\}$ le niveau i de (G, H) et $C = \{C_1, C_2, ..., C_p\}$ l'ensemble des groupes correspondant dans G. Le graphe dépendance $Dep_i(G, H) = (V_{Dep}, E_{Dep})$ du niveau i de (G, H) est défini comme suit :*

- $V_{Dep} = \{u_1, u_2, ..., u_p\}$.
- $e = \{u_j, u_k\} \in E_{Dep}$ *si et seulement si* $j \neq k$ *et* $C_j \cap C_k \neq \emptyset$

Remarque : Le graphe de dépendance d'un niveau i d'un graphe partitionné est un graphe sans arête.

2.2.2 Dessin de graphe

Les notions données dans cette partie sont définies pour les graphes non-orientés mais peuvent être trivialement adaptées aux graphes orientés et mixtes.

Définition 2.46 (Dessin planaire) *Soit $G = (V, E)$ un graphe. On appelle* dessin planaire *de G, un dessin de G sur une surface de genre 0 (e.g. un plan ou une sphère) dans lequel il n'y a aucun croisement d'arêtes.*

Définition 2.47 (Dessin en lignes droites) *oit $G = (V, E)$ un graphe. On appelle* dessin en lignes droites*, un dessin de G dans lequel chaque arête de G est représentée par un segment reliant les extrémités de l'arête.*

Définition 2.48 (Dessin en lignes brisées) *Soit $G = (V, E)$ un graphe. On appelle* dessin en lignes brisées *de G, un dessin de G dans lequel chaque arête de G est représentée par une ligne brisée, i.e. une séquence de segments contigus.*

Définition 2.49 (Point de contrôle) *Dans un dessin en lignes brisées, on appelle* point de contrôle *d'une arête, chaque « brisure » de la lignes brisée représentant l'arête, i.e. l'extrémité commune de chaque paire de segments contigus.*

Définition 2.50 (Résolution angulaire) *Dans un dessin en lignes droites, on appelle* résolution angulaire *le plus petit angle formé par deux segments (représentant deux arêtes) adjacents à un même sommet.*

Remarque : Cette définition peut être adaptée aux dessins en lignes brisées en considérant chaque point de contrôle comme un sommet.

Définition 2.51 (Dessin orthogonal) *Soit $G = (V, E)$ un graphe. On appelle* dessin orthogonal *de G, un dessin en lignes brisées de G dont les angles formés par deux segments adjacents (de deux arêtes) ou par deux segments contigus d'une ligne brisée (d'une même arête) sont orthogonaux ou sont plats.*

Définition 2.52 (Face d'un graphe) *Dans un dessin planaire les régions du dessin délimitées par les lignes représentant les arêtes sont appelées* faces*.*

Remarque : Il est possible de calculer les faces en ne se basant que sur la topologie du graphe.
Remarque : Dans le plan, un dessin planaire ne possède qu'une face non bornée, appelée *face extérieure.*

Définition 2.53 (Dessin convexe) *Un* dessin convexe *est un dessin planaire dont chaque face est un polygone convexe.*

2.3 Réseaux biologiques

Nous introduisons dans cette partie les notions biologiques de base nécessaires à la compréhension des travaux présentés dans cette thèse.

Définition 2.54 (Gène) *Un* gène *est une portion d'Acide DésoxyriboNucléique (séquence d'ADN), et plus précisément une séquence de molécules (bases azotées). Cette séquence d'ADN est destinée à être transcrite en acide ribonucléique (ARN). La molécule d'ARN ainsi produite est généralement traduite en protéine.*

Définition 2.55 (Protéine) *Une* protéine *est une macromolécule composée par une (ou plusieurs) séquence(s) d'acides aminés liés entre eux par des liaisons peptidiques. Une protéine peut avoir des fonctions très diverses, par exemple un rôle de catalyseur (i.e. est nécessaire pour qu'une réaction se produise, on dit alors que la protéine est une enzyme), ou un rôle de communication (i.e. est un message chimique permettant aux différentes parties de l'organisme de communiquer) ou encore un rôle structurel (i.e. permet la consolidation de la cellule).*

Définition 2.56 (Métabolisme ou Réseau métabolique) *Le* métabolisme ou réseau métabolique *est l'ensemble des réactions de synthèse (génératrices de matériaux), et de dégradation (génératrices d'énergie), qui s'effectuent au sein d'une cellule ou d'un organisme.*

Définition 2.57 (Voie métabolique) *Une* voie métabolique *est une sous-partie du réseau métabolique permettant de réaliser une fonction biologique particulière.*

Remarque : Les différentes voies métaboliques d'un organisme se chevauchent, en effet les composés d'entrée de certaines voies sont les composés de sortie d'autres.

2.4 Autres définitions

Définition 2.58 (Diagramme de Voronoï) *Soit $P = \{u_1, u_2, ..., u_n\}$ un ensemble de n points du plan, appelés sites. Le* diagramme de Voronoï *de P est une division du plan en n cellules, une pour chaque site de P. Soit un site $u_i \in P$, on note $Vor(u_i)$ la cellule associée à u_i, et $Vor(u_i)$ contient tous les points du plan au moins aussi proches de u_i que de tout autre site (cf figure 2.3).*

Définition 2.59 (Complexité d'un algorithme) *La complexité d'un algorithme est la quantité d'espace mémoire utilisée (complexité en espace) et/ou le nombre d'opérations élémentaires réalisées par l'algorithme (complexité en temps) en fonction de la taille des données en entrée de l'algorithme.*

Remarque : Dans cette thèse, on ne s'intéressera qu'à la complexité dans le pire des cas, notée $O(f(p))$, où $f(p)$ est fonction des paramètres d'entrée.

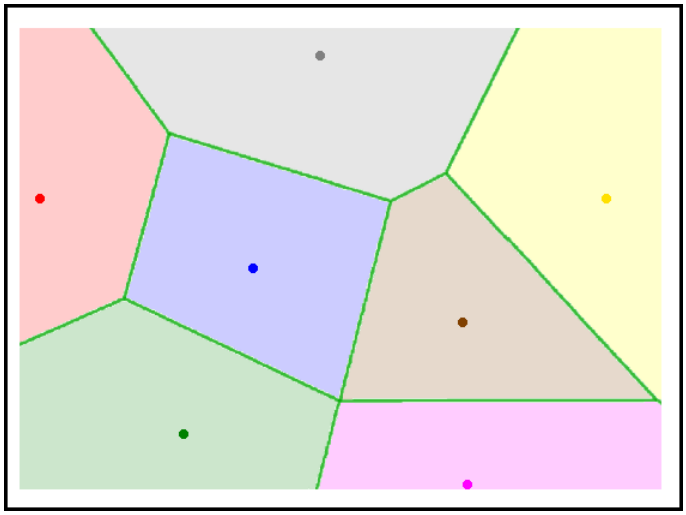

FIG. 2.3: Les segments verts représentent le diagramme de Voronoï des 7 points du plan (en bleu, vert foncé, jaune, rose, marron, gris et rouge). Les cellules de chaque site sont indiquées par le code de couleurs.

Définition 2.60 (Coloration de graphe) *Soit $G = (V, E)$ un graphe. On appelle coloration de G une fonction $col : V \to C = \{c_1, ...c_p\}$, et pour tout $1 \leq i \leq p$, c_i est appelé couleur.*

Définition 2.61 (Classe de couleur) *Soient $G = (V, E)$ un graphe, $C = \{c_1, ...c_p\}$ un ensemble de couleurs et col une coloration de G dans C. La classe de couleur c_i du graphe G est définie comme suit :*

$$classe(G, c_i) = \{u \in V | col(u) = c_i\}$$

.

Chapitre 3

Décomposition et Visualisation : État de l'art

Dans ce chapitre, nous présentons les travaux principaux dans les domaines abordés dans cette thèse. Dans la section 3.1, nous présentons les travaux portant sur la décomposition de graphe, et plus particulièrement ceux utilisés en visualisation d'information. Puis dans la section 3.2, nous décrivons différentes techniques de visualisation de graphe. Des méthodes particulières utilisées pour mettre en évidence la partition des sommets d'un graphe sont explicitées dans la partie 3.3. Enfin, nous verrons les principaux travaux relatifs à la représentation de données biologiques, en l'occurrence des réseaux métaboliques.

3.1 Décomposition de graphes

Dans le cas de données complexes, i.e. contenant plusieurs milliers voire plusieurs millions d'éléments, il est nécessaire d'extraire automatiquement de l'information. Cette étape d'extraction permet non seulement une analyse initiale des données mais surtout elle permet de construire une abstraction visuelle. Pour ce faire, il existe de nombreuses techniques de décomposition de graphe. Dans [114], Schaeffer présente les techniques les plus connues du domaine. Selon Schaeffer, il existe deux principales approches : les approches globales et les approches locales.

Parmi les approches globales, Schaeffer [114] cite les méthodes agglomératives (e.g l'algorithme de Newman [104]) et divisives. Les principales méthodes divisives sont les méthodes basées sur des métriques (e.g. les algorithmes de Auber *et al.* [13] et de Newman et Girvan [105]) mais aussi des méthodes spectrales (e.g. l'algorithme de McSherry [101]) ou encore des méthodes basées sur des marches aléatoires (e.g. l'algorithme MCL de van Dongen [126]).

Dans le cas de grands graphes, il n'est pas possible d'utiliser des méthodes quadratiques en temps et/ou en espace sur les ordinateurs classiques. Prenons l'exemple du graphe dont les sommets représentent des acteurs d'Hollywood et une arête relie toute paire de sommets dont les acteurs ont joué dans un film commun, appelé *graphe d'Hollywood*. La composante connexe la plus grande contient 117948 sommets et 1917841 arêtes, appliquer une méthode de décomposition en temps $O(|V|^2)$ conduirait à plus de 13 milliards d'opérations. D'autre

part, si l'espace mémoire nécessaire pour réaliser cette décomposition est aussi $O(|V|^2)$, alors cela pose un problème de quantité de mémoire vive puisque les ordinateurs actuels possèdent au mieux 4 Go ce qui ne suffit pas pour stocker autant d'éléments. Une solution consiste à utiliser des grilles de calcul et à paralléliser massivement l'algorithme, cependant les utilisateurs doivent alors avoir à disposition un tel matériel. Si l'on ne considère que les ordinateurs classiques, pour décomposer de tels graphes, il faut utiliser des algorithmes de décomposition linéaires ou sub-linéaires. Les méthodes locales (e.g. [113]) permettent de trouver une décomposition tout en respectant cette contrainte de complexité en temps et surtout en espace. L'idée principale de ces approches est de calculer localement les groupes de la décomposition sans avoir connaissance de la structure globale du graphe mais seulement du voisinage proche des groupes.

Pour une étude détaillée de l'existant dans le domaine de la décomposition de graphe, le lecteur peut se référer à l'article de Schaeffer [114]. Nous ne présentons dans cette partie que les travaux les plus couramment utilisés en visualisation d'information. Nous allons tout d'abord présenter trois mesures permettant d'évaluer la qualité d'une décomposition, puis nous verrons les méthodes divisives, agglomératives, et enfin une méthode basée sur le degré des sommets.

3.1.1 Qualité d'une décomposition

Le critère le plus largement admis pour qu'un ensemble de groupes forme une « bonne » décomposition du graphe est une densité intra-groupe[1] élevée et une densité inter-groupes[2] faible. Nous présentons ici deux mesures permettant d'évaluer la qualité d'une décomposition : la *modularité* [105] et la mesure MQ[98].

3.1.1.1 Modularité

Dans [105], Newman et Girvan introduisent une mesure de qualité, appelée *modularité*. Cette mesure, notée Q, est définie comme suit :

$$Q = \sum_i (e_{ii} - {a_i}^2) \tag{1}$$

où $a_i = \sum_j e_{ij}$ et $e_{ij} = |E(C_i, C_i)|/|E|$ est le nombre d'arêtes reliant un sommet du groupe C_i à un sommet du groupe C_j normalisé par le nombre d'arêtes du graphe. Cette mesure de qualité permet en fait de « comparer » le nombre d'arêtes internes à chaque groupe au nombre d'arêtes que ces groupes devraient contenir dans un graphe aléatoire. On peut

[1] La densité d'un groupe est le ratio entre le nombre d'arêtes du groupe et le nombre d'arêtes qu'un (sous-)graphe complet ayant le même nombre de sommets contient, cf section 4.4.

[2] La densité inter-groupes deux groupes est le ratio entre nombre d'arêtes reliant ces deux groupes et nombre maximal d'arêtes pouvant les relier, cf section 4.4.

réécrire la formule de la manière suivante :

$$\begin{aligned} Q &= \frac{1}{|E|}\sum_i \left(|E(C_i)| - \frac{\sum_j |E(C_i,C_j)|}{|E|} \cdot \sum_j |E(C_i,C_j)| \right) \\ Q &= \frac{1}{|E|}\sum_i \left(|E(C_i)| - p_{C_i} \cdot \sum_j |E(C_i,C_j)| \right) \end{aligned} \tag{2}$$

où p_{C_i} est la probabilité qu'une arête ait une extrémité dans le groupe C_i. Pour un groupe C_i, sachant qu'il y a $\sum_j |E(C_i,C_j)|$ arêtes avec au moins une extrémité dans C_i, $p_{C_i} \cdot \sum_j |E(C_i,C_j)|$ représente donc le nombre d'arêtes intra-groupes qu'il y aurait dans C_i dans un graphe aléatoire. Si la valeur de modularité obtenue par une décomposition est 0, cela signifie que cette décomposition est équivalente (en terme de qualité) à une décomposition aléatoire.

3.1.1.2 Mesure de qualité MQ

Mancoridis *et al.* introduisent dans [98] une mesure appelée MQ. Cette mesure compare le nombre d'arêtes intra-groupe (resp. inter-groupes) au nombre maximal possible de telles arêtes. Considérons un graphe $G = (V, E)$ et une partition $C = \{C_1, C_2, ..., C_k\}$ des sommets de G, MQ est défini comme suit :

$$MQ = \frac{1}{k}\sum_i s(C_i, C_i) - \frac{1}{k(k-1)}\sum_{i,j\neq i} s(C_i, C_j) \tag{3}$$

où $s(C_i, C_j) = \frac{|E(C_i,C_j)|}{|C_i|\cdot|C_j|}$. Nous définissons les deux notations suivantes :

$$MQ^+ = \frac{1}{k}\sum_i s(C_i, C_i) \tag{4}$$

et,

$$MQ^- = \frac{1}{k(k-1)}\sum_{i,j\neq i} s(C_i, C_j) \tag{5}$$

On a donc :

$$MQ = MQ^+ - MQ^- \tag{6}$$

Dans cette équation, MQ^+ représente la cohésion interne des groupes $C_1, ..., C_k$ tandis que MQ^- représente la cohésion externe des groupes (ou inter-groupes).

3.1.2 Approches divisives

Les méthodes divisives de partitionnement utilisent généralement une fonction permettant d'évaluer l'importance d'une arête dans un graphe. Puis en utilisant une borne et une mesure de qualité du partitionnement, les arêtes les moins importantes (arêtes inter-groupes) sont retirées.

Partant d'un état où la décomposition ne contient qu'un seul groupe (contenant tout le graphe), les arêtes sont supprimées successivement. Si la suppression d'une (ou plusieurs) arêtes déconnecte un groupe, alors ce groupe est divisé en deux nouveaux groupes. Ce processus est répété jusqu'à ce qu'il n'y ait plus aucune arête (i.e. chaque groupe contient exactement un sommet).

Nous présentons dans cette partie deux algorithmes connus en visualisation d'information : l'algorithme de Newman et Girvan [105] et l'algorithme de Auber *et al.* [13].

Dans [105], Newman et Girvan ont proposé un algorithme divisif utilisant la *Betweenness Centrality* des arêtes et la mesure de modularité Q.

Betweenness Centrality [53] : Dans [53], Freedman définit une métrique appelée *Betweenness Centrality*. Cette mesure permet de quantifier la centralité d'un sommet ou d'une arête dans un graphe. La *Betweenness Centrality* d'un sommet u (resp. d'une arête e) est le nombre de plus courts chemins passant par u (resp. par e) entre chaque paire de sommets, normalisé par le nombre de plus court chemins entre ces sommets. Plus formellement, la *Betweenness Centrality* d'un sommet v est définie comme suit :

$$Bet(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}} \tag{7}$$

où σ_{st} est le nombre de plus courts chemins de s à t et $\sigma_{st}(v)$ est le nombre de plus courts chemins de s à t passant par v. La *Betweenness Centrality* d'une arête est définie de manière analogue. Dans [24] et [103], Brandes et Newman donnent des algorithmes pour calculer la *Betweenness Centrality* des sommets et arêtes d'un graphe en temps $O(|V| \cdot |E|)$.

Algorithme : Dans [105], Newman et Girvan ont proposé l'algorithme divisif suivant :

1. Calculer la *Betweenness Centrality* de chaque arête dans le graphe
2. Supprimer l'arête de plus forte centralité, si cette suppression déconnecte un groupe alors créer deux nouveaux groupes
3. Calculer la centralité de chaque arête dans le graphe résultant
4. Aller à l'étape 2 tant qu'il existe des arêtes

A chaque fois que la suppression d'une arête déconnecte un groupe, un nouveau niveau dans la hiérarchie est créé. Cela permet d'obtenir un partitionnement de graphe sur plusieurs niveaux. Puis pour trouver le « meilleur » niveau de la hiérarchie, les auteurs utilisent ensuite la mesure de modularité.

Une autre approche divisive est donnée par Auber *et al.* dans [13]. Dans cet article, les auteurs évaluent la « force » d'une arête (*Strength*) [13, 32] puis utilisent la mesure de qualité MQ pour calculer la valeur optimale de la borne.

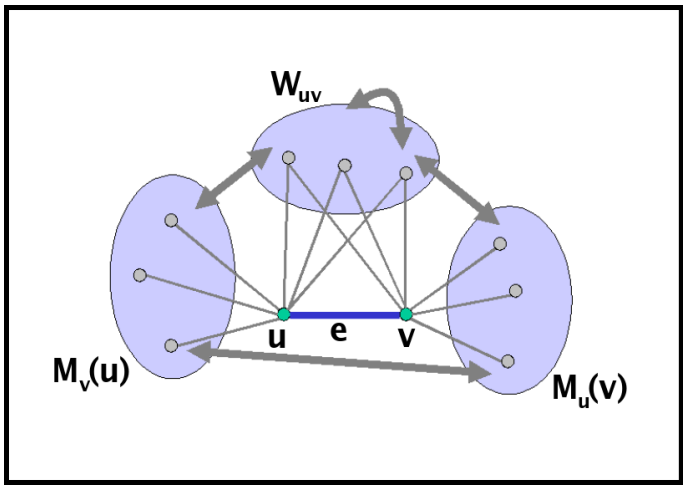

FIG. 3.1: La métrique Strength de l'arête $e = (u, v)$ est calculée en comparant le nombre de cycles de tailles 3 et 4 passant par cette arête au nombre maximal de tels cycles.

Métrique Strength [13, 32] : Cette métrique permet de quantifier la cohésion du voisinage d'une arête (plus précisément, du voisinage des extrémités d'une arête). Par conséquent, la métrique Strength [13, 32] permet d'évaluer si une arête est intra-communautaire (cohésion forte autour de l'arête) ou inter-communautaire (cohésion faible). Pour la définir, nous devons introduire quelques notations. Soient u et v deux sommets du graphe, on note

$$M_u(v) = N_G(v) \setminus (N_G(u) \cup \{u\})$$

l'ensemble des voisins de v (en excluant u) qui ne sont pas voisins de u, et on note

$$W_{uv} = N_G(u) \cap N_G(v)$$

l'ensemble des sommets voisins de u et de v. Soient A et B deux ensembles de sommets, on note $E(A, B)$ l'ensemble des arêtes reliant un sommet de A à un sommet de B. Enfin,

$$s(A, B) = |E(A, B)|/(|A| \cdot |B|)$$

est le ratio entre le nombre d'arêtes reliant A et B et le nombre maximal d'arêtes qu'il pourrait y avoir entre ces deux ensembles [3]. La métrique Strength d'une arête $e = (u, v)$, notée $w_s(e)$ est calculée comme suit :

$$w_s(e) = \gamma_3(e) + \gamma_4(e) \tag{8}$$

où :

$$\begin{aligned} \gamma_3(e) &= |W_{uv}|/(|M_v(u)| + |W(u, v)| + |M_u(v)|) \\ \gamma_4(e) &= s(M_v(u), M_u(v)) + s(M_v(u), W_{uv}) + s(W_{uv}, M_u(v)) + s(W_{uv}) \end{aligned} \tag{9}$$

Après discussion avec les auteurs de l'article, nous avons utilisé la version de la métrique Strength disponible dans le logiciel de visualisation de graphe Tulip [12]. La métrique Strength y est définie comme suit :

$$w_s(e) = \frac{\gamma_{3,4}(e)}{\gamma_{max}(e)} \tag{10}$$

[3] Lorsque $A = B$, on a $s(A, A) = s(A) = 2 \cdot |E(A)|/(|A| \cdot (|A| - 1))$

où :

$$\begin{aligned}\gamma_{3,4}(e) &= |W_{uv}| + E(M_v(u), M_u(v))| + |E(M_v(u), W_{uv})| + |E(W_{uv}, M_u(v))| + |E(W_{uv})| \\ \gamma_{max}(e) &= M_v(u)| + |W(u,v)| + |M_u(v)| \\ &\quad + |M_v(u)||M_u(v)| + |M_v(u)||W_{uv}| + |W_{uv}||M_u(v)| + |W_{uv}|(|W_{uv}| - 1)/2\end{aligned} \tag{11}$$

Cette métrique compte donc pour chaque arête le nombre de cycles de longueur 3 et 4 passant par cette arête et normalise cette valeur par le nombre maximum possible de tels cycles. La figure 3.1 illustre comment est calculée cette métrique sur une arête.

Algorithme : Dans l'approche utilisée par Auber *et al.*, les arêtes ne sont pas supprimées une à une. Pour une borne donnée, toutes les arêtes ayant une valeur de Strength inférieure à cette borne sont retirées. L'algorithme de Auber *et al.* calcule pour chaque borne, la valeur de MQ du partitionnement correspondant. Le résultat de cet algorithme est la partition obtenue en utilisant la borne optimale (i.e. la borne permettant de maximiser MQ).

3.1.3 Approches agglomératives

Les méthodes agglomératives tentent généralement de maximiser une fonction permettant d'évaluer la *qualité* d'un partitionnement. Partant d'un état où chaque sommet est dans un groupe et chaque groupe ne contient qu'un seul sommet, l'algorithme évalue à chaque itération quelle est la paire de groupes dont la « fusion » permettrait d'augmenter le plus (ou de diminuer le moins) la qualité de la décomposition et fusionne ces deux groupes en un nouveau groupe. La plupart de ces algorithmes diffèrent par la manière dont la mesure de qualité est calculée, et par conséquent par la définition même d'un « bon » partitionnement.

L'un des algorithmes agglomératifs les plus connus est celui de Newman [104]. Dans cet article, l'algorithme utilise la mesure de modularité définie dans la section précédente. Partant d'une partition du graphe dans laquelle chaque groupe ne contient qu'un seul sommet, cet algorithme fusionne à chaque itération les deux groupes (ou communautés) augmentant le plus (ou diminuant le moins) la valeur de Q jusqu'à ce qu'il n'y ait plus qu'un seul groupe (contenant le graphe entier). Afin d'accélérer la temps de calcul de l'algorithme, Newman explique dans [104] que seules les fusions de groupes reliés par au moins une arête doivent être testées. En effet, fusionner deux groupes non-connectés ne peut que diminuer la mesure de modularité.

3.1.4 Approches topologiques

Dans [117], Seidman présente une méthode basée sur le degré (et donc la topologie) pour décomposer un graphe. Cette méthode permet de trouver des groupes de sommets appelés *K-cores* du graphe. Un *k-core* est un sous-graphe induit tel que les sommets de ce sous-graphe ont un degré supérieur ou égal à k dans le sous-graphe. D'après cette

définition, tout sommet appartenant à un k-core, appartient aussi à tout k'-core, $k' \leq k$. On peut donc définir la *coreness* d'un sommet u de la manière suivante :

Définition 3.1 (Coreness d'un sommet) *Soient $G = (V, E)$ un graphe et u un sommet de G. La* coreness *de u dans G est c si et seulement si u appartient au c-core de G mais pas au $(c+1)$-core de G.*

Par abus de langage, on appellera k-core, l'ensemble des sommets ayant une *coreness* k. Pour trouver ces ensembles de sommets, l'algorithme fonctionne comme suit :

1. $deg_{courant} = 1$
2. Retirer tous les sommets de degré $deg_{courant}$
3. Tant qu'il reste des sommets de degré $deg_{courant}$, revenir à l'étape 2
4. Les sommets retirés lors de l'étape 2 et 3 induisent un $deg_{courant}$-core
5. S'il reste des sommets dans le graphe, incrémenter $deg_{courant}$ et revenir à l'étape 2

D'après la définition d'un *k-core*, il est évident qu'un *k-core* peut être un sous-graphe non-connexe. Il est donc possible d'augmenter la densité des groupes trouvés en décomposant chaque *k-core* en composantes connexes.

Récemment, des articles ont été publiés sur des décompositions de graphe basées sur la détection de sous-structures topologiques particulières. Ces algorithmes sont généralement intégrés dans des processus de dessin de graphe (e.g [122, 4, 7, 10]) et sont décrites dans la partie 3.2.6.

3.2 Visualisation de graphe

Les algorithmes de dessin de graphe ont été largement étudiés. En particulier, des algorithmes efficaces ont été mis en place pour certaines classes de graphes notamment les arbres, les graphes planaires ou encore les graphes orientés acycliques. Cependant de nombreux graphes construits à partir de données réelles ne font pas partie de ces classes de graphes.

Dans cette section, nous présentons les travaux portant sur le dessin de classes particulières de graphes : les graphes planaires et les graphes acycliques. Puis nous verrons les principales approches utilisées pour représenter les graphes (sans prendre en compte leurs classes). Ces approches peuvent être classées en quatre familles : les méthodes classiques par analogie physique, les méthodes par échantillonnage et modèle de forces, les méthodes basées sur les matrices et enfin les méthodes basées sur la détection de sous-structures (i.e. de partitionnement).

3.2.1 Algorithmes de dessin de graphe planaire

Il existe de très nombreux travaux portant sur le dessin de graphes planaires, nous ne détaillerons pas tous ces travaux dans cette partie (pour plus de détails sur ce sujet, le lecteur peut se référer à [107]). Nous présentons dans cette partie les algorithmes utilisant des décompositions du graphe en une partition ordonnée des sommets $V_1, V_2, ..., V_p$, appelée

ordre canonique (en anglais, *canonical ordering*). Cet ordre canonique est généralement utilisé pour connaître l'ordre d'insertion des sommets dans le dessin et ainsi obtenir un dessin sans croisement d'arêtes.

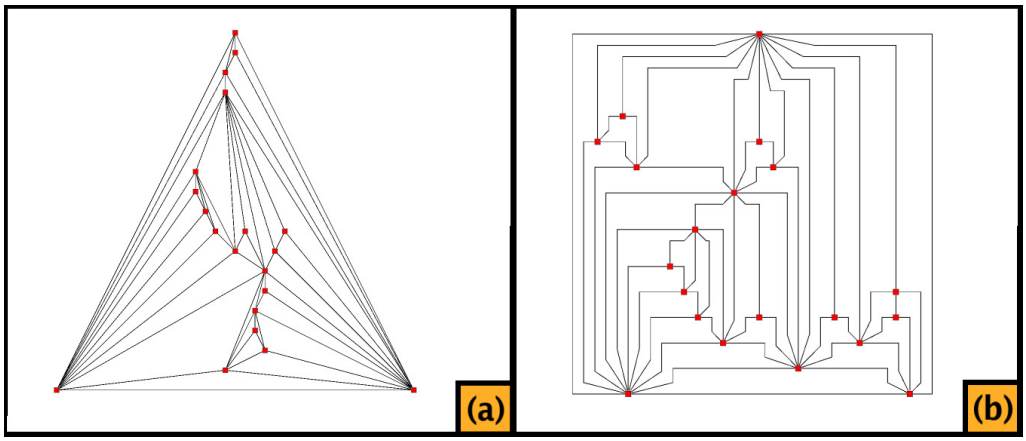

FIG. 3.2: Un exemple de graphe planaire triangulé dessiné par l'algorithme de De Fraysseix *et al.* [51] (a) et par l'algorithme de Gutwenger et Mutzel (b).

L'un des algorithmes de dessin (de graphe) planaire les plus connus est celui présenté dans [51] par De Fraysseix *et al.*. Cet algorithme permet de calculer un dessin planaire en lignes droites d'un graphe planaire triangulé (i.e. dont toutes les faces contiennent 3 sommets et arêtes) en temps $O(|V| \cdot log(|V|))$. D'autre part, De Fraysseix *et al.* donnent un algorithme permettant de construire un graphe planaire triangulé à partir d'un graphe planaire en temps linéaire. L'ordre canonique tel qu'il est défini par De Fraysseix *et al.* est un ensemble ordonné $V^* = \{v_1, v_2, ..., v_{|V|}\}$. Pour tout $1 \leq i \leq |V|$, on note G_i le sous-graphe induit par $V_1 \cup ... \cup V_i$ dans G. Pour construire un tel ordre, un sommet de la face extérieure de G_i est supprimé pour former le graphe G_{i-1}. Etant donné les propriétés de l'ordre de De Fraysseix, on peut ensuite aisément positionner les sommets dans l'ordre v_1, v_2,..., $v_{|V|}$ sans générer de croisement d'arêtes (cf figure 3.2.(a)).

Dans [81], Kant présente une généralisation de l'algorithme de De Fraysseix *et al.*. Cet algorithme permet de dessiner un graphe planaire triconnexe en temps $O(|V|)$. L'une des différences majeures entre l'algorithme de Kant et celui de De Fraysseix *et al.* est que l'algorithme de De Fraysseix *et al.* [51] utilise un ordre sur les sommets tandis que celui de Kant utilise une partition ordonnée des sommets. Dans cet article, Kant présente ensuite deux versions permettant d'obtenir deux types de représentations différentes : un dessin en lignes droites et un dessin orthogonal (si le degré de chaque sommet est au plus égal à 4).

Enfin, l'algorithme de Gutwenger et Mutzel présenté dans [64] permet de dessiner les graphes planaires biconnexes en temps linéaire (cf figure 3.2.(b)). Dans cet article, les auteurs donnent une généralisation de l'ordre canonique de Kant aux graphes biconnexes. Cet algorithme permet de construire un dessin en lignes brisées du graphe tel que chaque arête a au plus 3 points de contrôle et permet d'autre part d'optimiser la résolution angulaire (i.e. l'angle minimum formé par deux arêtes adjacentes). Le détail de cet algorithme est donné dans la partie 6.5.3.2 de cette thèse.

3.2.2 Algorithmes de dessin de graphe orienté acyclique

L'algorithme le plus connu dans la communauté de dessin de graphe et de visualisation d'information pour dessiner un DAG est celui de Sugiyama *et al.* [123]. Dans cet article, Sugiyama *et al.* présentent un l'algorithme de dessin hiérarchique dont la complexité en temps est $O(|V| \cdot |E| \cdot log(|E|))$ et la complexité en mémoire $O(|V| \cdot [E|)$. Cet algorithme de dessin hiérarchique (ou encore *k-layered*) se décompose en quatre étapes principales :

1. Transformation du graphe afin d'obtenir un graphe sans cycle, dont les sommets sont positionnés sur des couches $L_1, L_2, ..., L_p$ (avec $p = O(|V|)$) et si une arête $e = (u, v)$ et $u \in L_i$, $v \in L_j$ et $i > j + 1$ (i.e. l'arête e « coupe » plusieurs couches), alors e est « remplacée » par la séquence de sommets fictifs et contigus : $w_1, w_2, ..., w_{i-(j+1)}$. Le graphe obtenu est dit hiérarchie propre.
2. Minimisation du nombre de croisements d'arêtes par permutation des sommets d'une même couche
3. Calcul des positions horizontales des sommets de chaque couche afin d'obtenir un dessin dans lequel les arêtes sont aussi courtes que possible et le nombre de points de contrôle par arêtes « longues » (i.e. recouvrant plusieurs couches) aussi faible que possible.
4. Afficher le dessin en remplaçant les sommets et arêtes fictifs par les arêtes originales.

Pour chacune des étapes 2 et 3, Sugiyama *et al.* présentent deux méthodes : une méthode exacte et une méthode heuristique. Les méthodes heuristiques *BC* (pour *BarycentriC method*) et *PR* (pour *PRiority layout method*) permettent respectivement de minimiser le nombre de croisements d'arêtes et de minimiser les longueurs des arêtes. La méthode *BC* consiste à permuter les sommets de chaque couche L_1 à L_p. Considérons la couche $L_i = \{u_1^i, u_2^i, ...u_{|L_i|}^i\}$, les sommets de L_i sont positionnés aux barycentres de leurs voisins de L_{i-1} (ou de L_{i+1}). Dans l'algorithme de Sugiyama *et al.*, la méthode *BC* est répétée tant qu'une permutation (au moins) est effectuée ou que le nombre d'itérations est inférieur à une borne donnée. Reste alors à calculer les positions des sommets sur chaque couche. Le principe est de les positionner au barycentre de leurs voisins dans la couche précédente (ou suivante). Cependant, les positions des barycentres de deux sommets peuvent être identiques. Pour remédier à ce problème, la méthode *PR* attribue une valeur de priorité aux sommets de L_i en fonction du nombre de voisins qu'ils ont dans la couche L_{i-1} (ou L_{i+1}). D'autre part, les sommets fictifs se voient attribuer une valeur maximale afin d'éviter des points de contrôles inutiles sur les arêtes longues. L'algorithme *PR* positionne les sommets au barycentre de leurs voisins un à un du sommet ayant la plus forte valeur de priorité à celui ayant la plus faible.

L'un des désavantages de la technique de Sugiyama *et al.* [123] est sa complexité en mémoire $O(|V| \cdot [E|)$. Cela est dû au fait que la méthode d'augmentation utilisée pour obtenir une hiérarchie propre peut ajouter $O(|V| \cdot |E|)$ sommets fictifs. Dans [12], Auber présente un algorithme hiérarchique utilisant la technique de Sugiyama *et al.* dont la complexité en espace est $O(|V| + |E|)$. La figure 3.3 illustre la technique de Auber [12]. Etant donné que l'algorithme de Auber [12] n'ajoute que deux sommets fictifs par arête « longue », le nombre de sommets fictifs est $O(2 \cdot [E|)$.

Le second désavantage de l'algorithme de Sugiyama *et al.* est sa complexité en temps $O(|V| \cdot |E| \cdot log(|E|))$. Dans [48], Eiglsperger *et al.* donnent un algorithme permettant de

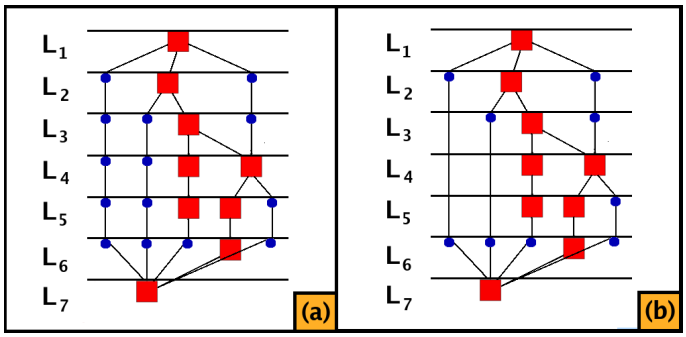

FIG. 3.3: Résultats des techniques d'augmentation utilisées par Sugiyama *et al.* [123] (a) et Auber [12] (b). Les sommets bleus sont les sommets fictifs ajoutés lors de cette étape.

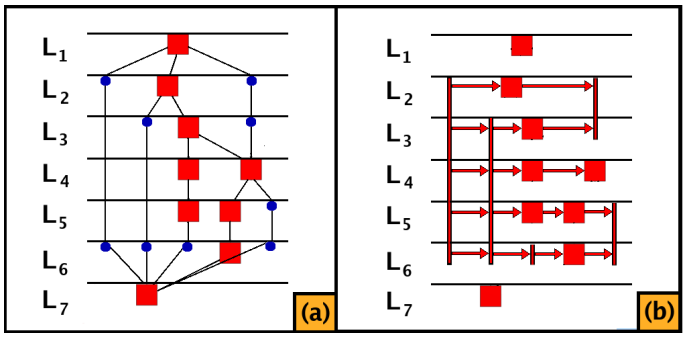

FIG. 3.4: (a) Hiérarchie construite par la méthode de Auber [12] ; (b) graphe de compaction correspondant [48].

réduire cette complexité à $O((|V|+|E|)\cdot log(|E|))$. Pour ce faire, Eiglsperger *et al.* utilisent la méthode de Auber [12] pour construire la hiérarchie propre. Les auteurs introduisent la définition du *graphe de compaction* (cf figure 3.4). L'utilisation du graphe de compaction et d'une méthode de minimisation de croisement d'arêtes couche par couche permet à l'algorithme d'avoir une complexité $O((|V| + |E|) \cdot log(|E|))$.

Dans [93], Kuntz *et al.* donnent une approche différente pour dessiner les DAGs : un algorithme *génétique*. Le principe des algorithmes génétiques est de tout d'abord générer un ensemble de solutions du problème (une *population d'individus*), puis de « croiser » ces individus pour obtenir de meilleures solutions. Partant d'un graphe dont les sommets ont été assignés aux k couches, l'algorithme de Kuntz *et al.* [93] construit tout d'abord une *population* de solutions. Dans l'algorithme de Kuntz *et al.*, une solution définit l'ordre des sommets de chaque couche. L'évolution de cette population après un certain nombre de générations permet d'obtenir une « bonne » solution. Pour ce faire, l'algorithme de Kuntz *et al.* [93] répète un nombre de fois constant les trois étapes suivantes :

1. « Croiser » deux individus
2. Effectuer une mutation
3. Effectuer une recherche locale pour réduire le nombre de croisements d'arêtes

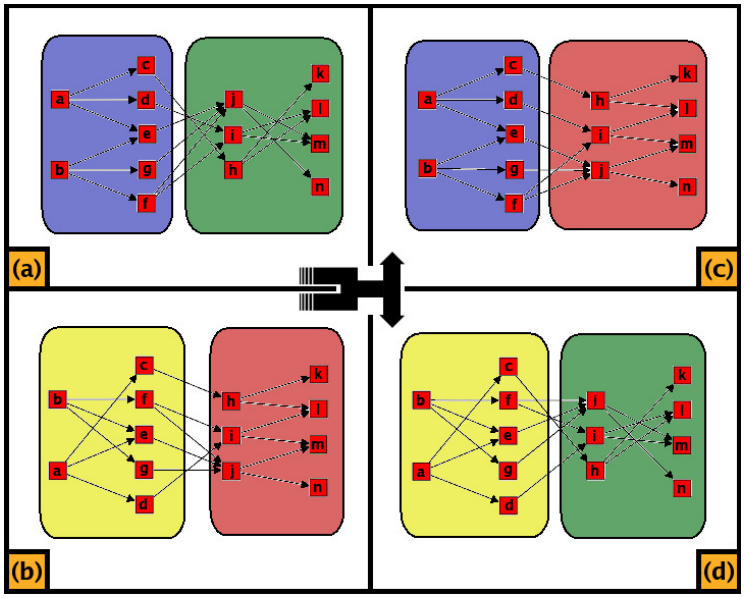

FIG. 3.5: Illustration du croisement *inter-couches* de Kuntz *et al.* [93]. (a) et (b) sont les deux individus parents ; (c) et (d) sont les résultats du croisement inter-couches. L'individu (c) (resp. (d)) est constitué des couches entourées en bleu (resp. entourées en vert) de (a) et des couches entourées en rouge (resp. entourées en jaune) de (b).

Pour « croiser » deux individus, Kuntz *et al.* définissent deux opérations : le croisement *inter-couches* et *intra-couche*. Le croisement de deux individus consiste à effectuer un croisement inter-couches suivi d'un croisement intra-couche. Les figures 3.5 et 3.6 montrent comment sont effectués ces croisements. La mutation, consistant à permuter aléatoirement deux sommets consécutifs d'une même couche, est ensuite effectuée sur les nouveaux individus. Enfin, une recherche locale utilisant des méthodes heuristiques *greedy-switch*, basée sur le barycentre ou sur la médiane est appliquée. Les individus ainsi obtenus sont ensuite ajoutés à la population de solutions et les individus parents en sont retirés.

3.2.3 Algorithmes par analogie physique

Afin de dessiner les graphes généraux, les communautés de dessin de graphe et de visualisation d'information ont mis en place des algorithmes basés sur des simulations de systèmes physiques, appelés *algorithmes par analogie physique*. Cette approche permet d'obtenir des résultats visuellement plaisants mais aussi structurellement significatifs. Brandes explique dans [86] qu'une méthode par analogie physique est composé de deux parties :

- Un *modèle* d'objets physiques représentant les éléments du graphe
- Un algorithme qui essaie de trouver la (une) configuration d'équilibre du modèle

Il existe deux principales méthodes par analogie physique : les modèles de forces et les placements par minimisation du niveau d'énergie. Dans un algorithme par modèle de forces, les sommets sont des objets physiques possédant une masse et éventuellement d'autres caractéristiques (e.g. une température). Les sommets sont considérés dans ce type d'algorithme comme des particules exerÃ§ant une force répulsive sur les autres particules et

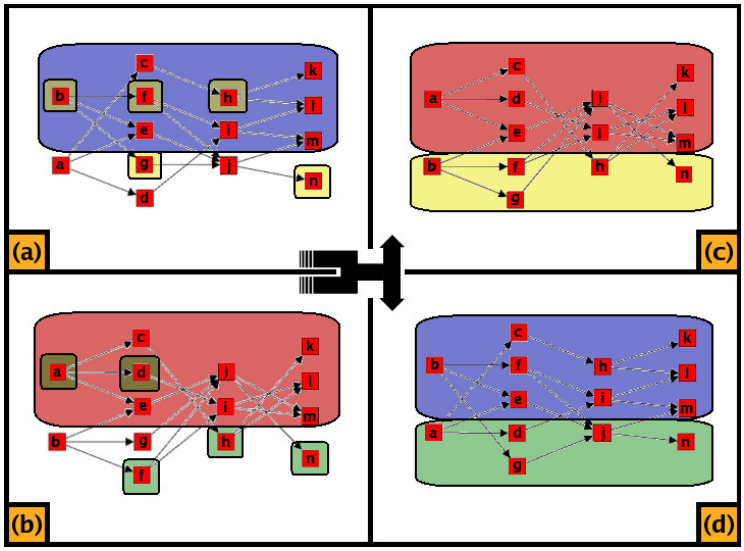

FIG. 3.6: Illustration du croisement *intra-couche* de Kuntz *et al.* [93]. (a) et (b) sont les deux individus parents ; (c) et (d) sont les résultats du croisement intra-couche. L'individu (c) est constitué des sommets de (b) entourés en rouge et des sommets de (a) non-entourés en rouge dans (b) (en jaune dans (a)). Inversement, l'individu (d) est constitué des sommets de (a) entourés en bleu et des sommets de (b) non-entourés en bleu dans (a) (en vert dans (b)).

les liens (arêtes) reliant les sommets sont eux considérés comme des ressorts (e.g. ressorts physiques, électromagnétiques). A chaque itération de ce type d'algorithme, la force totale exercée sur chaque sommet u par les autres sommets est calculée. Les sommets sont ensuite déplacés en fonction de cette force totale. La répétition du calcul des forces et des déplacements permet au système de se rapprocher à chaque itération d'un état plus stable. Cependant cette méthode ne permet pas de garantir une convergence vers l'état le plus stable du système (minimum global du niveau d'énergie). Dans les approches basées sur la minimisation du niveau d'énergie, ce ne sont pas les déplacements des sommets qui font baisser le niveau d'énergie mais plutôt une baisse du niveau d'énergie qui déplace les sommets. En effet, cette technique tente de diminuer le niveau d'énergie et calcule les mouvements de sommets que cela implique.

3.2.3.1 Algorithme par modèle de forces

L'un des algorithmes les plus connus dans ce domaine est celui de Eades [42]. Soit $G = (V, E)$ un graphe non-orienté, on note $p_u = (x_u, y_u)$ le vecteur de position du sommet u dans le plan et $\overrightarrow{p_u p_v}$ le vecteur unitaire de u à v. La force exercée sur un sommet u est composée d'une force d'attraction exercée par les voisins de u dans le graphe et d'une force de répulsion exercée par tous les autres sommets du graphe non voisins de u. La force d'attraction exercée par un voisin v de u est définie comme suit :

$$f_{att}(u,v) = c_{att} \cdot \log \frac{dist_{\mathbb{R}}(u,v)}{length} \cdot \overrightarrow{p_u p_v} \tag{12}$$

où c_{att} est une constante représentant la *raideur* des ressorts et *length* est la longueur au repos du ressort. La force de répulsion exercée par un sommet v (non voisin de u) sur u est elle :

$$f_{rep}(u,v) = \frac{c_{rep}}{dist_{\mathbb{R}}(u,v)^2} \cdot \overrightarrow{p_v p_u} \tag{13}$$

Enfin la force totale exercée sur un sommet u à chaque itération de l'algorithme est :

$$f_{totale}(u) = \sum_{e=\{u,v\}\in E} f_{att}(u,v) + \sum_{e=\{u,v\}\notin E} f_{rep}(u,v) \tag{14}$$

Etant donné que les sommets sont déplacés de manière synchrone (i.e. les forces sont calculées à un moment t et tous les sommets sont ensuite déplacés simultanément), utiliser f_{totale} comme vecteur de déplacement pourrait induire des déplacements excessifs. Afin de limiter les effets de ce problème, le vecteur de déplacement d'un sommet u est défini comme $disp(u) = \delta \cdot f_{totale}(u)$ où $\delta \in [0,1[$ est une constante.

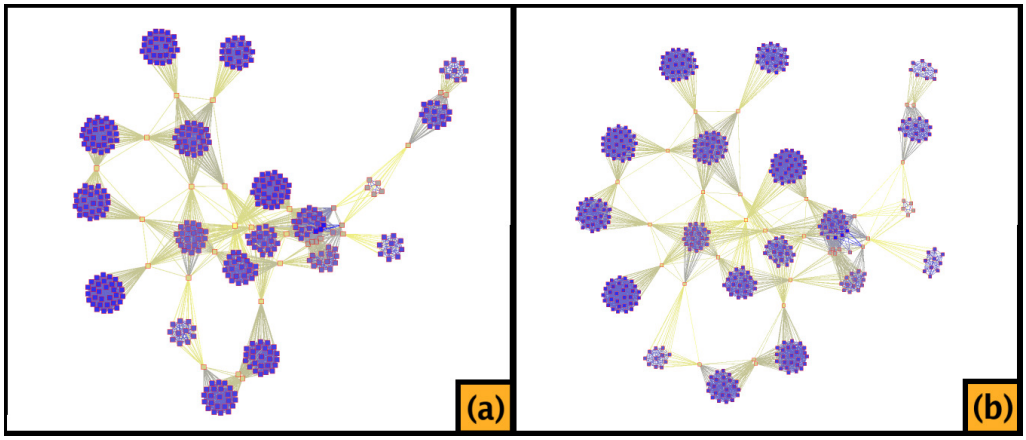

FIG. 3.7: Sous-graphe du « graphe d'Hollywood » où les sommets représentent des acteurs et une arête relie deux sommets dont les acteurs correspondants ont joué dans (au moins) un film commun, (a) dessiné par l'algorithme de Fruchterman et Reingold [55] et (b) par l'algorithme de Frick *et al.* [54].

Dans [55], Fruchterman et Reingold donnent un algorithme par modèle de forces légèrement différent de celui de Eades [42] (cf figure 3.7.(a)). Les modifications apportées aux calculs des forces permettent une convergence plus rapide de l'algorithme. Les formules utilisées pour le calcul de la force attractive exercée sur u par un sommet voisin v est :

$$f_{att}(u,v) = \frac{dist_{\mathbb{R}^2}(u,v)^2}{length} \cdot \overrightarrow{p_u p_v} \tag{15}$$

Et la force répulsive exercée sur u par tout autre sommet v :

$$f_{rep}(u,v) = s \cdot \frac{length^2}{dist_{\mathbb{R}^2}(u,v)} \cdot \overrightarrow{p_v p_u} \tag{16}$$

où s est une constante. L'une des différences principales est que tous les sommets du graphe exercent une force répulsive sur u (alors que seuls les sommets non-voisins de u ne

sont pris en compte dans l'algorithme de Eades [42]). On obtient donc la formule suivante :

$$f_{totale}(u) = \sum_{v \in N(u)} f_{att}(u,v) + \sum_{v \in V, v \neq u} f_{rep}(u,v) \tag{17}$$

L'autre différence importante est que Fruchterman et Reingold proposent une méthode heuristique permettant d'accélérer le temps de calcul de l'algorithme. Etant donnée la manière dont les forces répulsives sont calculées, un sommet éloigné de u n'exerce qu'une très faible force de répulsion sur u. Il n'est donc pas nécessaire de prendre en compte ces sommets lors du calcul des forces. Pour ce faire, cet algorithme utilise une grille contenant les sommets du graphes. Lors du calcul des forces répulsives exercées sur le sommet u, seuls les sommets positionnés dans des cellules voisines de la cellule de v sont pris en compte.

Enfin, contrairement à l'algorithme de Eades [42] où le déplacement est un facteur constant δ de la force totale exercée, dans l'algorithme de Fruchterman et Reingold ce facteur dépend du nombre d'itérations déjà effectuées : l'amplitude des déplacements est plus grande au début de l'exécution de l'algorithme qu'à la fin.

Dans [54], Frick *et al.* présentent un algorithme appelé GEM (cf figure 3.7.(b)). Cet algorithme apporte de nouvelles modifications à l'algorithme de Eades [42]. Tout d'abord, les forces appliquées sur un sommet u par un sommet v sont modifiées en :

$$f_{att}(u,v) = \frac{dist_{\mathbb{R}^2}(u,v)^2}{length \cdot \Phi(u)} \cdot \overrightarrow{p_u p_v} \tag{18}$$

$$f_{rep}(u,v) = \frac{length^2}{dist_{\mathbb{R}^2}(u,v)} \cdot \overrightarrow{p_v p_u} \tag{19}$$

où $\Phi(u)$ est une fonction dépendant du degré de u dans le graphe, $\Phi(u) = 1 + \frac{deg_G(u)}{2}$. De cette manière, les sommets de forts degrés ont des déplacements plus faibles que les sommets de faibles degrés. D'autre part, cet algorithme utilise une force gravitationnelle, exercée depuis le barycentre des sommets :

$$f_{grav}(u) = \Phi(u) \cdot \gamma \cdot \left(\frac{\sum_{v \in V} p_v}{|V|} - p_u \right) \tag{20}$$

où γ est une constante gravitationnelle. Contrairement à la force d'attraction, cette force est proportionnelle à $\Phi(u)$, i.e. plus le sommet est de fort degré plus il est sensible à la gravité. Enfin, une force aléatoire $f_{rand}(u)$ est appliquée à chaque sommet, cela permet à l'algorithme de sortir d'un état stable dont le niveau d'énergie est élevé (i.e. d'un minimum local de l'énergie du système). Toutes ces modifications, notamment des forces proportionnelles (ou inversement proportionnelles) au carré de la distance, permettent une convergence plus rapide de l'algorithme. La force totale exercée sur un sommet u est finalement :

$$f_{totale}(u) = \sum_{v \in N(u)} f_{att}(u,v) + \sum_{v \in V, v \neq u} f_{rep}(u,v) + f_{grav}(u) + f_{rand}(u) \tag{21}$$

La dernière amélioration apportée par GEM [54] porte sur le calcul du vecteur de déplacement. Le facteur δ utilisé pour calculer le vecteur de déplacement en fonction de la force totale est dans cet algorithme propre à chaque sommet u et est fonction du temps t, on le note $\delta_u(t)$. Cela permet d'atténuer les effets d'*oscillation* et de *rotation*. Si un sommet u oscille ou tourne autour d'un point « fixe », alors $\delta_u(t)$ prend une valeur faible. Si au contraire, le sommet u effectue un déplacement au temps t dans le même sens et la même direction que son déplacement au temps $t-1$, alors le facteur $\delta_u(t)$ prend une valeur forte, permettant ainsi un déplacement plus important.

3.2.3.2 Algorithme par minimisation du niveau d'énergie

Dans [79], Kamada et Kawai donnent un algorithme basé sur la minimisation du niveau d'énergie. L'énergie du système est définie comme suit :

$$E = \frac{1}{2} \cdot \sum_{u \in V} \sum_{v \in V, v \neq u} \frac{K}{d_G(u,v)^2} \cdot (dist_{\mathbb{R}}(u,v) - length \cdot d_G(u,v))^2 \tag{22}$$

Pour choisir quel sommet va être déplacé, cet algorithme calcule quel sommet permet de faire diminuer le plus le niveau d'énergie du système. Puis, tant que ce sommet permet de diminuer suffisament (supérieur à une constante ϵ fixée) le niveau d'énergie, l'algorithme déplace le sommet en utilisant le vecteur suivant :

$$f_{totale}(u) = \sum_{v \in V} \left(\frac{dist_{\mathbb{R}^2}(u,v)}{dist_G(u,v) \cdot length^2} - 1 \right) (pos(v) - pos(u)) \tag{23}$$

Ce vecteur de déplacement peut être interprété comme une force exercée sur le sommet u, c'est pourquoi dans la suite de cette thèse nous utiliserons le terme d'algorithme par modèle de forces lorsque l'on parlera de l'algorithme de Kamada et Kawai [79].

3.2.3.3 Complexités

Le tableau 3.1 montre les complexités en temps et en espace de chacun des 4 algorithmes présentés dans cette section.

Algorithme	Temps	Espace	Optimisation
Eades [42]	$O(\|V\|^3)$	$O(\|V\|)$	aucune
Fruchterman & Reingold [55]	$O(\|V\|^3)$	$O(\|V\|)$	par grille
Frick *et al.* [54]	$O(\|V\|^3)$	$O(\|V\|)$	aucune
Kamada & Kawai [79]	$O(\|V\|^4)$	$O(\|V\|)$	aucune
	$O(\|V\|^3)$	$O(\|V\|^2)$	aucune

TAB. 3.1: Complexités en temps et en espace des algorithmes de [42, 55, 54, 79].

Le nombre d'itérations des algorithmes par analogie physique n'est généralement pas borné. Cependant, les auteurs de [42, 55, 54] (ainsi que la communauté de dessin de graphe) considèrent que $O(|V|)$ itérations sont suffisantes pour obtenir un dessin de

« bonne » qualité. Etant donné que chaque itération a un coût de $O(|V|^2 + |E|)$, la complexité en temps de ces algorithmes est de $O(|V|^3)$.

En ce qui concerne l'algorithme de Kamada et Kawai [79], le calcul des forces requiert les distances dans le graphe. Deux solutions sont donc envisageables : calculer et stocker ces distances, ou calculer ces distances à chaque itération de l'algorithme. Calculer la matrice des distances entre chaque paire de sommets peut être évidemment réalisé en temps $O(|V|^3)$ mais nécessite $O(|V|^2)$ espace mémoire [82, 100]. La complexité de l'algorithme est donc en temps $O(|V|^3)$ et en espace $O(|V|^2)$. Une autre possibilité consiste à calculer à chaque itération la distance entre le sommet courant et tous les autres sommets. Le coût en temps de chaque itération est alors $O(|V|^3)$. On obtient dans ce cas une complexité totale $O(|V|^4)$ en temps et $O(|V|)$ en espace.

3.2.4 Approches par échantillonnage et modèle de forces

Les approches classiques par analogie physique donnent de bons résultats qualitatifs, cependant les complexités en temps de ces algorithmes ne permettent pas de les appliquer sur de grands graphes (avec des milliers de sommets et d'arêtes). Récemment, un certain nombre d'algorithmes basés sur des échantillonages et plus performants en temps ont été mis en place. Ces algorithmes réalisent généralement un compromis entre qualité du résultat et temps de calcul.

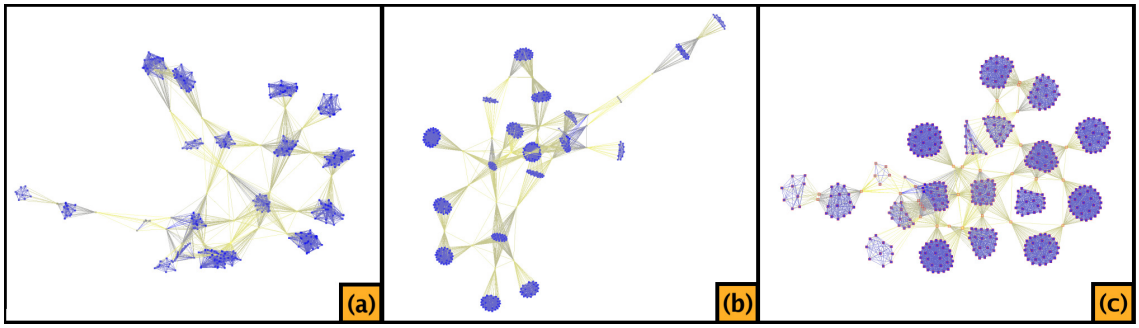

FIG. 3.8: Sous-graphe du « graphe d'Hollywood », (a) dessiné par l'algorithme GRIP [59, 58], (b) par l'algorithme de FM^3 [65] et (c) par LGL [5].

3.2.4.1 Algorithme multi-échelles

Dans [59, 58], Gajer *et al.* présentent un algorithme multi-niveaux pour dessiner les grands graphes appelé GRIP (cf figure 3.8.(a)). La première étape de cet algorithme consiste à calculer une hiérarchie de représentants. Cette hiérarchie de représentants est appelée *échantillonnage par ensembles indépendants maximaux* (noté MISF, pour *Maximal Independant Set Filtration*). Soit $\mathcal{V}= \{V_i\}_{0\leq i\leq k}$ tel que $V = V_0$ un MISF du graphe $G = (V, E)$, on a $V = V_0 \supset V_1 \supset ... \supset V_k$ et $\forall u, v \in V_i$, $dist_G(u, v) \geq 2^i$ et $|V_k| = 3$. L'idée est de prendre un ensemble de sommets à distance 2 comme représentants de l'ensemble des sommets, puis des sommets à distance 4 pour représenter les sommets à distance 2,

etc. Ce type d'échantillonnage permet d'obtenir une bonne répartition des représentants dans le graphe. Une technique simple pour calculer un MISF consiste à répéter pour tout i de 1 à k les étapes suivantes :

1. Soit $V^* = V_{i-1}$
2. Choisir aléatoirement un sommet u de V^*.
3. Enlever le sommet u de V^* ainsi que tout les sommets de V^* à distance au plus $2^i - 1$, et ajouter u à V_i.
4. Revenir à l'étape 2 tant qu'il reste des sommets dans V^*, sinon incrémenter i et revenir à l'étape 1.

La deuxième étape consiste à calculer les positions des sommets du MISF du plus haut échantillon V_k au plus bas V_0. Les sommets de V_k sont positionnés sur un triangle de telle sorte que les distances dans le dessin respectent au mieux les distances dans le graphe. Pour tout autre V_i, $k > i > 0$, les sommets de $V_i \setminus V_{i+1}$ sont dessinés en utilisant l'algorithme de Kamada et Kawai [79]. Enfin, les sommets de $V_0 \setminus V_1$ sont positionnés en utilisant l'algorithme de Fruchterman et Reingold [55]. Pour plus de détails sur cet algorithme, le lecteur peut se référer au chapitre 5.

Dans [65], Hachul et Jünger présentent un autre algorithme, appelé FM^3 (cf figure 3.8.(b)). L'algorithme FM^3 est le premier algorithme multi-échelles par modèle de forces dont la complexité en temps est prouvée : $O(|V| \cdot log(|V|) + |E|)$. La méthode d'échantillonnage utilisée est basée sur une analogie avec des systèmes solaires puisqu'elle consiste à chercher des étoiles, les planètes de ces étoiles et les lunes de ces planètes. Soit s une étoile, les planètes associées à s sont les voisins de s dans le graphe. Soit p une planète, les lunes associées à p sont les voisins de p dans le graphe (à l'exclusion de l'étoile associée à p). Pour trouver cet ensemble de systèmes solaires, l'algorithme fonctionne comme suit :

1. Soit $V^* = V$
2. Tirer un sommet s aléatoirement dans V^*
3. Retirer de V^* le sommet s ainsi que l'ensemble des sommets qui sont à une distance dans le graphe inférieure ou égale à 2 de s
4. Retourner à l'étape 2 tant qu'il reste des sommets dans V^*

On obtient ainsi un ensemble d'étoiles s et les sommets à distance au plus 2 de s qui leur sont associés. Pour chaque étoile s, les voisins de s sont marqués comme étant des planètes et les voisins des planètes (à l'exclusion de l'étoile et des autres planètes) sont marqués comme étant des lunes. Chaque système solaire est ensuite remplacé par un métanoeud pour donner le premier niveau de l'échantillonnage, G_1. Ce processus est répété sur G_i pour obtenir le graphe G_{i+1} et s'arrête lorsque le graphe G_k contient un nombre constant de sommets. L'algorithme positionne ensuite chacun des graphes G_i pour i allant de k à 0. Les sommets du graphe G_k sont tout d'abord placés en utilisant un algorithme par modèle de forces. Puis pour chaque G_i, chaque étoile s de G_i est positionnée aux coordonnées du sommet représentant le système solaire de s dans G_{i+1} et les planètes et lunes positionnées à distance i et $2 \cdot i$ de s (i.e. si $i = 1$, les planètes sont positionnées à distance 1 de s et les lunes à distance 2). Les positions des sommets de G_i sont ensuite affinées par un algorithme par modèle de forces.

3.2.4.2 Algorithme par extraction d'arbre couvrant

L'algorithme LGL présenté dans [5], inspiré de [31], a été mis en place pour visualiser de grands réseaux de protéines (cf figure 3.8.(c)). Etant donné que ces réseaux sont généralement arête-valués, cet algorithme permet de prendre en compte la valuation des arêtes. Tout d'abord, LGL calcule un arbre couvrant de poids minimal en utilisant l'algorithme de Kruskal [91]. La seconde étape consiste à trouver une racine u_{root} dans l'arbre couvrant. Pour ce faire, plusieurs méthodes sont envisageables : aléatoire, basée sur la centralité ou encore désignée par l'utilisateur. L'algorithme construit ainsi un arbre couvrant enraciné en u_{root}. Cet arbre est ensuite utilisé pour connaître l'ordre d'insertion des sommets dans le dessin et permet donc l'échantillonnage des sommets. Partant de la racine, l'algorithme dessine un à un les niveaux de l'arbre. Ayant dessiné le niveau i de l'arbre couvrant, l'algorithme place les sommets du niveau $i+1$ sur un cercle dont le rayon est déterminé par la position du centre de gravité et la distance entre les sommets du niveau $i-1$ et les sommets du niveau i. Les positions des sommets du niveau $i+1$ sont ensuite affinées en utilisant un algorithme par modèle de forces. Afin d'accélérer le processus, seuls les sommets du niveau $i+1$ sont déplacés par l'algorithme par modèle de forces. D'autre part, une technique de grille analogue à celle de Fruchterman et Reingold [55] est utilisée pour savoir quels sommets prendre en compte lors du calcul des forces répulsives.

3.2.4.3 Complexités

Algorithme	Temps	Espace	Méthode d'échantillonnage
GRIP [59, 58]	n.c.	n.c.	MISF
FM^3 [65]	$O(\|V\| \cdot log(\|V\|) + \|E\|)$	n.c.	analogie aux systèmes solaires
LGL [5]	n.c.	n.c.	arbre couvrant

Tab. 3.2: Complexités en temps et en espace des algorithmes de [59, 58, 65, 5].

Le tableau 3.2 montre les complexités en temps et en espace de chacun des 3 algorithmes par échantillonnage présentés dans cette section.

Comme mentionné ci-dessus, seule la complexité de l'algorithme FM^3 [65] a été prouvée. En ce qui concerne l'algorithme GRIP [59, 58], la complexité totale de l'algorithme n'a pas été prouvée cependant la complexité de l'étape de placement est $O(|V| \cdot log^2(\delta(G)))$ en temps et $O(|V| \cdot log(\delta(G)))$ en espace, où $\delta(G)$ est le diamètre du graphe. Enfin, la complexité totale de l'algorithme LGL [5] n'est pas prouvée. On connaît en revanche la complexité de l'algorithme de Kruskal [91], $O(|E| \cdot log(|V|)$ et la complexité de l'algorithme par modèle de forces utilisés, $O(|V|^2)$.

3.2.5 Approches basées sur les matrices

Une approche récente consiste à représenter un graphe sous forme d'une matrice et à réduire le nombre de dimensions (généralement, jusqu'à 2 ou 3 dimensions) via des techniques d'algèbre linéaire pour dessiner le graphe. Cette approche a l'avantage d'offrir des temps de calcul très rapides, cependant la qualité des résultats obtenus dépend de la

structure topologique du graphe, en particulier ces algorithmes ne donnent pas de très bons résultats sur les graphes contenant de nombreuses composantes biconnexes [66]. Dans cette partie, nous présentons deux approches basées sur l'utilisation du calcul matriciel.

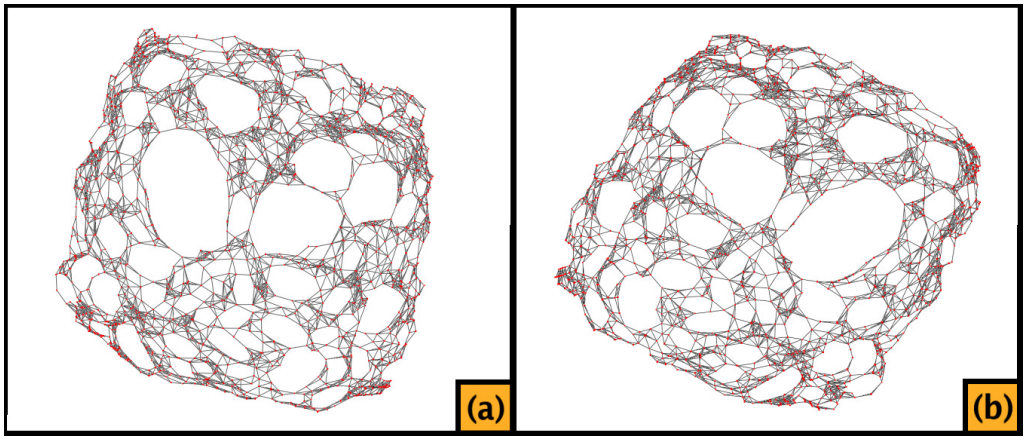

FIG. 3.9: Exemple de graphe sans-échelle contenant 2000 sommets et 9646 arêtes, (a) dessiné par l'algorithme HDE [68, 69] et (b) par l'algorithme de ACE [89].

Dans [68, 69], Harel et Koren donnent un algorithme, appelé HDE, permettant de dessiner un graphe en 2 ou 3 dimensions (cf figure 3.9.(a)). Cet algorithme consiste à tout d'abord calculer un plongement du graphe en m dimensions (les auteurs conseillent $m = 50$), puis il projette ce plongement sur 2 ou 3 dimensions. Pour définir ces m axes (un pour chaque dimension), l'algorithme calcule un ensemble de m *pivots* $\{p_1, p_2, ..., p_m\}$. L'axe i correspondant au pivot p_i représente le « point de vue » de p_i sur le graphe. Dans cet espace à m dimensions, la $i^{ème}$ coordonnée d'un sommet u est la distance dans le graphe entre le sommet u et le sommet p_i. Supposons que le graphe contienne n sommets $u_1, u_2, ..., u_n$, la matrice ainsi construite est donc une matrice de distances dans le graphe définie comme suit :

$$\begin{pmatrix} d_G(u_1, p_1) & ... & d_G(u_n, p_1) \\ . & ... & . \\ d_G(u_1, p_m) & ... & d_G(u_1, p_m) \end{pmatrix} \tag{24}$$

où $d_G(u, v)$ est la distance dans le graphe entre les sommets u et v. Afin d'avoir un plongement en m-dimensions correct, les pivots doivent être bien répartis dans le graphe (distribution uniforme des pivots dans le graphe). Pour obtenir un tel ensemble de pivots, l'algorithme fonctionne comme suit : le pivot p_1 est choisi aléatoirement, puis pour tout $1 < i \leq m$, p_i est le sommet qui maximise la distance avec les autres pivots déjà élus $p_1, p_2, ..., p_{i-1}$. Harel et Koren utilisent ensuite une méthode classique de réduction de dimension appelée *Principal Component Analysis* (pour plus de détails sur cette technique, le lecteur peut se référer à [49]). Cette méthode permet de trouver les dimensions (en pratique 2 ou 3) les plus « anti-corrélées » afin d'obtenir un bon plongement. La complexité en temps de cet algorithme est $O(m \cdot (|V| + |E|))$ dans le cas non-valué et $O(m \cdot (|V| \cdot log(|V|) + |E|))$ dans le cas arête-valué.

Dans [89], Koren *et al.* donnent une autre approche, appelée ACE, basée sur les matrices : cette approche utilise les vecteurs propres de la matrice Laplacienne pour calculer une projection en 2 ou 3 (ou plus) dimensions (cf figure 3.9.(b)). La matrice Laplacienne L d'un graphe est définie comme suit :

$$L = D - A \tag{25}$$

où D est la matrice des degrés, i.e. chaque $d_{i,i} = deg_G(i)$ et pour tout $i \neq j$, $d_{i,j} = 0$, et A est la matrice d'adjacence du graphe. Le principe de cette approche est d'estimer les vecteurs propres de la matrice Laplacienne originale en construisant une hiérarchie de matrices. La matrice de plus haut niveau doit être suffisamment « petite » pour pouvoir en calculer les vecteurs propres efficacement. Connaissant les vecteurs propres de la matrice de niveau i, les auteurs calculent successivement des estimations des vecteurs propres de la matrice $i - 1$, jusqu'à obtenir une estimation des vecteurs propres de la matrice Laplacienne originale. Puis, pour dessiner le graphe en d dimensions, les auteurs utilisent ensuite les d vecteurs propres ayant les plus petites valeurs propres.

3.2.6 Approches basées sur le partitionnement

Une autre approche utilisée pour dessiner de très grands graphes (i.e. contenant des dizaines de milliers de sommets et arêtes) est basée sur la détection de sous-structures. Ces sous-structures doivent par ailleurs avoir des caractéristiques topologiques particulières (e.g. arbres, composantes biconnexes).

Dans [122], Six et Tollis présentent un algorithme de dessin permettant de mettre en évidence les composantes biconnexes d'un graphe. La première étape de cet algorithme consiste donc à décomposer le graphe en composantes biconnexes. Puis l'algorithme construit un graphe tel qu'à chaque composante biconnexe et à chaque *articulation* [4] corresponde un sommet dans ce graphe. Deux sommets sont reliés par une arête si les sommets correspondants dans le graphe original sont reliés par (au moins) une arête. Etant donné que la décomposition effectuée est une décomposition en composantes biconnexes, ce graphe ne peut contenir de cycle (orienté ou non) et est par conséquent un arbre libre dont les sommets représentent soit une composante biconnexe soit un sommet d'articulation. Chaque composante biconnexe est ensuite dessinée par un algorithme circulaire et le graphe est lui dessiné en utilisant un algorithme de dessin d'arbre, plus précisément un algorithme de dessin radial [76].

Dans SPF [8], Archambault *et al.* reprennent l'idée de Six et Tollis [122]. Après détection des composantes biconnexes, l'algorithme utilise des algorithmes différents pour dessiner chaque composante biconnexe ainsi que l'arbre des composantes biconnexes. Tout d'abord, pour dessiner chaque composante biconnexe, SPF utilise une version modifiée de LGL [5], puis l'arbre des composantes biconnexes est dessiné en utilisant l'algorithme de dessin d'arbre RINGS [125].

[4] Une articulation est un sommet partagé par plusieurs composantes biconnexes

Dans [60], Gansner *et al.* présentent une technique pour visualiser de grands graphes arête-valués. Partant d'un dessin de ce graphe, ils construisent un arbre de hiérarchie. Afin d'obtenir une « bonne » abstraction du graphe, l'algorithme de partitionnement multi-échelles que les auteurs utilisent intègre deux contraintes majeures. Tout d'abord, la taille des différentes groupes doit être « approximativement » identique, la seconde contrainte est que la distance entre toute paire de sommets d'un groupe doit être faible. Une fois ce partitionnement hiérarchique effectué, le graphe quotient Q_1 correspondant [5] est dessiné. Un interacteur permet ensuite de désigner un ou plusieurs centres d'intérêt dans ce graphe. Les zones du graphe proches du (ou des) centres d'intérêt sont affichées de manière détaillée (les plus bas niveaux de la hiérarchie) tandis que les zones éloignées sont représentées de manière très abstraite (les plus hauts niveaux de la hiérarchie).

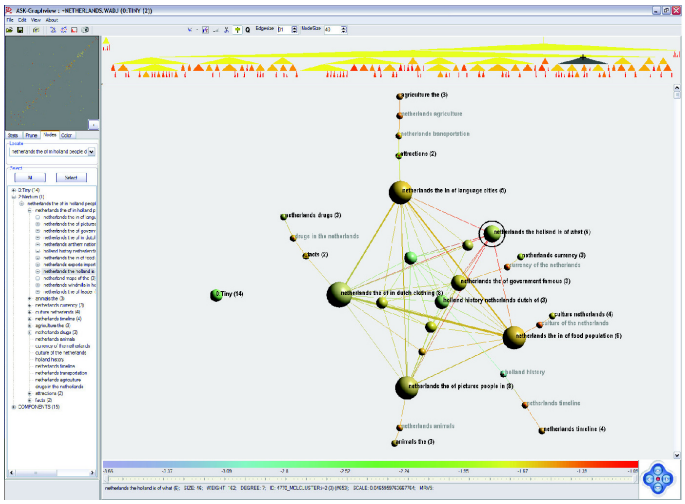

FIG. 3.10: Capture d'écran du système de visualisation présenté par Abello *et al.* dans [4].

Dans [4], Abello *et al.* présentent un système de visualisation de grands graphes (cf figure 3.10). L'algorithme utilisé recherche tout d'abord la *forêt*[6] périphérique en utilisant une méthode appelée *peeling*. Cette méthode consiste à supprimer les sommets de degré 1 et à répéter ce processus tant qu'il reste de tels sommets [7]. Le sous-groupe induit par les sommets supprimés forment la *forêt périphérique*. Soit r_T un sommet du graphe auquel est relié (au moins) un arbre de la forêt, tous les arbres reliés à r_T ainsi que r_T sont remplacés par un métanoeud. L'étape suivante consiste à rechercher les composantes biconnexes du graphe résultant et à les remplacer par des métanoeuds. Le principe développé dans [4] est qu'il faut limiter le nombre d'éléments affichés pour augmenter la lisibilité de la visualisation. Par conséquent, si certaines composantes biconnexes sont de taille trop importante (pour plus de détail voir [4]), les auteurs appliquent alors l'algorithme MCL de [126] pour les décomposer. Cet algorithme simule une marche aléatoire, l'idée étant que les arêtes intra-groupes sont visitées plus « souvent » par cette marche aléatoire que les arêtes inter-groupes.

[5] i.e. le graphe quotient correspondant au premier niveau de la hiérarchie

[6] cf définition dans la section 2.2

[7] Cette forêt périphérique correspond aux sommets de *coreness* 1 du graphe

Le graphe quotient ainsi obtenu est dessiné par un algorithme par modèle de forces. Lorsque l'utilisateur « demande » plus de détails et donc plus d'informations, il « ouvre » un métanoeud et le sous-graphe qu'il représente est lui aussi dessiné par l'algorithme par modèle de forces. Etant donné que la taille des groupes ne doit pas dépasser un certain seuil, le calcul du dessin peut être effectué en un temps permettant une interaction efficace.

Le travail de Archambault *et al.* [7], appelé Topolayout a largement inspiré celui de Abello *et al.* [4]. Archambault *et al.* présentent dans [10] (version longue de [7]) un système de visualisation de graphe basé sur la recherche des structures suivantes :

1. Détection de la forêt périphérique
2. Détection des composantes biconnexes
3. Détection de sous-graphes pouvant être « correctement » dessinés par l'algorithme HDE [68] (pour plus de détails voir [10]), dits *« HDE suitable »*
4. Détection de cliques
5. Algorithme Strength de Auber *et al.* [13]

Ce processus de recherche est appliqué récursivement sur chaque groupe, donnant ainsi une partition multi-échelles de structures topologiques. Ensuite, à chacune de ces structures correspond un algorithme particulier. Les arbres de la forêt périphérique sont soit dessinés par l'algorithme *Bubble Tree* [63] soit par l'algorithme *Walker* [26]. Les composantes biconnexes sont représentées en utilisant une méthode similaire à celle de Six et Tollis [122]. Pour dessiner les cliques, Topolayout utilise un algorithme circulaire. Les groupes *HDE suitable* sont évidemment dessinés par l'algorithme HDE [68, 69]. Enfin, les groupes trouvés par Strength et les groupes de topologie inconnue sont dessinés par une version modifiée de GEM [54]. Finalement, Archambault *et al.* utilisent une heuristique pour réduire le nombre de croisements d'arêtes basée sur la rotation des groupes trouvés.

3.3 Visualisation de graphe partitionné

Les techniques de partitionnement de graphe (cf section 3.1) permettent de regrouper les éléments similaires dans un groupe unique. Dans cette section nous présentons les principales approches et techniques utilisées en visualisation d'information pour représenter des partitions multi-échelles de graphes. Représenter « correctement » une partition de graphe consiste à offrir une visualisation dans laquelle chaque groupe de la partition est visuellement identifiable.

L'un des premiers articles traitant cette problématique est celui de Eades et Feng [43]. Dans cet article, les auteurs introduisent une définition de partitionnement de graphe (définition reprise et généralisée dans la partie 2.2). D'autre part, ils donnent deux représentations différentes de graphe partitionné, appelées *représentation plane* et *représentation multi-niveaux*.

La représentation plane, ou *dessin plan*, consiste à dessiner les sommets et les arêtes dans un plan. Dans ce type de représentation, un sommet u du niveau i de l'arbre de partition est représenté par une région R, de plus :

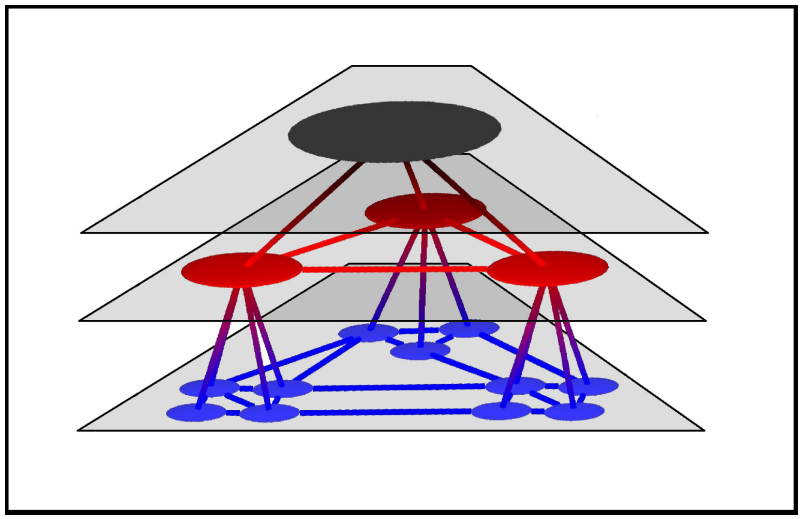

FIG. 3.11: *Représentation multi-niveaux* de Eades et Feng [43] : chaque plan représente un niveau de la hiérarchie.

- les régions des fils de u dans l'arbre sont contenues dans la région R
- les régions des autres sommets de niveau i sont à l'extérieur de la région R
- si une arête e du graphe relie deux fils de u dans l'arbre de partition, alors e est contenue dans la région R

Dans [43], les auteurs donnent aussi deux algorithmes de représentation plane : un algorithme de dessin en lignes droites et un algorithme de dessin orthogonal. Ces deux approches respectivement développées dans [44] et [45] consistent à utiliser des algorithmes de dessin connus pour représenter les graphes partitionnés.

Quant à la représentation multi-niveaux, c'est une séquence de dessins dans le plan, un pour chaque graphe quotient de la hiérarchie. Le graphe partitionné est tout d'abord dessiné par un algorithme de dessin plan. Puis les sommets de chaque niveau de l'arbre de partition sont positionnés au barycentre de leurs fils dans l'arbre (voir la figure 3.11).

Dans [129, 74, 46], les auteurs donnent des approches par modèle de forces pour visualiser les graphes partitionnés.

Dans [129], la force exercée sur un sommet u est une combinaison de trois vecteurs de forces : la force exercée par les autres sommets du groupe de u (force intra-groupe), la force exercée par les sommets qui n'appartiennent pas au groupe de u (force inter-groupes) et enfin la force exercée par les autres groupes sur le groupe de u (méta-force). On a donc :

$$\overrightarrow{F(u)} = \overrightarrow{F_{intra}}(u) + S(t)\overrightarrow{F_{inter}}(u) + (1 - S(t))\overrightarrow{F_{meta}}(u) \tag{26}$$

où t indique le nombre d'itérations effectuées et $S(t)$ est une fonction décroissante entre 1 et 0. L'exécution de cet algorithme se fait en trois intervalles (cf figure 3.12) :

- $t \in [0..t']$: période stable où $S(t) = 1$,
- $t \in [t' + 1..t'']$: période de transition où $S(t)$ décroît de 1 à 0,
- $t \geq t'' + 1$: période stable où $S(t) = 0$

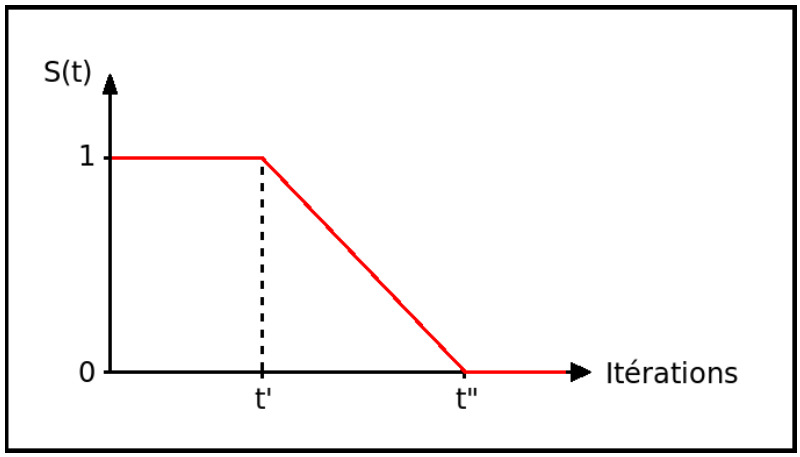

FIG. 3.12: Evolution de $S(t)$ en fonction t. La partie transitoire entre t' et t'' est donnée à titre d'exemple.

Faire varier l'importance des forces inter-groupes et des méta-forces permet d'obtenir des dessins esthétiquement plaisants. D'autre part, les auteurs expliquent que l'ajout d'un moteur de résolution de contraintes permet d'améliorer la qualité du dessin.

Dans [74, 46], les auteurs donnent une autre approche basée sur un algorithme par modèle de forces. Un sommet virtuel est ajouté dans chaque groupe du partitionnement et une arête virtuelle est ajoutée entre ce sommet virtuel et chaque sommet « réel » du groupe. Ils décrivent donc un modèle où l'on trouve trois types d'interactions : les interactions intra-groupes, les interactions inter-groupes, et les interactions virtuelles. Ces interactions virtuelles permettent de conserver la cohésion des groupes. Dans ces articles, les auteurs présentent par ailleurs une méthode de navigation et une visualisation interactive dans une hiérarchie de groupes appelée DA-TU.

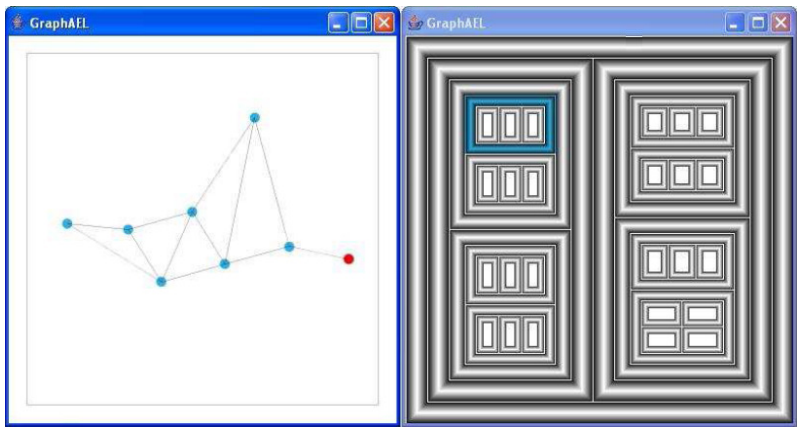

FIG. 3.13: Capture d'écran de l'application présentée dans [3]. Sur la gauche, la vue du graphe quotient et sur la droite la vue sur l'arbre hiérarchie.

Dans [3], Abello *et al.* présentent un système de vues multiples pour visualiser de grands graphes partitionnés (voir la figure 3.13). Ce système utilise une vue du graphe

quotient (d'un niveau donné) et une vue de l'arbre de hiérarchie. Ils décrivent d'autre part un système d'interactions sur ces deux vues qui consiste à *ouvrir* et *fermer* un métanoeud du graphe quotient pour obtenir plus de détails dans la vue du graphe quotient. Lorsque l'utilisateur « ouvre » un métanoeud (et donc demande du détail sur une partie du graphe quotient), les métanoeuds les plus éloignés dans le graphe se « ferment » afin de conserver une représentation lisible. La combinaison de cet interacteur et de ces deux vues permet de conserver le contexte global tout en ayant une information locale.

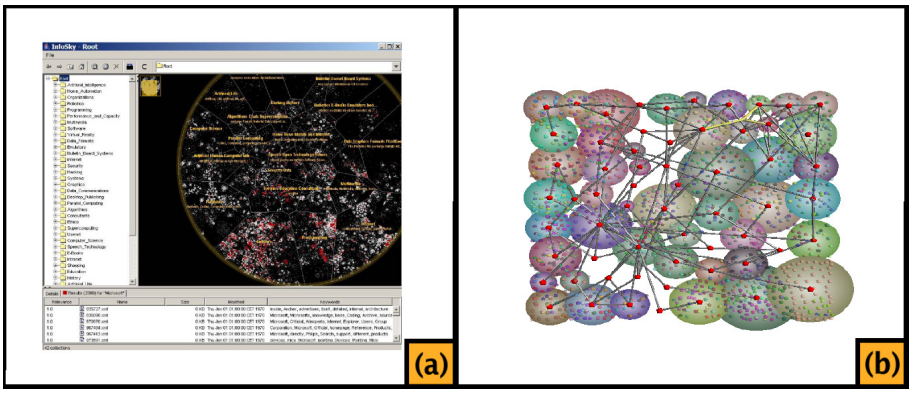

FIG. 3.14: (a) Capture d'écran du système de visualisation présenté par Granitzer *et al.* dans [62] ; (b) Visualisation de données financières par la méthode de Kumar et Garland [92].

Dans [62, 92], les auteurs présentent des algorithmes par modèle de forces permettant de prendre en compte les valuations des arêtes (cf figure 3.14). Ces algorithmes utilisent une stratégie multi-échelles descendante qui consiste à dessiner le premier niveau de la hiérarchie puis à descendre progressivement. Après avoir dessiné le graphe quotient correspondant au niveau i de la hiérarchie (en utilisant un algorithme par modèle de forces), ces algorithmes calculent les diagrammes de Voronoï[8] des positions des noeuds (ou métanoeuds) de ce niveau. Pour un métanoeud donné v du niveau i, les fils de v dans l'arbre de partition sont dessinés dans la cellule de Voronoï de v.

Dans [62], lorsque le graphe Q_i est dessiné (i.e. graphe quotient du niveau i de la hiérarchie, cf section 2.2) et les cellules de Voronoï calculées, l'algorithme effectue le placement des sommets de Q_{i+1} (i.e. le niveau $i+1$ de la hiérarchie) en utilisant les positions et les cellules des sommets de Q_i. Les sommets de Q_{i+1} sont initialement positionnés aux coordonnées de leurs représentants dans Q_i. Soit v un sommet de Q_i et S l'ensemble des sommets fils de v dans la hiérarchie. Le sous-graphe induit par S dans Q_{i+1}, noté $Q_{i+1}[S]$, est dessiné en utilisant un algorithme par modèle de forces. Pour obtenir un dessin final esthétiquement plaisant, l'algorithme par modèle de forces prend en compte les voisins de S dans Q_{i+1} lors du calcul des forces exercées sur les sommets de S. Enfin, Granitzer *et al.* appliquent une simple mise à l'échelle des positions des sommets de S pour contraindre ces sommets dans la cellule de v. Ce processus est répété pour tout sommet v de Q_i et donc jusqu'à ce que le graphe Q_{i+1} soit dessiné. Cette approche offre de bons résultats mais compte tout de même un inconvénient majeur. Si les sommets ne sont pas contraints dans leur cellule lors de l'exécution de l'algorithme par modèle de forces, après un certain

[8] cf définition dans la section 2.4

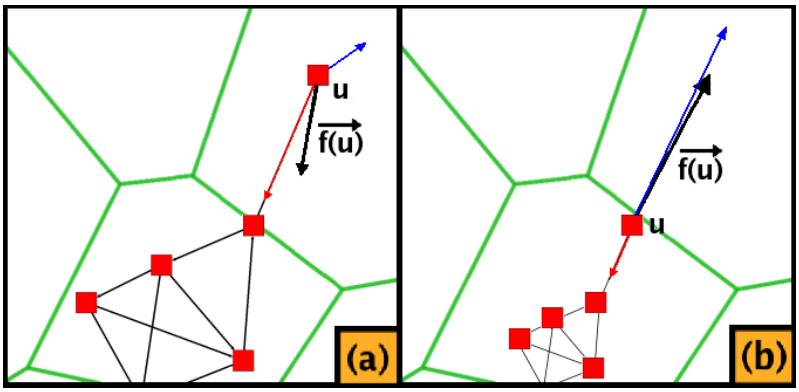

FIG. 3.15: Les flèches rouges et bleues représentent respectivement les forces d'attraction et de répulsion exercées sur le sommet u ; (a) dans le cas où u n'est pas contraint dans sa cellule (en vert) ; (b) dans le cas où u est contraint dans sa cellule. On remarque que ces forces n'ont ni la même intensité, ni le même sens, ni la même direction.

nombre d'itérations, certaines interactions entre des sommets de $Q_{i+1}[S]$ peuvent devenir « faibles » (e.g. les distances entre ces sommets sont trop grandes). La figure 3.15 illustre cet inconvénient, dans la figure 3.15.(a) la force totale exercée sur le sommet u est différente de celle exercée dans figure 3.15.(b), c'est-à-dire lorsque le graphe $Q_{i+1}[S]$ est mis à l'échelle de la cellule.

Dans [92], Kumar et Garland utilisent un processus similaire. La différence principale tient au fait qu'ils essaient de diminuer la complexité en temps de l'algorithme. Pour ce faire, lorsqu'ils calculent le dessin de $Q_{i+1}[S]$, ils ne prennent en compte aucune arête reliant un sommet de S à un sommet de $V_{Q_{i+1}} \setminus S$. Etant donné que pour un niveau de la hiérarchie, aucune arête *inter-groupes* n'est prise en compte, cela conduit à un dessin contenant des arêtes « longues » et à un certain nombre de croisements d'arêtes inutiles.

Dans [9], Archambault *et al.* présentent un système de représentation et de navigation interactif appelé Grouse (cf figure 3.16). Grouse permet de visualiser le résultat de l'algorithme de partitionnement présenté dans [7, 10], c'est-à-dire un algorithme détectant certaines structures topologiques (e.g. les sous-arbres, les composantes biconnexes). Tout comme dans [7, 8, 10], à chaque structure topologique correspond dans Grouse un algorithme de dessin adapté. L'idée principale de cette approche est de dessiner chaque sous-graphe à la demande. Au chargement du graphe partitionné, le graphe quotient Q_1 (i.e. correspondant au premier niveau de la hiérarchie) est dessiné. Lorsque l'utilisateur demande l'« ouverture » d'un métanoeud, Grouse utilise l'algorithme approprié pour dessiner le sous-graphe correspondant et utilise un système d'animation pour rendre plus aisée la compréhension des modifications engendrées. L'un des aspects importants est que si le sous-graphe correspondant à un métanoeud n'a pas encore été dessiné, la taille de ce métanoeud est estimée en fonction du nombre de sommets qu'il représente. Si au contraire, le métanoeud a déjà été « ouvert » et le sous-graphe qu'il représente dessiné, lors de la « fermeture » de ce métanoeud, sa taille sera fixée à l'espace nécessaire pour dessiner le sous-graphe. Dans GrouseFlocks [11], Archambault *et al.* ont ajouté quelques fonctionnalités à Grouse notamment la possibilité de modifier l'arbre de hiérarchie et donc de naviguer

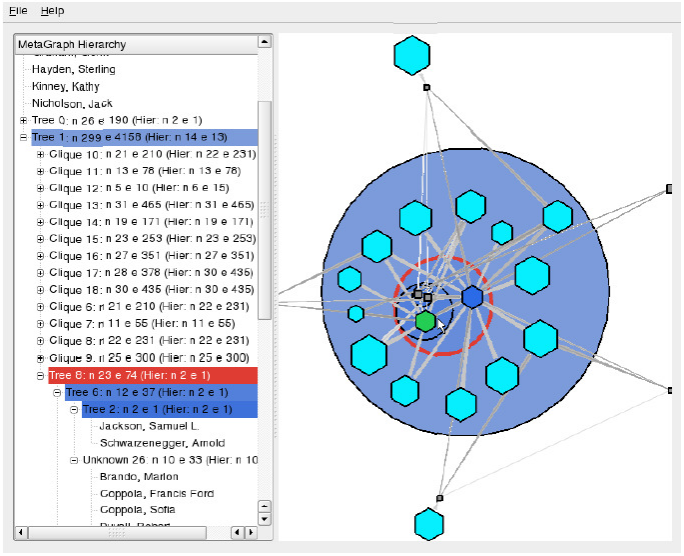

FIG. 3.16: Interface du système de visualisation de graphe partitionné, Grouse [9]. La partie gauche montre l'arbre de partition et la vue de droite est une vue du graphe quotient dont certains métanoeuds ont été « ouverts ».

non seulement dans la hiérarchie mais aussi dans l'espace des hiérarchies possibles sur un graphe.

3.4 Visualisation de données biologiques : Cas particulier du métabolisme

L'un des aspects de cette thèse porte sur la visualisation de métabolisme. Comme décrit dans la section 2.3, le réseau métabolique d'un organisme peut être décrit par un ensemble de voies métaboliques lui permettant de réaliser certaines fonctions biologiques particulières. Il existe donc plusieurs niveaux d'analyse de ces réseaux : du plus simple (quelques réactions), au plus complexe (le réseau métabolique entier) en passant par un niveau intermédiaire (les voies métaboliques). Quel que soit le niveau de l'étude, les outils de visualisation pour la biologie doivent respecter un certains nombre de contraintes de représentation. Certaines de ces contraintes proviennent du dessin de graphe comme la limitation du nombre de croisement d'arêtes ou encore la maximisation de la résolution angulaire. D'autres contraintes, fixées par les biologistes, portent sur la représentation de sous-structures topologiques particulières (tels que les cycles ou les cascades de réactions). Enfin, dans le cas de la visualisation de réseaux métaboliques, il est important que les biologistes puissent aisément identifier les groupes de sommets réalisant une fonction biologique particulière (i.e. les voies métaboliques), les composés et réactions d'une même voie doivent donc être dessinés dans une même région du dessin. Nous présentons dans cette section les principaux travaux sur la visualisation de voies et de réseaux métaboliques.

3.4.1 Visualisation de voies métaboliques

Etant donnée l'organisation hiérarchique des voies métaboliques, les premiers systèmes de visualisation de voies métaboliques présentés dans cette partie sont basés sur l'utilisation d'algorithmes de dessin hiérarchiques (cf partie 3.2.2).

Dans [121], Sirava *et al.* présentent un logiciel d'exploration et de visualisation de voies métaboliques appelé *BioMiner*. Chaque voie métabolique est dessinée en utilisant l'algorithme de dessin hiérarchique de Sugiyama *et al.* [123] implémenté dans le logiciel de visualisation yFiles [133].

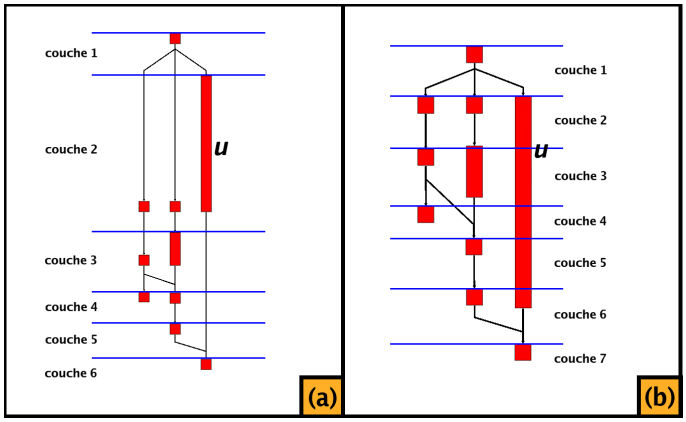

FIG. 3.17: Algorithmes hiérarchiques ; (a) Dans l'algorithme de Sugiyama *et al.* [123], chaque sommet est sur une unique couche, e.g. le sommet u ; (b) dans la version de Schreiber, le sommet u est sur 5 couches.

Dans [115], Schreiber présente l'algorithme de dessin de voies métaboliques utilisé dans *BIOPATH* [23]. Cet algorithme est une version modifiée de l'algorithme de Sugiyama *et al.* [123]. Les deux principales modifications apportées à l'algorithme de Sugiyama *et al.* [123] sont tout d'abord une meilleure prise en compte des tailles des sommets. Dans l'algorithme de Sugiyama *et al.* [123], l'espace occupé par chaque couche est donné par la taille du sommet le plus grand de la couche. La figure 3.17.(a) illustre ce problème : le sommet u occupe une grande surface et par conséquent la couche 2 aussi. Afin de réduire l'espace occupé, Schreiber répartit les sommets « grands » sur plusieurs couches. Dans la figure 3.17.(b), le sommet u est positionné sur plusieurs couches permettant ainsi de réduire l'espace occupé et d'augmenter la densité d'information. Enfin, la seconde modification est la prise en compte de contraintes telles que :

- Contrainte *haut-bas*, i.e. deux sommets doivent être positionnés l'un au dessous de l'autre sur deux couches contiguës
- Contrainte *gauche-droite*, i.e. deux sommets doivent être positionnés l'un à côté de l'autre (sur la même couche)
- Contrainte *horizontale*, i.e. deux sommets doivent être positionnés sur la même couche
- Contrainte *verticale*, i.e. deux sommets doivent être positionnés l'un au dessous de l'autre sur deux couches quelconques

Dans [25], Brandes *et al.* présentent un outil, intégré dans *Wilmascope* [39], permettant de comparer plusieurs voies métaboliques dans un espace en « deux dimensions et demie » (on peut en fait considérer ce dessin comme un dessin en 3 dimensions). La première étape consiste à assigner une strate (ou un plan) à chaque voie et à superposer ces strates. Puis si un composé ou une réaction est présent dans plusieurs voies, tous les sommets correspondants sont « fusionnés » (i.e. on considère maintenant qu'ils ne forment plus qu'un seul sommet). Le graphe résultant[9] est alors dessiné en utilisant l'algorithme hiérarchique de Gansner *et al.* [61]. L'un des aspects intéressants de ce travail est qu'il introduit un

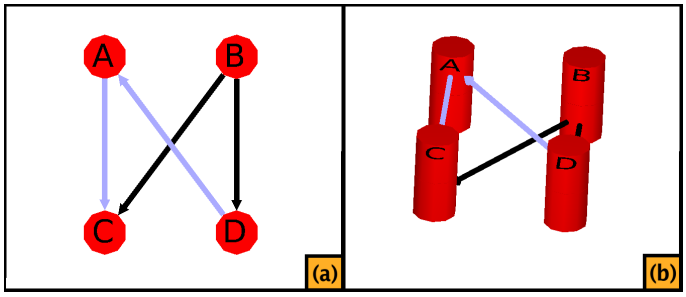

FIG. 3.18: Un nouveau problème de croisement d'arêtes ; (a) et (b) montrent ce graphe sous deux angles différents. Les croisements d'arêtes de strates différentes peuvent être « supprimés » par simple rotation du dessin.

nouveau problème de réduction du nombre de croisements d'arêtes. La figure 3.18 illustre ce fait, dans la figure (a) on observe un croisement d'arêtes, cependant on remarque dans la figure (b) que ce croisement n'est que visuel et ne doit pas être supprimé (puisqu'une simple rotation permet de ne plus le percevoir).

Le problème principal de ces trois techniques est qu'elles ne respectent pas l'une des conventions les plus utilisées en biologie puisqu'elles ne mettent pas en évidence les cycles de réactions.

Pour répondre à ce problème, la première étape de l'algorithme de Becker et Rojas [17] consiste à détecter le plus long cycle de la voie métabolique et à le regrouper en un métanoeud. Si ce métanoeud est supprimé, le graphe résultant peut être non-connexe (e.g. si le cycle est central dans la voie). Dans le cas échéant, le graphe résultant est alors décomposé en composantes connexes. Les auteurs distinguent deux cas : si une composante connexe ne contient qu'un sommet alors ce sommet est *composant interne* du cycle, sinon la composante est une *composante externe* du cycle. Chaque composante externe est alors regroupée en un métanoeud. Si une composante externe est un DAG, alors elle est dessinée en utilisant un algorithme hiérarchique, sinon l'algorithme est répété récursivement sur la composante. Le plus long cycle est lui dessiné par un algorithme circulaire. Enfin, le graphe quotient est dessiné par un algorithme par modèle de forces respectant certaines contraintes. Tout d'abord, les composantes internes sont contraintes dans le cycle auquel elles appartiennent, et deuxièmement l'algorithme interdit le chevauchement de métanoeuds.

[9] Lorsque les voies à comparer sont « proches », ce graphe est connexe

Dans [131], Wegner et Kummer ont affiné cette technique. La première étape de leur algorithme consiste à trouver tous les cycles ne partageant qu'un seul sommet, du plus petit au plus grand. Puis, comme dans l'algorithme de Becker et Rojas [17], chaque cycle est dessiné par un algorithme de dessin circulaire et les DAG par un algorithme hiérarchique. L'étape suivante consiste à « fusionner » les cycles partageant des sommets. Enfin, le dessin du graphe quotient est réalisé par un algorithme par modèle de forces.

Enfin, Gabouje et Zimányi proposent dans [57] une approche similaire. La première différence tient au fait que l'utilisateur peut contraindre manuellement le dessin (e.g. ne pas dessiner tel cycle sur un cercle). D'autre part, cet algorithme recherche tous les cycles mais aussi les cascades de réaction (séquence de sommets contigus de degré 2). Enfin, le graphe quotient est dessiné en utilisant un algorithme hiérarchique et non un algorithme par modèle de forces.

Ces algorithmes permettent de représenter plusieurs voies métaboliques (jusqu'à cinq selon [17]) simultanément mais ne peuvent être appliqués aux réseaux entiers.

3.4.2 Visualisation de réseaux métaboliques

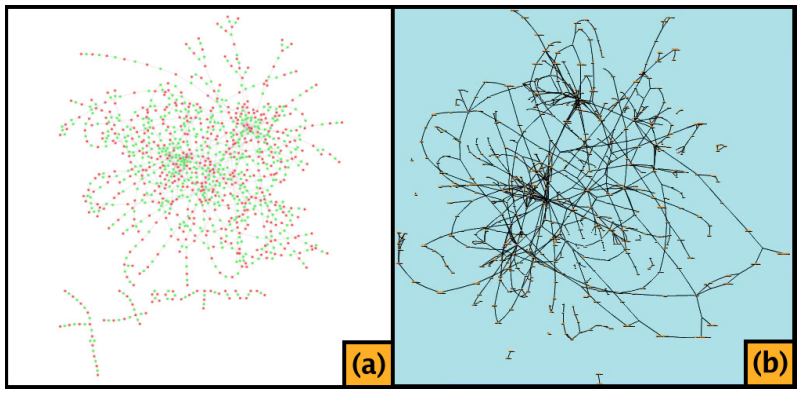

FIG. 3.19: Réseau métabolique de Escherichia Coli ; (a) dessiné par l'algorithme par modèle de forces de Cytoscape [118] et (b) par celui de SBMLviewer [1].

L'une des premières approches lorsque l'on veut représenter un réseau métabolique consiste à utiliser un algorithme classique de dessin de graphe. L'outil *SBMLviewer* [1] utilise en particulier un algorithme par modèle de forces (cf figure 3.19). *Cytoscape* [118], l'un des outils les plus usités pour visualiser des données biologiques, intègre d'autres algorithmes de dessin tels que des algorithmes hiérarchiques ou encore des algorithmes de dessin d'arbre. Cependant, comme mentionné dans [112], ces algorithmes ne produisent pas de dessin satisfaisant du point de vue des biologistes. La première raison tient au fait que ces algorithmes ne respectent pas les conventions de dessin. En particulier, les algorithmes par modèle de forces ne correspondent pas aux attentes des biologistes puisque les regroupements visuels de sommets ne correspondent pas à la décomposition du réseau en voies métaboliques. En effet, d'une part les regroupements de sommets créés par ce type d'algorithme ne correspondent pas à une voie métabolique mais plus généralement

à un agglomérat de voies métaboliques. D'autre part une voie métabolique n'est pas nécessairement connexe et donc il peut être difficile de visualiser une voie particulière dans son contexte (i.e. le réseau métabolique).

Il existe deux principaux outils permettant de dessiner automatiquement les réseaux métaboliques tout en conservant l'information liée au découpage en voies métaboliques.

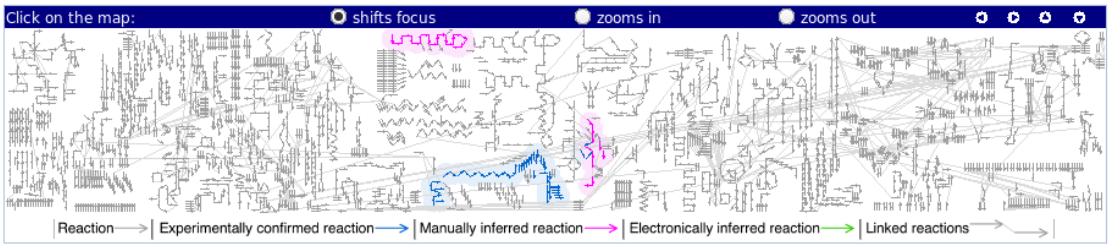

FIG. 3.20: Capture d'écran du logiciel Reactome [77]. Cette carte représente le réseau métabolique de Homo Sapiens. Les zones en rouge et en bleu du dessin représentent les cycles de vie du virus HIV.

Reactome [77] est une application internet permettant de visualiser des données (e.g. voies et réseaux métaboliques, réseaux d'interaction de protéines) provenant de différentes bases de données (dont celle de Reactome). La figure 3.20, montre une capture d'écran de cette application. Dans cet outil, chaque voie métabolique est représentée indépendamment des autres et des lignes colorées en gris clair permettent de visualiser les réactions partagées sur plusieurs voies. Afin que l'utilisateur puisse reconnaître plus aisément certains types de voies métaboliques, des « formes » (en anglais, *pattern*) ont été associées à certaines fonctionnalités. D'autre part, l'utilisateur peut mettre en évidence une sous-partie du réseau (e.g. une voie métabolique) en la sélectionnant dans une liste. Dans la figure 3.20, les cycles de vie du virus HIV sont mis en évidence en rouge et en bleu.

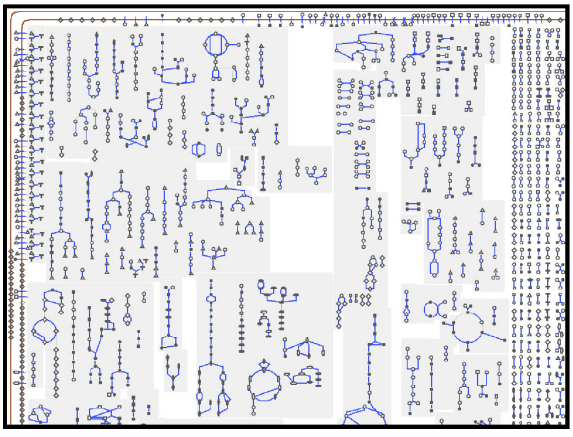

FIG. 3.21: Réseau métabolique complet de *E. Coli* dessiné par Pathway Tools cellular overview diagram

L'outil *Pathway Tools cellular overview diagram* [108] est utilisé dans *BioCyc* [87], l'une des bases de données importantes en biologie et bioinformatique. Tout comme dans

Reactome, les voies métaboliques sont représentées indépendamment. Pour positionner les voies les unes par rapport aux autres, *Pathway Tools cellular overview diagram* utilise l'ontologie des voies métaboliques de *MetaCyc* [30]. Les voies ayant des fonctionnalités « proches » sont donc représentées « proches » dans le dessin. Par exemple, les voies de synthèse sont positionnées sur la gauche, de dégradation sur la droite et de transport d'énergie au centre du dessin. Les liens entre les différentes voies métaboliques ne sont pas affichés afin d'augmenter la lisibilité, cependant l'utilisateur peut demander à afficher certains de ces liens.

Parmi les outils existant, aucun ne permet de respecter l'ensemble des attentes des biologistes. En effet, *SBMLViewer* [1] et *Cytoscape* [118] ne prennent pas en compte les contraintes fixées par les biologistes puisqu'ils utilisent des algorithmes classiques pour représenter les réseaux métaboliques. Quant à *Reactome* [77] et *Pathway Tools cellular overview diagram* [108], ces outils autorisent la duplication de sommets et ne permettent donc pas d'avoir une vue globale du réseau métabolique mais plutôt un ensemble de vues locales. Dans ce type de représentation, la connectivité Â«Â affichéeÂ Â» n'est pas représentative de la connectivité réelle du réseau, il semble nécessaire d'interdire la duplication de sommets pour obtenir une vue globale du réseau.

Chapitre 4
Décomposition de graphe

Nous nous intéressons dans ce chapitre à la décomposition de graphe en groupes chevauchants, c'est-à-dire dont l'intersection est non-vide. La recherche de tels groupes est utile dans de nombreux domaines d'application et en particulier en biologie. Par exemple, appliquée aux réseaux protéine-protéine, la décomposition permet d'identifier des familles protéines. De plus, trouver ces groupes permet de construire de bonnes abstractions de réseaux aidant ainsi les experts à analyser plus efficacement leurs données. Dans la section 4.1, nous présentons tout d'abord l'intérêt qu'apporte une décomposition chevauchante par rapport à une partition du graphe. Nous présentons ensuite les approches principales utilisées en visualisation d'information dans la section 4.2. Dans la section 4.3, nous expliquons l'algorithme que nous avons utilisé, puis nous donnons les généralisations de deux mesures de qualité de partition au cas général des décompositions dans la section 4.4. Enfin dans la section 4.5, nous donnons les résultats obtenus sur un jeu de données provenant d'un réseau social.

4.1 Introduction

Trouver des groupes (ou communautés) dans un réseau est généralement traduit en un problème de décomposition de graphe. Les algorithmes de décomposition recherchent des groupes d'éléments (ou *clusters*) ayant une (ou plusieurs) propriété(s) commune(s). Le critère le plus largement admis pour qu'un ensemble de groupes forme une « bonne » décomposition du graphe est que la densité de chaque groupe soit élevée mais aussi que la densité entre les différents groupes soit faible. Comme défini dans la section 2.2, une décomposition de graphe peut être représentée comme un couple (G, H) où G est le graphe et H est le DAG de décomposition. La figure 4.1 illustre cette définition. La figure 4.1.(a) montre un graphe G ainsi qu'un ensemble de groupes (mis en évidence par des enveloppes convexes de couleurs distinctes) et la figure 4.1.(b) montre le DAG de décomposition correspondant.

La plupart des algorithmes de décomposition produisent un partitionnement des sommets du graphe ce qui soulève une question importante : le résultat de l'algorithme est-il correct ? Ou encore, peut-il être correct ? Dans certains domaines, notamment en sciences sociales ou en biologie, les groupes « logiques » ne forment pas un partitionnement du

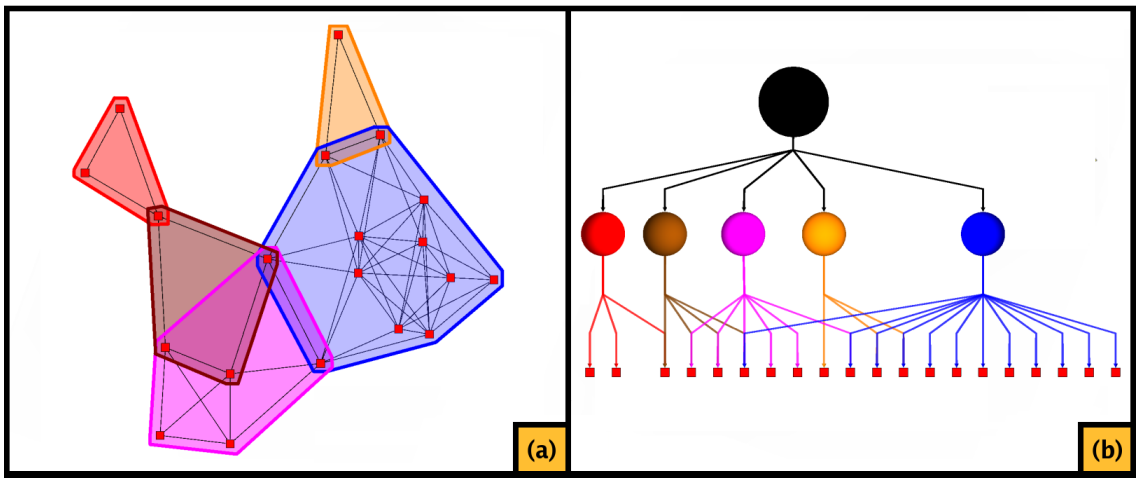

FIG. 4.1: Exemple de décomposition de graphe $\{G, H\}$. La figure (a) montre le graphe G et la figure (b) le DAG de décomposition H. Le sommet bleu de H représente l'ensemble des sommets du graphe G (feuilles dans H) que l'on peut atteindre dans le DAG depuis le sommet bleu.

graphe mais plutôt une décomposition chevauchante (i.e. différents groupes peuvent partager des sommets et arêtes). La figure 4.2.(a) montre un sous-graphe du « graphe d'Hollywood » où les sommets représentent des acteurs et deux acteurs sont reliés par une arête si les acteurs ont participé à un même film. On trouve dans cet exemple deux cliques maximales représentant deux films (mises en évidence par les arêtes bleues et vertes). Retrouver ces deux communautés consiste à trouver les groupes denses les plus grands possibles. Des algorithmes de partitionnement pourraient donner les résultats de la figure 4.2.(b) ou (c), tandis que le meilleur résultat est donné dans la figure 4.2.(d). Dans cet exemple et dans bien d'autres, la meilleure décomposition est une décomposition chevauchante.

Etant donné que dans le cas général, la « meilleure » décomposition n'est pas une partition des sommets, nous nous intéressons dans ce chapitre à une décomposition de graphe en groupes chevauchants. En d'autres termes, nous cherchons le « squelette » du graphe où les « os » sont des groupes de densités élevées et les articulations sont les sommets partagés par ces groupes (par exemple, voir la figure 4.3). Notre approche est basée sur la notion de *k-connexité* [1].

4.2 Etat de l'art

Nous présentons brièvement [2] dans cette section les trois approches les plus fréquemment utilisées en visualisation d'information pour décomposer un graphe : les approches divisives, agglomératives et topologiques. Les algorithmes présentés ici ne forment pas une liste exhaustive de l'état de l'art de la décomposition de graphe, pour une présentation plus précise des algorithmes existants le lecteur peut se référer à l'article de Schaeffer [114].

[1] cf définition d'un graphe k-connexe dans la section 2.2

[2] Voir la section 3.1 pour une présentation plus détaillée

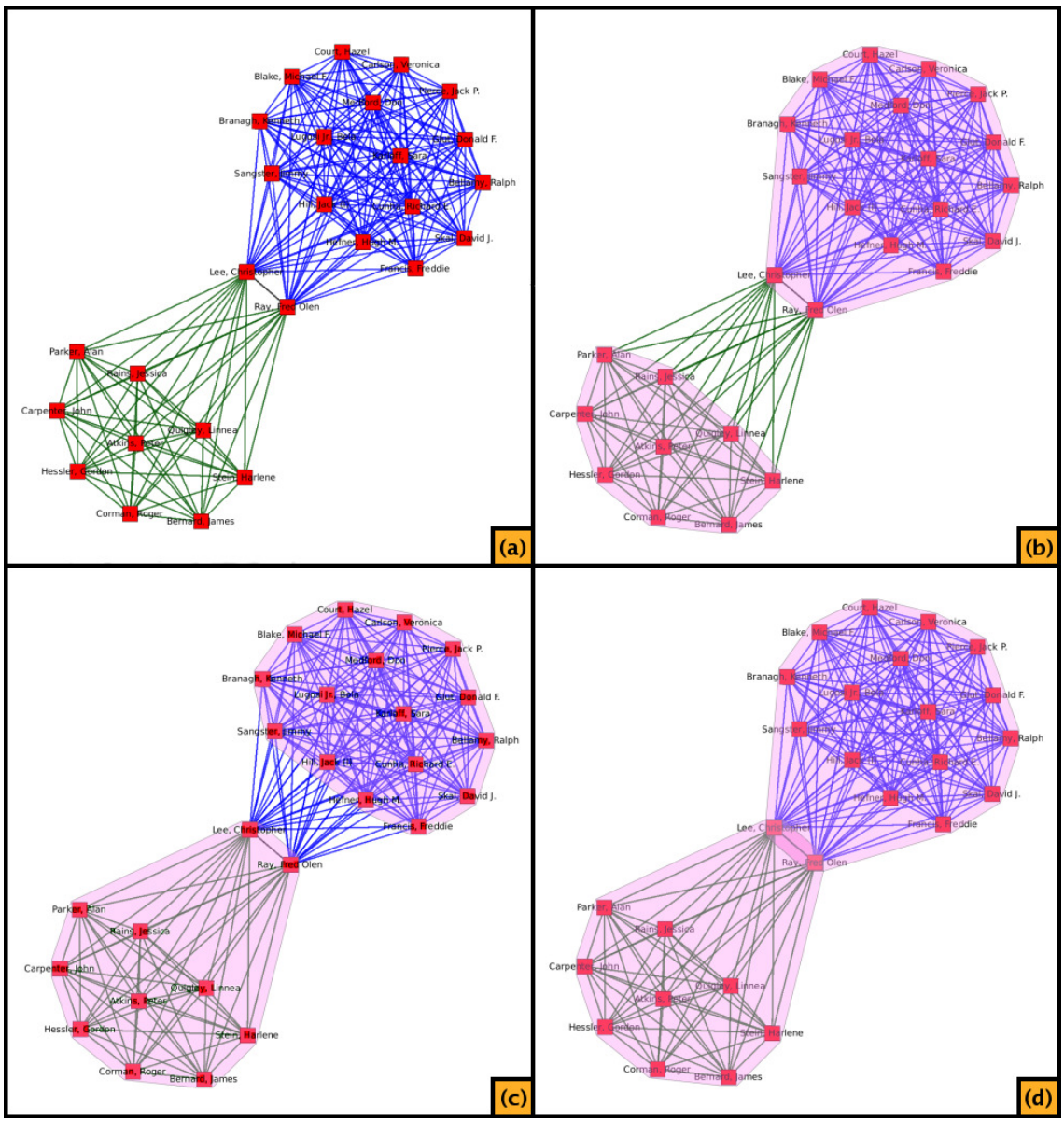

FIG. 4.2: (a) Sous-graphe du « graphe d'Hollywood » ; (b) et (c) Résultats d'algorithmes de partitionnement, chaque groupe est entouré par une enveloppe convexe, dans chacun de ces deux cas, l'une des deux cliques maximales a été « scindée » ; (d) Chacune des deux cliques maximales a été détectée.

Dans [105], Newman et Girvan présentent un algorithme divisif basé sur la métrique *Betweenness Centrality* [53] et la mesure de modularité, notée Q (introduite dans [105]), dont la complexité en temps est $O(|V| \cdot |E|^2)$. L'idée de cet algorithme est de retirer une à une les arêtes de fortes *Betweenness Centrality* puisque ces arêtes font le « pont » entre les communautés et d'ainsi décomposer le graphe en groupes. Dans [13], Auber *et al.* donnent un autre algorithme divisif basé sur la métrique *Strength* [32, 13] et sur la mesure de qualité MQ introduite par Mancoridis *et al.* [98]. La complexité totale de l'algorithme n'est pas donnée dans [13] cependant la complexité en temps du calcul de *Strength* est $O(|E| \cdot d_{max}^2)$ où d_{max} est le degré maximum du graphe. Dans cette approche, les arêtes de « faibles » *Strength* sont retirées, la borne utilisée est celle permettant de maximiser la mesure MQ.

Dans [104], Newman présente un algorithme agglomératif basé sur la mesure de modularité Q dont la complexité en temps est $O(|V| \cdot |E|)$. Le principe de cet algorithme est de partir d'un état où chaque groupe contient un sommet unique (et chaque sommet est dans un groupe), puis d'agglomérer tour à tour deux groupes. A chaque étape, le choix des groupes agglomérés est réalisé grâce à la mesure de modularité : les groupes permettant d'augmenter le plus la valeur de la modularité sont « fusionnés ».

Dans ce chapitre, nous allons évaluer un autre algorithme non publié. Cet algorithme agglomératif est une adaptation de l'algorithme de Newman [104] à la mesure de qualité MQ. A chaque itération, cet algorithme utilise la mesure MQ pour trouver la paire de groupes dont la « fusion » augmenterait le plus la qualité du partitionnement. Dans la suite, nous appelons cet algorithme *MQ_agglo*.

Récemment, des articles ont été publiés sur des décompositions de graphe basées sur la détection de sous-structures topologiques particulières. Ces algorithmes sont généralement intégrés dans des processus de dessin de graphe [122, 7, 4, 10]. Toutes ces approches sont efficaces dans le cadre d'un système de visualisation, cependant elles ne permettent pas de trouver de groupes de densités élevées (hormis la décomposition en composantes biconnexes).

Dans [117], Seidman propose une méthode basée sur les degrés des sommets permettant de trouver les *k-cores* d'un graphe. Un *k-core* est un sous-graphe induit tel que les sommets de ce sous-graphe ont un degré supérieur ou égal à k dans le sous-graphe. Afin de décomposer le graphe, Seidman introduit la *coreness* d'un sommet u, i.e. l'entier maximum c tel que u appartient au c-core mais pas au $(c+1)$-core. L'algorithme consiste à décomposer le graphe en groupes de sommets d'égales *coreness*. D'après la définition de la *coreness*, il est évident que les groupes ainsi trouvés peuvent être non-connexes. Pour obtenir des groupes de densités plus élevées, nous décomposons donc chaque sous-graphe obtenu par cet algorithme en composantes connexes. Dans la suite de ce chapitre, nous appelons *K-cores* l'algorithme modifié de Seidman [117].

4.3 Décomposition en composantes hautement connexes

Les travaux existants en visualisation d'information (e.g. [122, 4, 10]) ont montré que la décomposition en composantes biconnexes est efficace. L'idée développée dans ce chapitre est de généraliser ces résultats aux décompositions en composantes 3- et 4-connexes

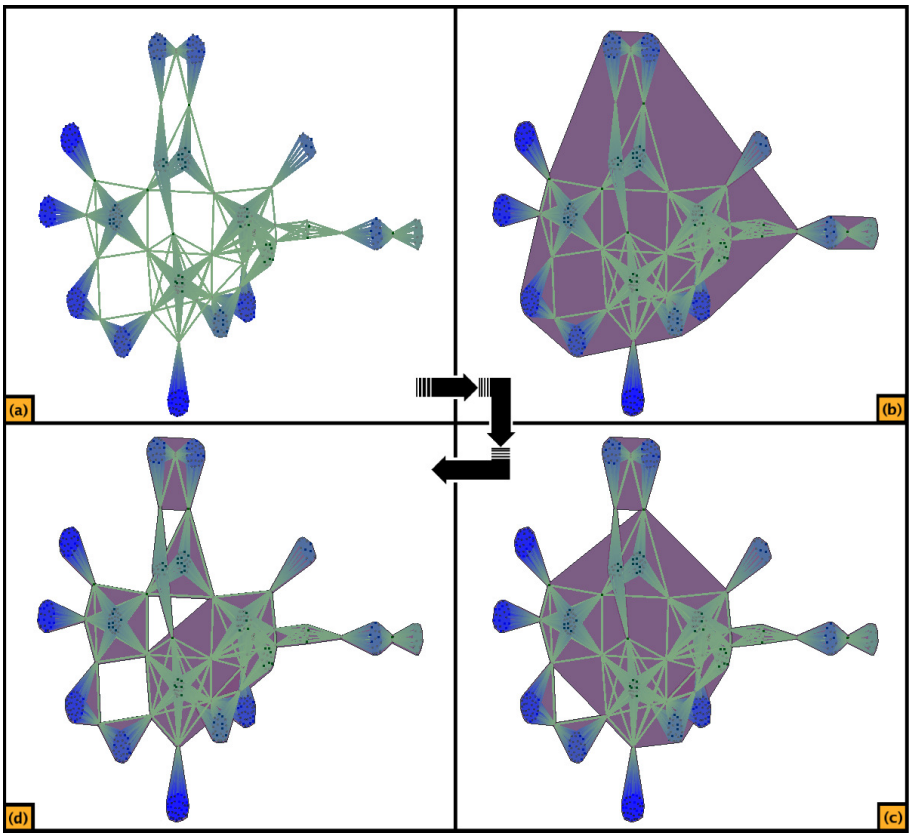

FIG. 4.3: (a) Sous-réseau de IMDb ; (b) Les composantes biconnexes sont entourées par des enveloppes convexes ; (c) Les composantes triconnexes sont entourées par des enveloppes convexes ; (d) Les composantes 4-connexes sont entourées par des enveloppes convexes.

en les comparant aux résultats d'approches utilisées en visualisation d'information (cf chapitre 3). Par conséquent, notre approche consiste à appliquer successivement des algorithmes de décomposition en composantes biconnexes, puis triconnexes et enfin 4-connexes (voir la figure 4.3). Les résultats de ces décompositions sont des groupes chevauchants, puisque chaque composante *k-connexe* ($k > 1$) partage au moins un sommet avec une autre composante *k-connexe* (et peut aussi partager une ou plusieurs arêtes).

4.3.1 Décompositions en composantes 2- et 3-connexes

Les propriétés des graphes *k-connexes* ont été largement étudiées en informatique théorique. De nombreux travaux ont été effectués en particulier sur la décomposition en composantes *biconnexes* et en composantes *triconnexes*.

4.3.1.1 Décomposition en composantes biconnexes

Il existe de nombreux algorithmes pour trouver les composantes biconnexes d'un graphe. Des algorithmes classiques de décomposition en composantes biconnexes sont

donnés dans [124] et [15]. La méthode consiste à premièrement calculer un arbre couvrant en utilisant un parcours en profondeur [33] du graphe (aussi appelé *DFS*). Lors de ce parcours, certains arcs sont inversés (la source devient la destination et inversement) de telle sorte que l'arbre couvrant soit un arbre enraciné. Puis l'algorithme classifie les arcs en fonction des positions relatives de leurs extrémités. En particulier, un arc dont la source a une profondeur plus élevée que la destination est appelé *arc retour*.

Soit T_r le sous-arbre enraciné en un sommet r de l'arbre couvrant. S'il existe au moins un chemin entre chaque descendant de r et r formé d'arcs de T_r et de au plus un arc retour, et s'il n'existe aucun de ces chemins entre un descendant de r et un ancêtre de r, alors le graphe induit dans G par les sommets de T_r est une composante biconnexe de G. Cet algorithme permet de calculer les composantes biconnexes d'un graphe $G = (V, E)$ en $O(|V| + |E|)$.

4.3.1.2 Décomposition en composantes triconnexes

Dans [73], Hopcroft et Tarjan donnent un algorithme analogue pour calculer les composantes triconnexes d'un graphe biconnexe. Tout comme dans la décomposition en composantes biconnexes, cet algorithme commence par effectuer un *DFS* et classifier les arcs. Puis pour chaque sommet u, ils définissent deux valeurs : $lowPt1(u)$ et $lowPt2(u)$. Ces deux valeurs sont respectivement la profondeur du sommet le moins profond et la profondeur du « deuxième » sommet le moins profond dans l'arbre couvrant que l'on peut atteindre depuis u par un chemin comme défini dans la partie 4.3.1.1. En utilisant ces deux valeurs, les composantes triconnexes d'un graphe biconnexe $G = (V, E)$ sont calculées en temps $O(|V| + |E|)$.

La première étape de notre approche consiste à calculer les composantes biconnexes du réseau en utilisant l'algorithme présenté dans [15]. Cela permet d'obtenir le premier niveau de notre décomposition (par exemple, voir la figure 4.3.(b)). Afin de raffiner cette décomposition, nous recherchons ensuite les composantes triconnexes de chaque composante biconnexe du graphe en utilisant l'algorithme de Hopcroft et Tarjan [73] (par exemple, voir la figure 4.3.(c)). Cette recherche est effectuée dans chacune des composantes biconnexes afin non seulement d'obtenir des composantes plus denses mais aussi de réduire le temps de calcul. Par abus de langage, nous appellerons, dans la suite de cette thèse, composantes triconnexes du graphe les composantes obtenues lors de cette étape.

4.3.2 Décomposition en composantes 4-connexes

Etant donné que le principe de notre approche est de trouver des composantes de plus en plus denses, nous effectuons ensuite une décomposition en composantes 4-connexes de chaque composante triconnexe trouvée lors de l'étape précédente. Nous avons pour cela implémenté notre propre algorithme. L'idée de notre approche est de calculer ces composantes 4-connexes en utilisant l'algorithme de décomposition en composantes triconnexes [73].

Si l'on supprime un sommet u d'une composante triconnexe C_{trico} du graphe et que la composante résultante C'_{trico} n'est plus triconnexe, alors la composante C_{trico} n'est

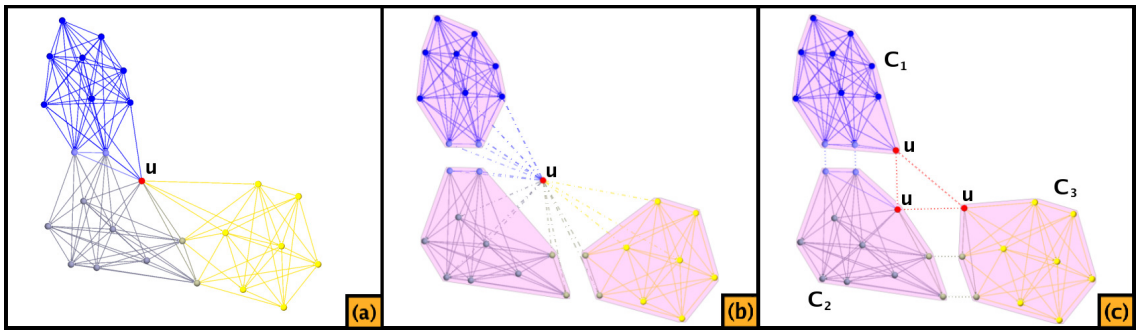

FIG. 4.4: (a) Un exemple de graphe triconnexe ; (b) Après la suppression du sommet u, le graphe résultant peut être décomposé en composantes triconnexes ; (c) Si on ajoute le sommet u à chaque composante de (b), on obtient alors trois composantes C_1, C_2 et C_3 et il n'existe pas de *3-séparateurs* (cf définition dans la section 2.2) contenant u dans C_1, C_2 ou C_3.

pas 4-connexe. En appliquant l'algorithme de Hopcroft et Tarjan [73] sur C'_{trico}, on trouve tous les *2-séparateurs*[3] de C'_{trico} et par conséquent tous les *3-séparateurs* de la composante triconnexe C_{trico} contenant le sommet u. La figure 4.4 illustre cette technique, lorsque que l'on supprime le sommet u du graphe de la figure 4.4.(a), le graphe résultant n'est plus triconnexe. On obtient la décomposition du graphe résultant en appliquant l'algorithme de décomposition en composantes triconnexes [73] (cf figure 4.4.(b)). La figure 4.4.(c) montre les composantes que l'on obtient par ce processus. En répétant pour chaque sommet ce processus, on obtient tous les *3-séparateurs* de la composante triconnexe. On peut ainsi construire la décomposition en composantes 4-connexes.

Comme nous répétons $|V|$ fois l'algorithme de décomposition en composantes triconnexes, dans le pire des cas la décomposition en composantes 4-connexes est effectuée en temps $O(|V| \cdot |E|)$ (le pire des cas est celui où le graphe est triconnexe). Cette complexité en temps est relativement élevée, cependant il est important de noter que la recherche de composantes 4-connexes n'est effectuée que dans les composantes triconnexes ce qui rend utilisable cette technique (voir la section 4.5 pour les temps de calculs). Par abus de langage, nous appellerons, dans la suite de cette thèse, composantes 4-connexes du graphe les composantes obtenues lors de cette étape et décomposition en composantes 4-connexes l'algorithme permettant de les obtenir.

En supprimant un sommet d'une composante 4-connexe et en recherchant les composantes 4-connexes de la composante résultante, on pourrait calculer les composantes 5-connexes en temps $O(|V|^2 \cdot |E|)$. Cette approche pourrait donc être aisément généralisée à la décomposition en composantes k-connexes. La complexité en temps d'un tel algorithme serait alors $O(|V|^{k-3} \cdot |E|)$. Dans ce chapitre, nous nous sommes concentrés sur la décomposition en composantes 4-connexes. En effet, pour tout $k > 4$, quand bien même la décomposition en composantes k-connexes donnerait des résultats excellents, calculer ces composantes nécessiterait un temps de calcul trop important rendant ainsi l'algorithme inutilisable sur des données réelles.

[3] cf définition des *k-séparateurs* d'un graphe dans la section 2.2

4.4 Evaluer la qualité d'une décomposition chevauchante

Le résultat de notre algorithme est une décomposition en groupes chevauchants (i.e. partageant des sommets et/ou des arêtes). Afin d'évaluer la qualité de ces résultats, nous devons utiliser des mesures de qualité permettant de comparer les résultats avec ceux obtenus par des algorithmes de partitionnement. Nous présentons dans cette section trois mesures de qualité initialement utilisées pour évaluer les qualités de partitionnements. Pour deux de ces mesures, nous montrons qu'elles peuvent être appliquées à toute décomposition (chevauchante ou non). En ce qui concerne la mesure de qualité MQ, nous en donnons une généralisation aux décompositions de chevauchantes.

4.4.1 Densité

Une méthode naïve pour évaluer la qualité d'une décomposition est de calculer la densité moyenne de ses groupes. Supposons que $C = \{C_1, C_2, ..., C_k\}$ soit l'ensemble des groupes de la décomposition. Soit δ_{C_i} la densité du groupe C_i définie comme suit :

$$\delta_{C_i} = \frac{|E(C_i, C_i)|}{|C_i|(|C_i| - 1)/2} \tag{1}$$

où $|E(C_i, C_i)|$ est le nombre d'arêtes du graphe dont les deux extrémités sont dans C_i et $|C_i|$ est le nombre de sommets contenus dans C_i. La densité moyenne δ des groupes de C est définie comme suit :

$$\delta = \frac{1}{k} \sum_{i=1}^{k} \delta_{C_i} \tag{2}$$

La densité moyenne des groupes donne une idée de la qualité de la décomposition, cependant il semble que cette mesure ne soit pas assez fine pour évaluer efficacement cette qualité. Prenons l'exemple d'une décomposition où chaque groupe est composé d'une arête et de deux sommets (clique à deux sommets, K_2). Dans ce cas, la densité moyenne sera égale à 1. Il est cependant clair que cette décomposition ne peut être optimale pour tout graphe. Par conséquent, cette mesure n'est utile que si elle est utilisée conjointement avec d'autres mesures de qualité.

4.4.2 Modularité

La mesure de qualité de Newman et Girvan [105], appelée *modularité*, a été mise en place pour évaluer la qualité d'une partition des sommets (voir la section 3.1.1.1). On rapelle que la formule de la modularité est :

$$Q = \frac{1}{|E|} \sum_{i} (|E(C_i)| - p_{C_i} \cdot \sum_{j} |E(C_i, C_j)|) \tag{3}$$

Intuitivement, nous pouvons utiliser cette mesure pour évaluer la qualité d'une décomposition chevauchante. En effet, cette mesure ne tient pas compte d'arêtes inter-groupes mais uniquement des arêtes intra-groupes et des arêtes avec au moins une extrémité dans chaque groupe (qu'elles soient intra- ou inter-groupes). Nous pouvons donc mesurer la qualité de notre décomposition mais aussi la comparer à d'autres algorithmes, et plus particulièrement à des algorithmes de partitionnement.

4.4.3 Généralisation de MQ

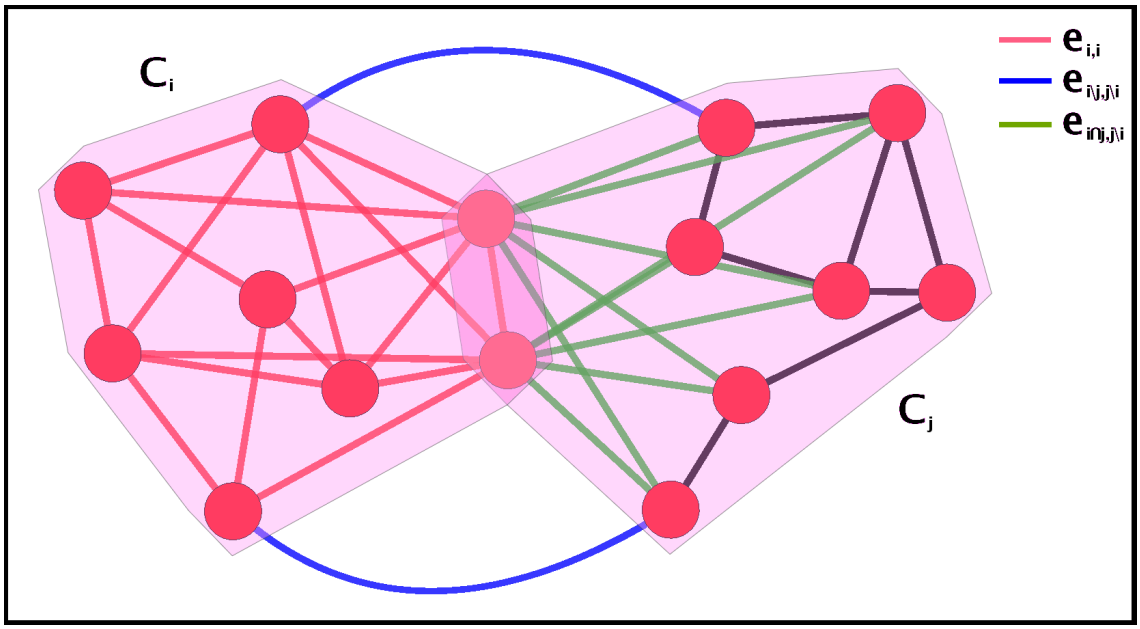

FIG. 4.5: Nous nous intéressons sur cet exemple au groupe C_i. On observe trois types d'arêtes : les arêtes internes à C_i notées $e_{i,i}$ (en rouge), les arêtes dont une extrémité est dans $C_i \cap C_j$ et l'autre dans $C_j \setminus C_i$ notées $e_{i\cap j,j\setminus i}$ (en vert), et les arêtes dont une extrémité est dans $C_i \setminus C_j$ et l'autre dans $C_j \setminus C_i$ notées $e_{i\setminus j,j\setminus i}$ (en bleu). Seules les arêtes vertes et bleues doivent compter dans la cohésion externe du groupe C_i

Dans [13], Auber *et al.* utilisent la mesure de qualité MQ d'une partition des sommets introduite par Mancoridis *et al.* dans [98] (cf chapitre 3). On rappelle que pour un graphe $G = (V, E)$ et une partition $C = \{C_1, C_2, ..., C_k\}$ des sommets de G, MQ est définie comme suit :

$$MQ = MQ^+ - MQ^- \tag{4}$$

avec

$$MQ^+ = \frac{1}{k} \sum_i s(C_i, C_i) \tag{5}$$

et,

$$MQ^- = \frac{1}{k(k-1)} \sum_{i,j \neq i} s(C_i, C_j) \tag{6}$$

où $s(C_i, C_j) = \frac{|E(C_i, C_j)|}{|C_i| \cdot |C_j|}$.

Afin d'adapter la mesure MQ aux décompositions chevauchantes, il n'est pas nécessaire de modifier MQ^+. En effet, MQ^+ représente la cohésion interne des groupes (i.e. la densité) et la cohésion des groupes ne doit pas dépendre du nombre de sommets (et arêtes) partagés par les groupes. Par contre, nous devons modifier la fonction MQ^- pour prendre en compte le fait que nous n'avons pas une partition des sommets et des arêtes mais une décomposition chevauchante. On peut réécrire la formule de MQ^- comme suit :

$$MQ^- = \frac{1}{k(k-1)} \sum_i mq^-(C_i) \tag{7}$$

où

$$mq^{-}(C_i) = \sum_{j \neq i} s(C_i, C_j)$$

est la « participation » du groupe C_i à la cohésion externe de la décomposition. Le principe de base est qu'une arête interne à un groupe C_i ne doit pas être prise en compte dans le calcul de $mq^{-}(C_i)$. Dans la figure 4.5, aucune arête rouge (arête interne à C_i) ne doit être prise en compte dans la participation de C_i à la cohésion externe de la décomposition. Seules les arêtes bleues et vertes, i.e. les arêtes dont une extrémité est dans C_i et l'autre n'est pas dans C_i, doivent être comptabilisées dans $mq^{-}(C_i)$. Pour une décomposition chevauchante, nous obtenons donc la formule suivante :

$$mq^{-}_{Over}(C_i) = \sum_{j \neq i} s_{Over}(C_i, C_j)$$

où $s_{Over}(C_i, C_j) = \frac{|E(C_i, C_{j \setminus i})|}{|C_i| \cdot |C_{j \setminus i}|}$ et $C_{j \setminus i} = C_j \setminus (C_j \cap C_i)$. La cohésion externe de la décomposition est alors

$$MQ_{Over}{}^{-} = \frac{1}{k(k-1)} \sum_i mq^{-}_{Over}(C_i) \tag{8}$$

Finalement, on obtient pour une décomposition chevauchante la formule suivante :

$$MQ_{Over} = MQ^{+} - MQ_{Over}{}^{-} \tag{9}$$

Il est intéressant de noter que dans le cas d'une partition des sommets, on a $MQ_{Over} = MQ$. En effet, dans le cas d'une partition $C_i \cap C_j = \emptyset$, $\forall i \neq j$, et par conséquent $s_{Over}(C_i, C_j) = s(C_i, C_j)$.

4.5 Evaluation de la qualité de la décomposition

Pour évaluer la qualité et les performances de notre méthode, nous l'avons testée sur un banc d'essai de référence. Le jeu de données utilisé est celui du concours InfoVis 2007 [2] dont il est bien connu qu'il contient des structures de communauté.

4.5.1 Données et protocole

Le jeu de données du concours InfoVis 2007 [2] est extrait de la base de données cinématographique IMDb [4] (pour *Internet Movie Database*). Nous avons construit un graphe correspondant à ces données de la manière suivante : les sommets du graphe sont des acteurs (ou actrices) et deux sommets sont reliés par une arête si les deux acteurs correspondants ont participé à (au moins) un même film. La figure 4.6.(a) montre la composante connexe principale du graphe ainsi obtenue, cette composante contient 117948 sommets et 1917841 arêtes. Le nombre de sommets et d'arêtes de ce graphe ne permet pas d'appliquer tous les algorithmes de décomposition que nous voulons comparer. Nous avons par conséquent effectué notre étude sur un ensemble de plus de 400 sous-graphes

[4] `http ://www.imdb.com`

extraits de la composante principale. Pour obtenir des sous-graphes connexes contenant des structures de communauté, nous les avons extraits de la manière suivante : partant d'un sommet tiré aléatoirement, nous effectuons un parcours en largeur [5] du graphe (aussi appelé *BFS*) de profondeur 3 (i.e. la profondeur de l'arbre obtenu est égale à 3). Etant donné que ce type de réseaux a un faible diamètre [6], nous avons borné le nombre de voisins visités à chaque étape du *BFS* à 10. Enfin, le sous-graphe extrait est le sous-graphe induit dans le graphe par les sommets visités lors du *BFS*. Etant donné la méthode utilisée pour extraire cette collection de graphes, il est clair que ces graphes peuvent partager et partagent des sommets. Il est intéresant de noter que tout sommet du graphe original est contenu par (au moins) l'un des sous-graphes de la collection. La figure 4.6 montre une sous-partie des sous-graphes extraits par cette méthode, ces graphes contiennent en moyenne 342.4 sommets et 1280.7 arêtes. On peut aisément remarquer que ces sous-graphes contiennent bien des structures de communauté.

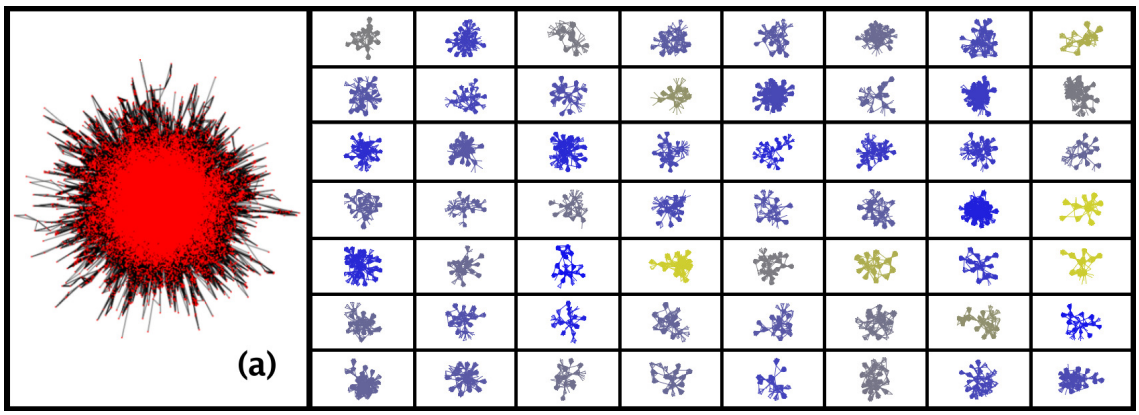

FIG. 4.6: (a) Composante connexe principale des données du concours InfoVis 2007 : ce graphe contient 117948 sommets et 1917841 arêtes ; Les autres graphes sont issus de la collection de sous-graphes extraits du graphe (a). On peut remarquer dans ces sous-graphes des parties ayant une haute densité, ce sont les structures de communauté. La couleur des sous-graphes est calculée en fonction de la valeur de MQ_{Over} de la décomposition en composantes 4-connexes du jaune pour les valeurs faibles au bleu pour les plus fortes (de 0.81 à 0.95).

4.5.2 Résultats et Analyse

Dans cette partie, nous comparons les résultats que nous avons obtenus à ceux obtenus par les algorithmes agglomératifs de [104] et de son adaptation à la mesure MQ, par les algorithmes divisifs de [105] et [13] et enfin aux résultats de l'algorithme topologique [117]. Cette comparaison est faite sur le temps de calcul, la densité moyenne, la modularité et MQ_{Over}. Etant donné que les algorithmes de [104] et [105] sont des algorithmes de partitionnement multi-niveaux (i.e. l'arbre de partition correspondant a une

[5] cf définition dans la section 2.2

[6] i.e. la longueur du plus long des plus courts chemins entre deux sommets

hauteur strictement supérieure à 2), nous devons choisir quels niveaux de ces décompositions nous allons comparer. Comme décrit dans [105, 104], pour obtenir le meilleur niveau de la hiérarchie, nous calculons la valeur de Q pour chaque niveau et nous n'utiliserons que le niveau qui maximise Q dans notre comparaison. De la même manière, nous utilisons la mesure MQ pour définir le « meilleur » niveau de la hiérarchie obtenue par MQ_agglo.

4.5.2.1 Algorithmes agglomératifs

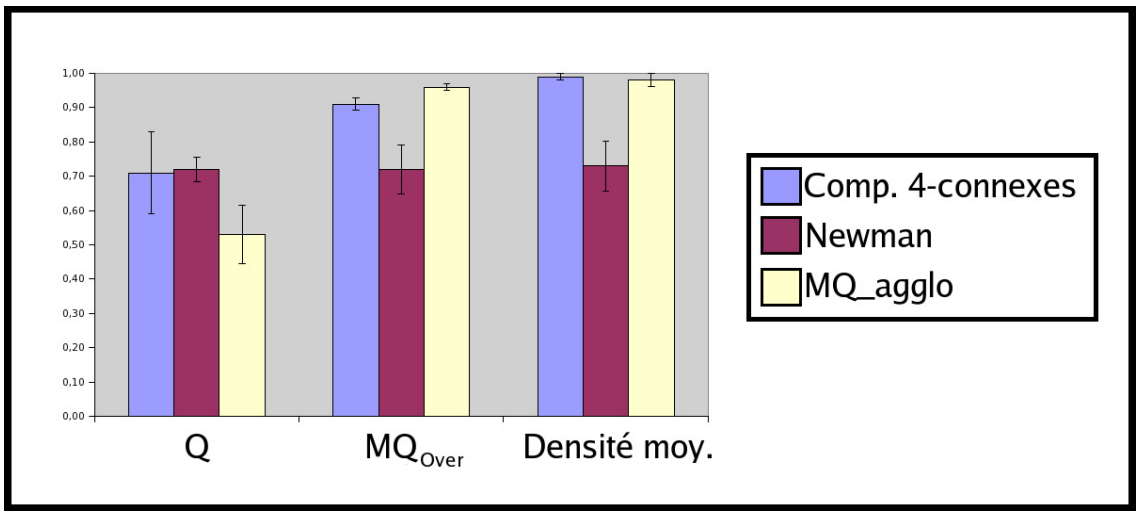

FIG. 4.7: Résultats qualitatifs de chacun des deux algorithmes agglomératifs Newman [104] et MQ_agglo ainsi que de la décomposition en composantes 4-connexes sur la collection de sous-graphes extraits des données du concours InfoVis 2007. Ce graphique montre les valeurs moyennes de Q, MQ_{Over} et de la densité moyenne des groupes ainsi que leurs écarts types respectifs.

La figure 4.7 montre les résultats obtenus par les algorithmes de Newman [104], MQ_agglo ainsi que ceux de la décomposition en composantes 4-connexes.

Comme attendu, l'algorithme de [104] (resp. MQ_agglo) est celui des trois algorithmes qui offre les meilleurs résultats pour la mesure Q (resp. MQ_{Over}). En effet, chacun de ces deux algorithmes a été mis en place pour maximiser ces mesures de qualité.

L'algorithme de [104] trouve des résultats qui ont en moyenne une valeur $Q = 0.72$ avec un écart type de 0.05. La décomposition en composantes 4-connexes (resp. l'algorithme MQ_agglo) trouve une valeur moyenne $Q = 0.71$ avec un écart type 0.17 (resp. $Q = 0.53$ avec un écart type 0.16). L'algorithme MQ_agglo donne donc de mauvais résultats pour la mesure Q puisque que la modularité moyenne est relativement faible et que l'écart type est lui relativement fort. En ce qui concerne la décomposition en composantes 4-connexes, la modularité moyenne est très proche de celle obtenue par l'algorithme de Newman [104]. Cependant l'écart type est 0.17 ce qui montre que la qualité des résultats (selon la mesure Q) varie suivant le graphe à décomposer. Les graphes de la collection dont la décomposition en composantes 4-connexes n'offre pas une « bonne » modularité ont tous un point commun, ce sont des graphes contenant une grande composante 4-connexe.

Tandis que l'algorithme de Newman [104] décompose ces composantes, la décomposition en composantes 4-connexes ne le permet pas.

Pour ce qui est de la mesure MQ_{Over}, l'algorithme MQ_agglo donne une valeur moyenne $MQ_{Over} = 0.96$ avec un écart type de 0.01 tandis que les algorithmes de [104] et la décomposition en composantes 4-connexes donnent respectivement $MQ_{Over} = 0.72$ et $MQ_{Over} = 0.91$ avec des écart types 0.1 et 0.02. L'algorithme de Newman [104] offre une valeur relativement bonne de MQ_{Over}, cependant cette valeur est très inférieure à celles obtenues par MQ_agglo et par la décomposition en composante 4-connexes. D'autre part, l'écart type montre que la qualité varie en fonction du graphe en entrée. Au contraire, la décomposition en composantes 4-connexes offre une bonne valeur moyenne de MQ_{Over} et un écart type très faible. Il est intéressant de noter que, sur ce jeu de données, les décompositions en composantes 4-connexes n'offrant pas de « bonnes » mesures de modularité ont au contraire de « bonnes » valeurs de MQ_{Over}. Ces deux mesures ne permettent effectivement pas de mesurer les mêmes « qualités » pour une décomposition donnée, la modularité semblant plus sensible aux tailles des groupes trouvés.

Enfin, la densité moyenne des groupes trouvés par la décomposition en composantes 4-connexes est de $\delta = 0.99$ avec un écart type de 0.01 alors que celles des algorithmes de Newman [104] et MQ_agglo sont respectivement de $\delta = 0.73$ et $\delta = 0.98$ avec des écarts types de 0.1 et 0.02. Dans les cas où la décomposition en composantes 4-connexes ne permet pas d'avoir une décomposition de « bonne » qualité selon Q (ou MQ_{Over}), la densité moyenne (ainsi que l'écart type) montre que les groupes obtenus représentent réellement des communautés (ou des agglomérats de communautés proches).

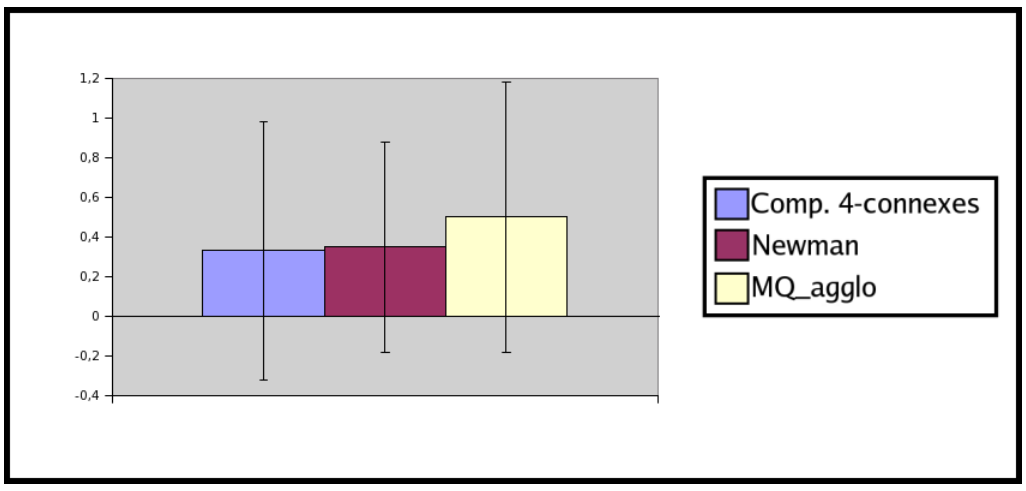

FIG. 4.8: Temps de calcul moyen en secondes ainsi que l'écart type de chacun des deux algorithmes agglomératifs Newman [104] et MQ_agglo ainsi que de la décomposition en composantes 4-connexes sur la collection de sous-graphes extraits des données du concours InfoVis 2007.

La figure 4.8 montre les temps de calcul de ces trois algorithmes. Ces temps sont comparables puisque les complexités théoriques sont identiques. Seul l'algorithme MQ_agglo offre un temps moyen légèrement supérieur aux deux autres algorithmes (cela est dû au

coût du calcul de MQ_{Over} à chaque itération de l'algorithme agglomératif). Il est intéressant de noter que l'écart type du temps de calcul de la décomposition en composantes 4-connexes est relativement élevé, encore une fois, cela est dû à la présence de Â«Â grandesÂ Â» composantes triconnexes dans certains graphes du jeu de données.

La décomposition en composantes 4-connexes est donc un bon compromis entre ces deux approches agglomératives puisque pour des temps de calcul similaires, elle permet d'obtenir de bonnes valeurs moyennes (comparables au meilleur des deux algorithmes) de Q et de MQ_{Over}.

4.5.2.2 Algorithmes divisifs

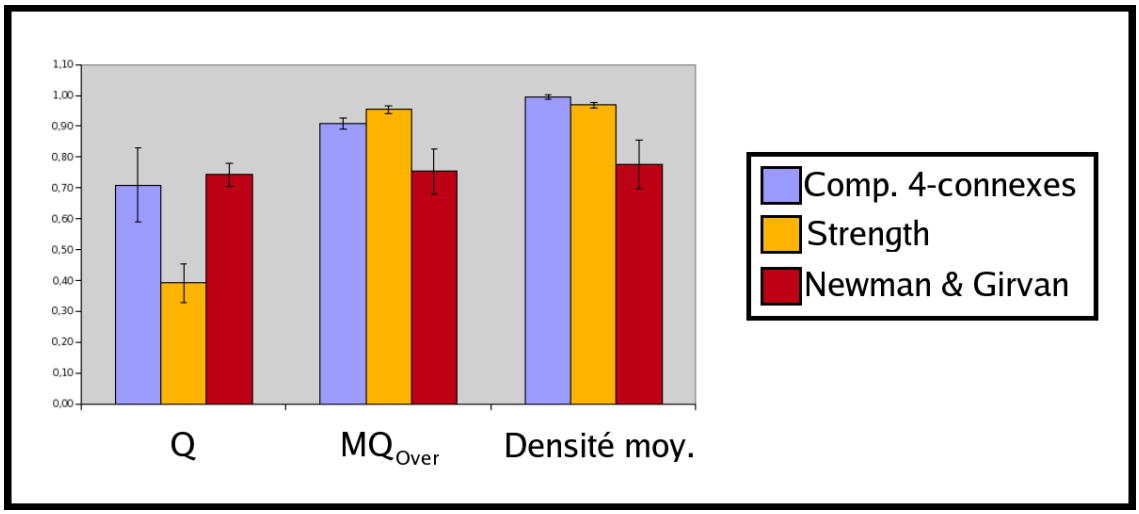

FIG. 4.9: Résultats qualitatifs de chacun des deux algorithmes divisifs Newman et Girvan [105] et Auber *et al.* [13] ainsi que de la décomposition en composantes 4-connexes sur la collection de sous-graphes extraits des données du concours InfoVis 2007. Ce graphique montre les valeurs moyennes de Q, MQ_{Over} et de la densité moyenne des groupes ainsi que leurs écarts types respectifs.

La figure 4.9 montre les résultats obtenus par les algorithmes divisifs de Newman et Girvan [105] et Auber *et al.* [13] ainsi que ceux de la décomposition en composantes 4-connexes. Encore une fois, les algorithmes de Newman et Girvan [105] et de Auber *et al.* [13] donnent respectivement les meilleurs résultats pour les mesures Q et MQ_{Over}. Ces résultats ne sont pas étonnants puisque ces algorithmes tentent de maximiser ces deux mesures.

L'algorithme de Newman et Girvan [105] donne une mesure de modularité moyenne $Q = 0.74$ avec un écart type de 0.05 tandis que l'algorithme de Auber *et al.* [13] donne $Q = 0.39$ avec un écart type 0.17. La décomposition en composantes 4-connexes permet donc d'obtenir une valeur moyenne de Q proche de celle offerte par l'algorithme de Newman et Girvan [105]. Encore une fois, étant donné l'écart type des résultats de la décomposition en composante 4-connexes (0.17), il faut nuancer ces résultats. Même en considérant l'écart

type, ces résultats restent bien meilleurs que ceux offerts par l'algorithme de Auber *et al.* [13].

Pour ce qui est de la mesure MQ_{Over}, l'algorithme de Auber *et al.* [13] donne une valeur moyenne $MQ_{Over} = 0.95$ avec un écart type de 0.01 tandis que celui de Newman et Girvan donne $MQ_{Over} = 0.75$ avec un écart type de 0.1. On peut noter qu'ici encore, la décomposition en composantes 4-connexes donne des résultats similaires à ceux de l'algorithme de Auber *et al.* [13].

Enfin, les algorithmes de Newman et Girvan [105] et de Auber *et al.* [13] donnent respectivement des densités moyennes de 0.77 et 0.97 avec des écarts types de 0.1 et 0.01. La densité moyenne des groupes trouvés par l'algorithme de Newman et Girvan [105] est donc très inférieure à celle des groupes trouvés par la décomposition en composantes 4-connexes ($\delta = 0.99$).

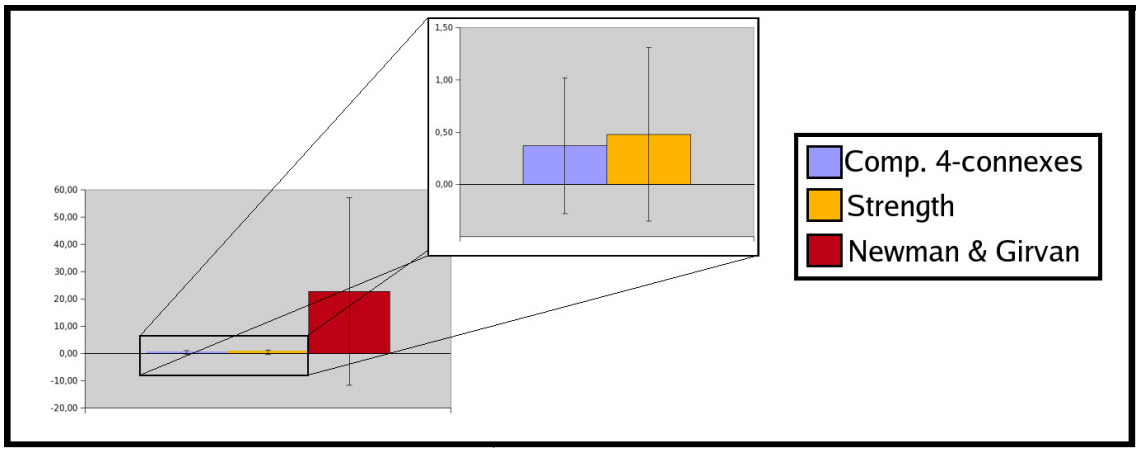

FIG. 4.10: Temps de calcul moyen en secondes ainsi que l'écart type de chacun des algorithmes divisifs de Newman et Girvan [105] et de Auber *et al.* [13] ainsi que de la décomposition en composantes 4-connexes sur la collection de sous-graphes extraits des données du concours InfoVis 2007.

La figure 4.10 montre les temps de calcul de chacun des algorithmes de Newman et Girvan [105], de Auber *et al.* [13] et de la décomposition en composantes 4-connexes. La vue de gauche montre les résultats de ces trois algorithmes. On remarque que l'algorithme de Newman et Girvan [105] nécessite un temps de calcul moyen très important (22.64 secondes) comparé aux temps de calcul des deux autres algorithmes. La vue de droite montre les temps de calcul moyens de l'algorithme de Auber *et al.* [13] et de la décomposition en composantes 4-connexes, ces temps de calcul sont comparables puisque respectivement de 0.51 et de 0.34 secondes.

Comparé à ces deux algorithmes divisifs, la décomposition en composantes 4-connexes est donc efficace non seulement en terme de temps de calcul mais aussi en terme de qualité (que ce soit pour Q, pour MQ_{over} ou encore pour δ).

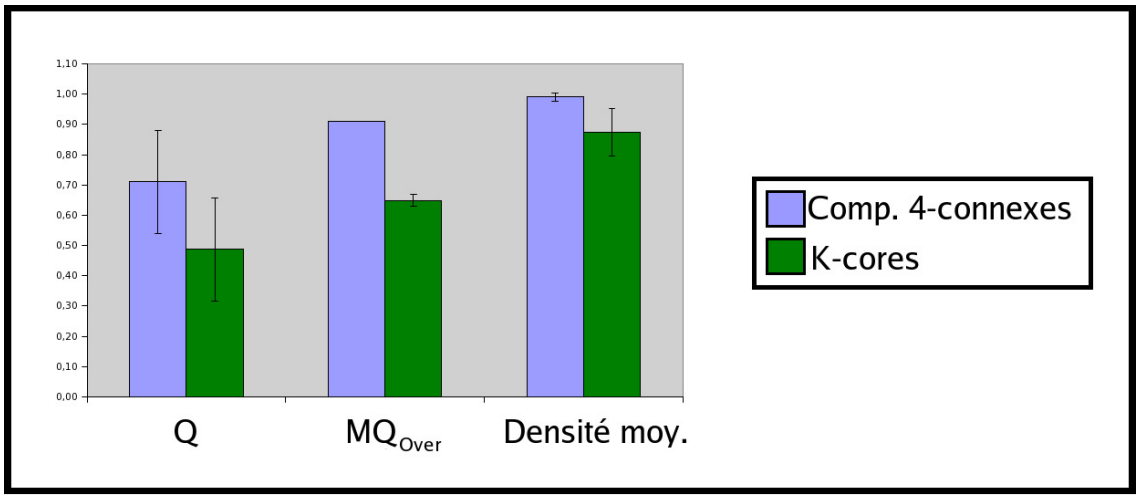

FIG. 4.11: Résultats de l'algorithme K-cores et de la décomposition en composantes 4-connexes sur la collection de sous-graphes extraits des données du concours InfoVis 2007. Ce graphique montre les valeurs moyennes de Q, MQ_{Over} et de la densité moyenne des groupes ainsi que leurs écarts types respectifs.

4.5.2.3 Algorithme topologique

La figure 4.11 montre les résultats de l'algorithme K-cores de Seidman [117] (cf section 3.1.4) et de la décomposition en composantes 4-connexes. Que ce soit pour la mesure Q ou MQ_{Over}, l'algorithme K-cores [117] offre de bien moins bons résultats que la décomposition en composantes 4-connexes. Cet algorithme trouve des décompositions dont la qualité moyenne est $Q = 0.49$ et $MQ_{Over} = 0.65$ avec des écarts types respectifs de 0.14 et 0.16. Seule la densité moyenne des groupes trouvés approche celle de la décomposition en composantes 4-connexes, puisqu'elle est de $\delta = 0.87$ avec un écart type de 0.01.

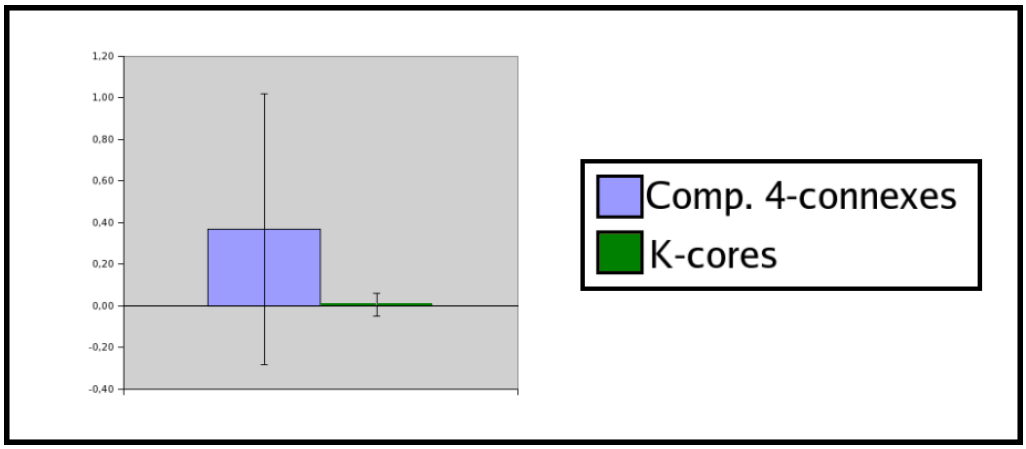

FIG. 4.12: Temps de calcul moyen en secondes ainsi que l'écart type de l'algorithme K-cores et de la décomposition en composantes 4-connexes sur la collection de sous-graphes extraits des données du concours InfoVis 2007.

La figure 4.12 montre les temps de calcul de ces deux algorithmes. Le temps moyen de calcul de l'algorithme K-cores est de 0.01 seconde ce qui est très rapide comparé aux temps de calcul de la décomposition en composantes 4-connexes (mais aussi aux temps de calcul de tous les autres algorithmes testés).

Quand bien même le temps de calcul offert par l'algorithme K-cores est très bon, la qualité des résultats est inférieure à ceux de la décomposition en composantes 4-connexes. D'autre part, le temps de calcul de la décomposition en composantes est suffisamment faible pour que cet algorithme soit utilisable.

4.6 Conclusion

Dans ce chapitre, nous avons d'une part présenté un algorithme permettant de trouver des structures de communauté. L'un des intérêts de cette méthode est qu'elle permet de trouver des communautés chevauchantes et ainsi de détecter les articulations du réseau.

Algorithmes		Avantages	Inconvénients
Agglomératifs	Newman [104]	Valeur de Q	Valeurs de δ et MQ_{Over}
	MQ_agglo	Valeurs de δ et MQ_{Over}	Valeur de Q
Divisifs	Newman & Girvan [105]	Valeur de Q	Temps de calcul
	Strength [13]	Valeurs de δ et MQ_{over}	Valeur de Q
Topologique	K-core [117]	Temps de calcul	Valeurs de Q et MQ_{Over}
Comp. 4-connexes		Valeurs de δ, Q et MQ_{over}	

Tab. 4.1: Avantages et Inconvénients des six algorithmes comparés dans ce chapitre.

Nous avons d'autre part mené une évaluation de notre méthode sur l'un des jeux de données les plus utilisés (provenant des sciences sociales) et comparé les résultats à ceux d'algorithmes bien connus. Pour ce faire, nous introduisons une généralisation de la mesure de qualité MQ aux décompositions chevauchantes de graphe. Nous avons montré que notre méthode permet d'obtenir de bons résultats en terme de temps mais aussi en terme de qualité. En effet, la décomposition en composantes 4-connexes donne des valeurs fortes pour les trois mesures de qualité que nous avons utilisées, c'est-à-dire la densité moyenne des groupes, la modularité et la formule généralisée de MQ. Au contraire, les autres algorithmes évalués ne permettent pas d'avoir des mesures Q et MQ fortes simultanément (cf tableau 4.1).

La décomposition en composantes 4-connexes pourrait être visualisée en utilisant des outils existants tels que ceux de Abello *et al.* [4] et de Archambault *et al.* [9] bien que ces outils soient dédiés à la visualisation de partition de graphe. De plus, elle peut aussi être visualisée par des systèmes permettant la représentation de décomposition de graphe tel que l'outil de Koenig *et al.* [88].

Ce travail offre un certain nombre de perspectives, nous souhaitons notamment mettre en place une nouvelle mesure de qualité adaptée aux décompositions chevauchantes. L'idée principale est de définir formellement les avantages et les inconvénients de la modularité et

de MQ_{Over} (et MQ). Cela devrait permettre de trouver une fonction d'évaluation de décomposition plus efficace. Un autre aspect porte sur l'amélioration des performances d'algorithmes (basés sur Q et MQ) en utilisant la décomposition en composantes 4-connexes comme pré- ou post-traitement.

Chapitre 5

Visualisation de partitionnement de graphes arête-valués : application aux réseaux d'interactions

Nous traitons dans ce chapitre de la visualisation de partitionnement de graphes arête-valués. Ce problème est intéressant puisqu'il pose un certain nombre de contraintes notamment l'identification « facile » de chacun des groupes du partitionnement tout en prenant en compte les valuations des arêtes. Nous nous intéressons particulièrement ici à la visualisation de réseaux d'interactions en biologie, comme les réseaux protéine-protéine (intéractions de protéines pour former de plus grandes molécules protéiques) ou gène-protéine (régulation des gènes par les protéines). Dans un premier temps, nous posons la problématique liée à la visualisation de ce type de données dans la section 5.1, puis nous présentons les travaux existants dans la section 5.2. Dans la section 5.3, nous donnons une vue d'ensemble de notre approche. Nous présentons ensuite quelques prérequis dans la section 5.4. Enfin dans la section 5.5, nous donnons les détails de notre algorithme et présentons des résultats obtenus sur le réseau gène-protéine de la mouche.

5.1 Problématique

La visualisation de partitionnement de graphes est un domaine de recherche actif depuis de nombreuses années [129, 43, 44, 74, 45, 46, 3, 62, 60, 92]. Il n'existe cependant que très peu d'algorithmes capables de prendre en compte les valuations des arêtes. La prise en compte des valuations des arêtes dans une représentation de partitionnement de graphe permet de visualiser simultanément le résultat d'un algorithme de partitionnement tout en conservant une idée des valuations des arêtes.

Dans ce chapitre, nous prendrons l'exemple des réseaux d'interactions gène-protéine. Notre système de visualisation doit pouvoir rendre compte des relations entre gènes et protéines ainsi que leur niveau d'expression pour une puce à ADN donnée[1], résultats d'expériences biologiques. La première difficulté est la taille du réseau. D'autre part, comme

[1] Permet de quantifier l'activité de gènes donnés au cours d'expériences biologiques

on peut le voir sur la figure 5.14.(a), la forte connectivité de tels réseaux produit un effet *« brouillon »* lorsqu'on les dessine avec un algorithme par modèle de forces. De tels problèmes sont fréquents dans les graphes biologiques, en effet nombre d'entre eux sont des graphes *sans échelles*[2]. La propriété sans-échelle des graphes biologiques est actuellement un sujet très discuté dans les communautés de la biologie et de la bioinformatique (pour plus de détails sur ce sujet, le lecteur peut se référer à [94]). Un moyen efficace de visualiser et d'analyser ce type de données est de les partitionner en groupes et si nécessaire de partitionner ces groupes en sous-groupes, et ainsi de suite. Il existe de nombreuses manières de construire de telles partitions multi-échelles de graphes [99, 106, 114].

La difficulté principale dans la visualisation de tels réseaux est de dessiner un graphe en prenant non seulement en compte un arbre de partition (qui le décompose) mais aussi en prenant en compte les valuations des arêtes, c'est-à-dire la force des interactions entre gènes et protéines. Un tel algorithme doit donc respecter les contraintes suivantes :

1. obtenir un dessin dans lequel chaque groupe est entouré par une enveloppe convexe tout en interdisant les chevauchements visuels de ces enveloppes,
2. préserver les inclusions des groupes dans le dessin, pour voir la hiérarchie produite par l'algorithme de partitionnement,
3. respecter les distances valuées dans le graphe en utilisant une fonction de minimisation d'énergie.

Dans ce chapitre, nous expliquons une approche que nous avons présentée dans [20] afin de respecter ces contraintes.

5.2 Etat de l'art

Dans cette section, nous présentons brièvement les travaux portant sur la visualisation de graphe partitionné. Pour plus de détails sur ces techniques, le lecteur peut se référer à la section 3.3.

Dans [43], Eades et Feng décrivent deux représentations pour la visualisation de graphe partitionné : la représentation *plane* et la représentation *multi-niveaux*. Ces techniques sont basées sur l'utilisation d'algorithmes de dessin de graphe planaire [44, 45].

Une deuxième approche consiste à utiliser des algorithmes par modèle de forces pour dessiner les graphes partitionnés (e.g. [129, 74, 46]). La technique présentée par Wang et Miyamoto dans [129] permet de plus d'intégrer un moteur de résolution de contraintes pour améliorer la qualité du dessin final.

Dans [3], Abello *et al.* présentent un système de vues multiples pour visualiser de grands graphes partitionnés : une vue du graphe quotient (d'un niveau donné) et une vue de l'arbre de partition. Le graphe quotient est dessiné par un algorithme par modèle de forces et l'arbre de partition est lui représenté par une *tree-map*. Un système d'interactions permet ensuite de « demander » plus de détails sur une partie d'intérêt tout en conservant le contexte global.

[2] i.e. dont la distribution des degrés suit une loi de puissance.

Dans [60, 4], les auteurs présentent des outils de visualisation de grands graphes basés sur un partitionnement préalable. Le graphe quotient Q_1 (correspondant au plus haut niveau de la hiérarchie) est dessiné en utilisant des algorithmes par modèle de forces. Un système d'interactions permet ensuite d'« ouvrir » (ou de « fermer ») des métanoeuds et donc d'afficher plus ou moins de détails.

Dans [62, 92], les auteurs présentent des algorithmes multi-échelles descendants par modèle de forces permettant de prendre en compte les valuations des arêtes. Ces méthodes consistent à dessiner le graphe quotient Q_1 puis à descendre progressivement jusqu'au plus bas niveau de la hiérarchie. Lorsque le graphe quotient Q_i est dessiné, ces algorithmes calculent le diagramme de Voronoï des positions des sommets de Q_i. La cellule de Voronoï d'un sommet u de Q_i permet ensuite de délimiter la région du plan allouée aux sommets de Q_{i+1} représentés par u.

5.3 Vue d'ensemble de la méthode

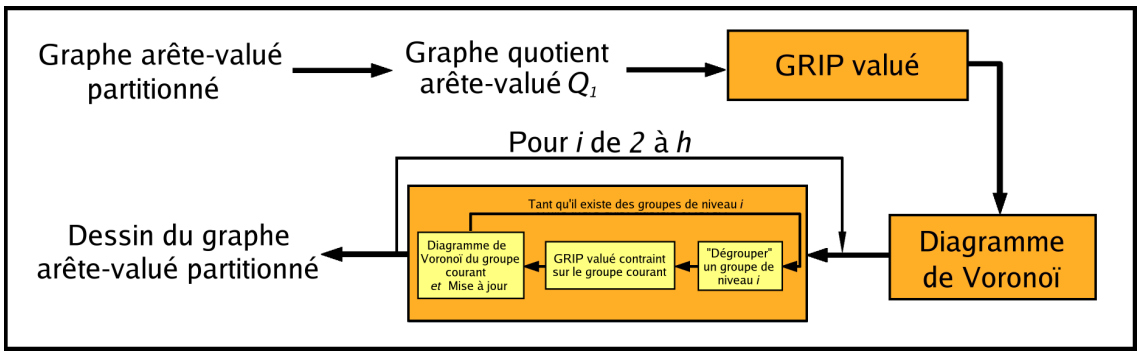

FIG. 5.1: Une vue d'ensemble illustrant toutes les étapes de notre algorithme : du graphe arête-valué partitionné au dessin respectant les contraintes de la section 5.1

.

Dans cette section, nous présentons une vue d'ensemble de notre méthode. La figure 5.3 illustre les différentes étapes de notre algorithme qui est basé sur une technique due à Granitzer *et al.* [62] (cf chapitre 3).

Partant d'un graphe arête-valué partitionné, la première étape consiste à construire le graphe quotient arête-valué correspondant[3]. L'utilisation conjointe d'une version modifiée de [59, 58] prenant en compte les valuations des arêtes, et de diagrammes de Voronoï [50] permet ensuite de calculer les dessins du plus haut niveau de la hiérarchie au plus bas.

La donnée d'entrée de notre algorithme est un graphe partitionné (G, T_G) arête-valué [4] tel que T_G a une hauteur h. La figure 5.2.(a) montre un exemple de graphe et la figure 5.2.(b) montre un arbre de partition possible tel que $h = 3$.

[3]Pour valuer des méta-arêtes, il existe plusieurs méthodes : valeur minimale, maximale, médiane ou encore moyenne des arêtes qu'une méta-arête représente. Ici, nous considérons que la valuation d'une méta-arête est égale à la valuation moyenne des arêtes qu'elle représente.

[4]i.e. G est un graphe arête-valué

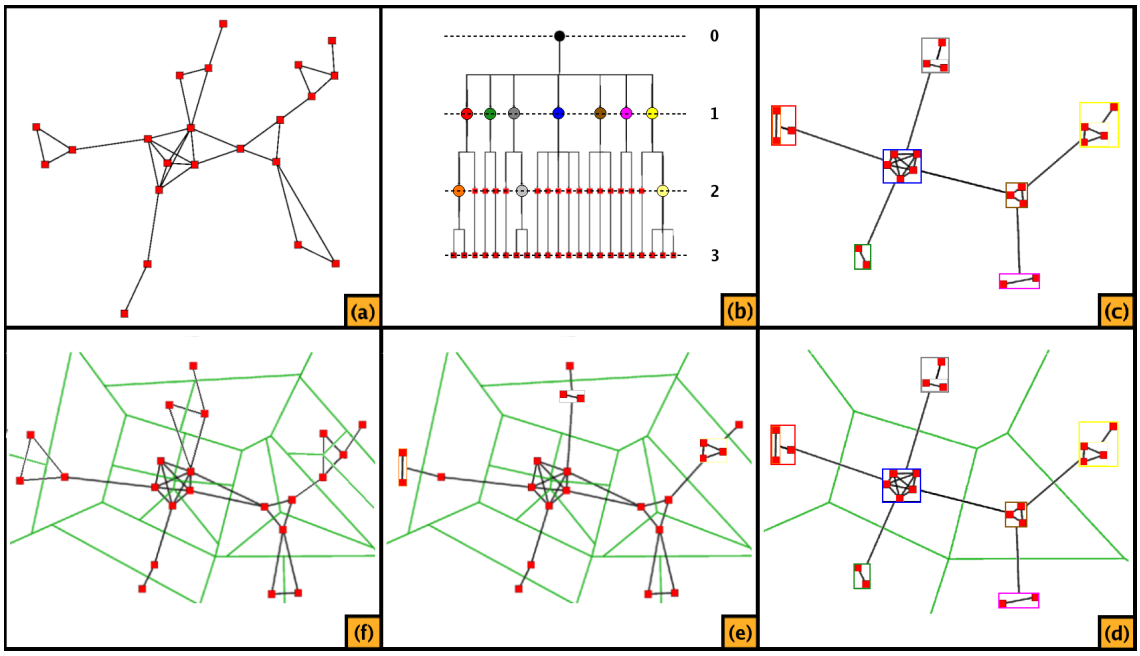

FIG. 5.2: (a)Un exemple de graphe ; (b) Un exemple d'arbre de partition du graphe montré en (a), dans cet exemple la *hauteur* de l'arbre est 3 ; (c) Dessin du graphe quotient Q_1 correspondant au niveau 1 du graphe partitionné présenté en (a) et (b) en utilisant GRIP valué ; (d) Diagramme de Voronoï des positions des sommets du graphe Q_1 ; (e) Dessin du graphe quotient Q_2 ainsi que le diagramme correspondant à ce niveau ; (f) Résultat final.

.

La première étape de notre algorithme consiste à dessiner le graphe quotient Q_1, représentant le niveau 1 de l'arbre de partition. Pour ce faire, nous utilisons un algorithme de dessin par modèle de forces qui prend en compte les valuations des arêtes. Nous obtenons alors un dessin du plus haut niveau d'abstraction du graphe G. la figure 5.2.(c) montre le résultat de cette première étape sur le graphe partitionné des figures 5.2.(a) et (b).

Afin de garantir que les contraintes 1 et 2 (présentées dans la section 5.1) sont respectées, nous devons dessiner tous les noeuds de G représentés par un métanoeud commun de Q_1 dans une région convexe qui ne recoupe aucune autre région. Pour calculer ces régions, nous utilisons des diagrammes de Voronoï [128] (défini dans la section 2.4) des positions des sommets de Q_1. La figure 5.2.(d) montre le diagramme de Voronoï du graphe de la figure 5.2.(c). Ce processus nous permet de connaître précisément les régions convexes dans lesquelles on peut dessiner les noeuds représentés par un métanoeud de Q_1.

Les étapes suivantes de notre approche consistent à dessiner un à un les niveaux i, $1 < i \leq h$ de l'arbre de partition en utilisant le dessin du niveau $i-1$. Soit v un sommet de Q_{i-1} et S l'ensemble des sommets fils de v dans la hiérarchie. Pour dessiner le graphe quotient Q_i, nous dessinons séparément chaque sous-graphe induit par Q dans Q_i, noté $Q_i[S]$. Pour cela, nous utilisons une version contrainte de l'algorithme de dessin par modèle de forces utilisé pour dessiner Q_1. Les sommets de $Q_i[S]$ sont initialement positionnés à la position de v dans le dessin de Q_{i-1}. Lors de l'exécution de l'algorithme, les sommets de $Q_i[S]$ sont contraints dans la cellule de Voronoï de v. Ce processus est répété pour tout

sous-graphe de Q_i[5], puis les nouvelles cellules de Voronoï sont calculées afin de pouvoir appliquer notre technique sur les niveaux inférieurs de la hiérarchie. La figure 5.2.(e) montre le résultat obtenu sur le graphe quotient Q_2. Les positions des sommets de Q_2 seront utilisées pour dessiner Q_3. Enfin, la figure 5.2.(f) montre quant à elle le résultat final de notre approche sur le partitionnement de graphe des figures 5.2.(a) et (b).

Pour résumer, notre approche est similaire à celle de [62]. La différence principale tient au fait que les auteurs de [62] considèrent que les arêtes *inter-groupes* et *intra-groupes* doivent avoir la même influence lors du calcul du dessin (puisque les positions ne sont transformées qu'à la fin du dessin). Au contraire, nous accordons une importance toute particulière à la relation *intra-groupe*, c'est pourquoi les positions sont contraintes à chaque itération de l'algorithme par modèle de forces.

5.4 Prérequis

5.4.1 Algorithme GRIP

Etant donné que nous étendons l'algorithme GRIP [59, 58] aux graphes arête-valués, nous donnons dans cette partie les détails de cet algorithme. Celui-ci compte deux étapes principales : la première consiste à construire une hiérarchie dans les sommets et la seconde utilise cette hiérarchie pour calculer le positionnement des sommets. Cet algorithme ne permet pas de prendre en compte les valuations des arêtes. Dans cette partie, nous considérons donc que le graphe n'est pas arête-valué.

5.4.1.1 Echantillonnage par ensembles indépendants maximaux

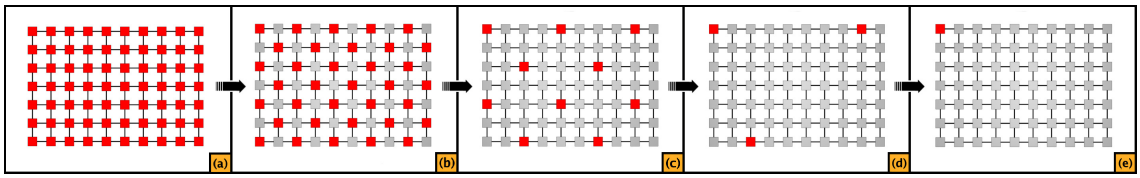

FIG. 5.3: (a) Un exemple de graphe contenant 70 sommets et 123 arêtes ; (b) Les sommets en rouge représentent l'ensemble maximal V_1 de sommets à distance au moins 2 ; (c) Les sommets en rouge représentent l'ensemble maximal V_2 de sommets à distance au moins $2^2 = 4$; (d) Les sommets en rouge représentent l'ensemble maximal V_3 de sommets à distance au moins $2^2 = 8$; (e) Les sommets en rouge représentent l'ensemble maximal V_4 de sommets à distance au moins $2^2 = 16$.

La première étape de l'algorithme de [59, 58] consiste à calculer un échantillonnage par ensembles indépendants maximaux $\mathcal{V}$ (*Maximal Independent Set Filtration*, noté *MISF*) de l'ensemble des sommets V du graphe. On définit $\mathcal{V}$ comme suit, $\mathcal{V}= \{V_i\}_{0 \leq i \leq k}$ avec $V = V_0 \supset V_1 \supset ... \supset V_k$, $|V_k| \leq 3$ et $\forall 0 < i < k$, V_i est un sous-ensemble maximal de V_{i-1} tel que $\forall u, v \in V_i$, $dist_G(u, v) \geq 2^i$. La figure 5.3 illustre cet échantillonnage : la

[5] représenté par un métanoeud de Q_{i-1}

figure 5.3.(a) montre un graphe dont est extrait un ensemble maximal V_1 de sommets à distance au moins 2 (figure 5.3.(b)), puis un ensemble maximal V_2 de sommets à distance au moins 2^2 (figure 5.3.(c)), puis V_3 (figure 5.3.(d)) et enfin $V_4 = V_k$ (figure 5.3.(e)).

Il existe plusieurs moyens de construire le *MISF* d'un graphe. Une technique simple consiste à répéter pour tout i de 1 à k :

1. Soit $V^* = V_{i-1}$
2. Choisir aléatoirement un sommet u de V^*.
3. Enlever le sommet u de V^* ainsi que tous les sommets de V^* à distance au plus $2^i - 1$, et ajouter u à V_i.
4. Revenir à l'étape 2 tant qu'il reste des sommets dans V^*, sinon incrémenter i et revenir à l'étape 1.

Prendre pour chaque V_i, un ensemble maximal de sommets à distance au moins 2^i permet d'obtenir une bonne distribution des sommets dans le graphe et par conséquent d'avoir de « bons » représentants.

5.4.1.2 Placement

Dans la seconde étape de l'algorithme de [59, 58], le processus réalise le placement des sommets du graphe. L'algorithme positionne les sommets de chaque ensemble de $\mathcal{V}$ du plus haut niveau V_k de l'échantillonnage au plus bas, V_0. Plus précisément, lorsque le niveau i est traité, seuls les sommets de V_i qui n'ont pas encore été positionnés sont pris en compte. Pour chaque niveau, cette étape de placement peut être décomposée en deux phases : un positionnement initial intelligent (i.e. les sommets sont positionnés « proches » de leur position finale), puis une phase pendant laquelle le dessin est amélioré grâce à un algorithme par modèle de forces.

Placement initial intelligent

Lors de cette phase de placement initial, l'algorithme fait la distinction entre le placement de l'ensemble V_k et le placement de tout autre V_i ($i < k$).

Placement initial de V_k : Les auteurs de [59, 58] considèrent que l'ensemble de V_k contient exactement 3 sommets. Si ce n'est pas le cas, des sommets de V_{k-1} sont ajoutés à V_k. Soient v_1, v_2 et v_3 ces trois sommets, alors v_1, v_2 et v_3 sont positionnés sur un triangle tel que :

$$\begin{cases} dist_{\mathbb{R}}(v_1, v_2) = d_G(v_1, v_2) \\ dist_{\mathbb{R}}(v_1, v_3) = d_G(v_1, v_3) \\ dist_{\mathbb{R}}(v_2, v_3) = d_G(v_2, v_3) \end{cases} \quad (1)$$

où $dist_{\mathbb{R}}(u, v)$ est la distance euclidienne entre u et v.

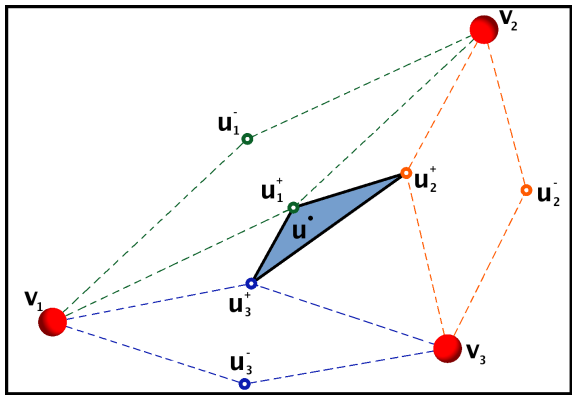

FIG. 5.4: Positionnement initial d'un sommet u de V_i, $1 \leq i < k$. Les sommets v_1, v_2 et v_3 sont ses trois plus proches voisins déjà positionnés. Les points u_1^-, u_1^+, u_2^-, u_2^+, u_3^- et u_3^+ correspondent aux solutions des systèmes d'équations ci-dessous.

Placement initial de V_i***,*** $i < k$ ***:*** Pour tout autre niveau i, $0 \leq i < k$, l'algorithme ne considère que les sommets de V_i qui n'ont pas encore été positionnés. Soient $u \in V_i \setminus V_{i+1}$, et v_1, v_2 et v_3 les trois sommets les plus proches de u déjà positionnés (en terme de distance dans le graphe). Le placement initial de u est calculé en résolvant les systèmes suivants :

$$\begin{cases} dist_{\mathbb{R}}(u, v_1) = d_G(u, v_1) \\ dist_{\mathbb{R}}(u, v_2) = d_G(u, v_2) \end{cases} \tag{2}$$

$$\begin{cases} dist_{\mathbb{R}}(u, v_1) = d_G(u, v_1) \\ dist_{\mathbb{R}}(u, v_3) = d_G(u, v_3) \end{cases} \tag{3}$$

$$\begin{cases} dist_{\mathbb{R}}(u, v_2) = d_G(u, v_2) \\ dist_{\mathbb{R}}(u, v_3) = d_G(u, v_3) \end{cases} \tag{4}$$

Pour ces systèmes d'équations, il existe au plus six solutions (deux par système d'équations) notées u_1^-, u_1^+, u_2^-, u_2^+, u_3^- et u_3^+. Le sommet u est positionné au barycentre des trois solutions les plus proches (les solutions u_1^+, u_2^+ et u_3^+ dans la figure 5.4).

Ce placement initial est dit « intelligent » puisque l'utilisation des distances dans le graphe permet de positionner les sommets proches de leur position finale. Afin d'améliorer les positions des sommets de V_i, $0 \leq i \leq k$, une étape d'affinement des positions est ensuite effectuée.

Affinement des positions : utilisation d'algorithmes par modèle de forces

Afin d'améliorer le positionnement des sommets, les auteurs de [59, 58] utilisent des algorithmes de dessin par modèle de forces. Les distinction est faite ici entre les ensembles V_i, $1 \leq i \leq k$, et l'ensemble V_0.

Affinements des positions des sommets de V_i**,** $1 \leq i \leq k$ **:** Pour chaque niveau i, $1 \leq i \leq k$, l'algorithme de dessin utilisé est celui de Kamada et Kawai [79] (cf section 3.2.3). A chaque itération de cet algorithme les forces exercées sur un sommet v de $V_i \setminus V_{i+1}$ sont calculées de la manière suivante :

$$\overrightarrow{F_{KK}}(v) = \sum_{u \in V} \left(\frac{dist_{\mathbb{R}^2}(u,v)}{dist_G(u,v) \cdot edgeLength^2} - 1 \right) (pos(u) - pos(v)) \tag{5}$$

où $pos(u)$ est la position de u dans le plan et $edgeLength$ est la longueur « idéale » d'une arête. Cet algorithme prend donc en compte les distances dans le graphe lors du calcul des forces exercées sur un sommet. Afin de limiter la complexité en temps, le nombre de sommets pris en compte lors du calcul des forces exercées sur le sommet v est limité aux $\frac{deg_{avg}(G) \cdot |V|}{|V_i|}$ plus proches dans V_i.

Affinements des positions des sommets de V_0 **:** Pour affiner les positions des sommets de $V_0 \setminus V_1$, l'algorithme de [59, 58] utilise l'algorithme de Fruchterman et Reingold[55] (cf section 3.2.3). Les forces exercées sur chaque sommet v de $V_0 \setminus V_1$ sont calculées comme suit :

$$\begin{aligned} \overrightarrow{F_{FR}}(v) = & \sum_{u \in Adj(v)} \frac{dist_{\mathbb{R}^2}(u,v)^2}{edgeLength^2} (pos(u) - pos(v)) \\ & + \sum_{u \in V, u \neq v} s \frac{edgeLength^2}{dist_{\mathbb{R}^2}(u,v)^2} (pos(v) - pos(u)) \end{aligned} \tag{6}$$

où s est une « petite » constante, et $Adj(v)$ est l'ensemble des sommets adjacents à v. Encore une fois, pour limiter la complexité en temps, les forces répulsives exercées sur v (deuxième partie de la somme) sont calculées en ne prenant en compte qu'un certain nombre de sommets les plus proches de v dans V_0. On voit clairement que cet algorithme n'utilise pas les distances dans le graphe pour calculer les forces exercées sur un sommet, cela permet d'augmenter les performances en temps de [59, 58]. En effet, le calcul de distances dans un graphe est coûteux en temps et/ou en espace.

5.4.2 Diagramme de Voronoï

Comme décrit dans la section 2.4, le diagramme de Voronoï d'un ensemble P de n points du plan (sites) est une subdivision du plan en n régions (cellules). Pour chaque site p de P, la cellule de p contient tous les points au moins aussi proches de p que de tout autre point de P.

Dans [50], Fortune donne un algorithme optimal pour calculer le diagramme de Voronoï d'un ensemble de sommets du plan. Cet algorithme calcule le diagramme de Voronoï de n points du plan en temps $O(nlog(n))$ et en espace $O(n)$. Le diagramme est construit en utilisant des *cônes d'influences*[6] et un plan de balayage. Cette technique consiste à calculer les intersections des cônes d'influence pour déterminer les intersections des arêtes de chaque cellule (qui est un polygone). La figure 5.6 illustre cette technique.

[6] Cône défini par un point p du plan, d'axe Δ parallèle à l'axe (Oz) et passant par p, et d'angle $\pi/4$.

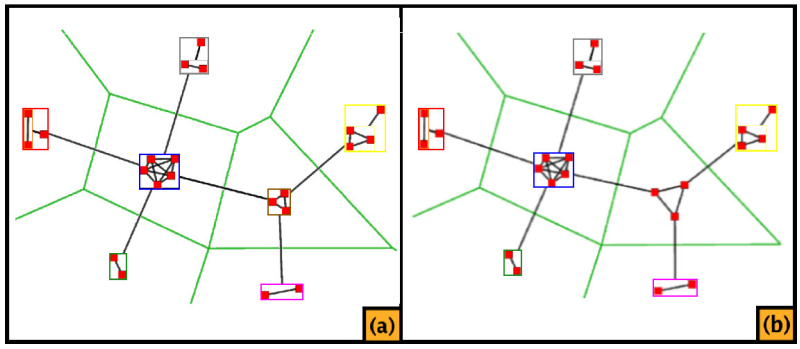

FIG. 5.5: (a) En vert, le diagramme de Voronoï des positions des sommets du graphe Q_1 de la figure 5.2.(c) ; (b) Résultat après le dessin du sous-graphe correspondant au métanoeud 1 de la figure (a).

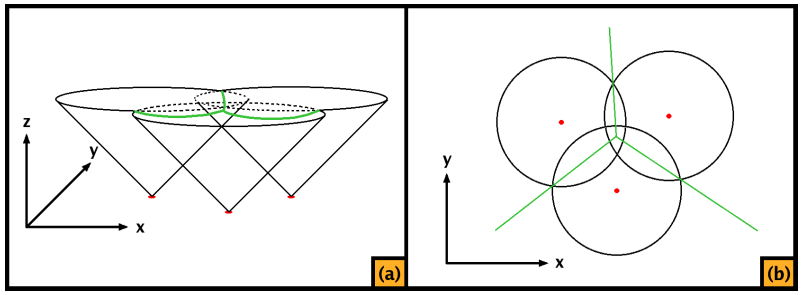

FIG. 5.6: Illustration de l'algorithme de Fortune [50] sur un ensemble de trois points du plan (en rouge). (a) Les cônes d'influences des trois points du plan, en vert les intersections de ces cônes ; (b) Projection sur le plan (Oxy), les demi-droites en vert représentent le diagramme de Voronoï des trois points du plan.

5.5 Algorithme

Dans cette partie, nous expliquons plus en détail les étapes principales de notre approche. Nous présentons tout d'abord les adaptations apportées à l'algorithme GRIP [59, 58] pour qu'il prenne en compte les valuations des arêtes. Puis nous expliquons le processus de dessin de notre approche, nous verrons ensuite un post-traitement permettant d'augmenter la lisibilité de la représentation. Enfin, nous montrons les résultats obtenus sur le réseau gène-protéine de la mouche.

5.5.1 GRIP valué

5.5.1.1 Echantillonnage par ensembles indépendants maximaux d'un graphe arête-valué

Calculer directement le *MISF* sur un graphe arête-valué en utilisant les distances valuées dans le graphe n'a pas de sens. En effet, la méthode d'échantillonnage de l'algorithme GRIP [59, 58] permet d'obtenir de « bons représentants » dans un graphe non-valué, cependant l'idée même de « bon représentant » change lorsque l'on considère les valuations

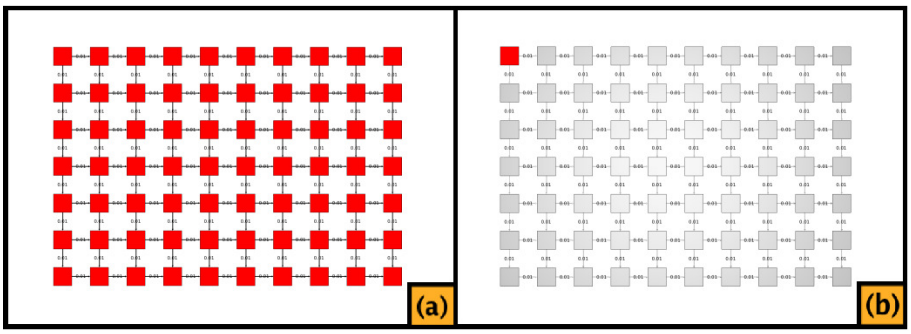

FIG. 5.7: (a) Graphe de la figure 5.3 pour lequel on considère maintenant les valuations des arêtes. Ces valuations sont toutes égales à 0.01 ; (b) Le *MISF*, $\mathcal{V}$ a été calculé sur le graphe en utilisant les distances valuées. L'ensemble V_1 de $\mathcal{V}$ ne contient qu'un seul sommet, en rouge dans le dessin.

des arêtes. La figure 5.7 montre un exemple où le graphe de la figure 5.3 est valué par de « petites » valeurs (e.g. 0.01). Il n'existe dans ce graphe aucune paire de sommets à distance valuée supérieure à 2. On obtient donc un très mauvais échantillonnage des sommets puisqu'alors $\mathcal{V}= \{V_0, V_1\}$ et $|V_1| = 1$.

Une deuxième approche pourrait consister à calculer l'ensemble indépendant sans prendre en compte les valuations des arêtes. Dans ce cas, nous aurions le problème « inverse » : certains sommets pourraient être les représentants d'autres sommets bien que ceux-ci soient à une grande distance valuée dans le graphe. On obtiendrait donc un mauvais échantillonnage des sommets.

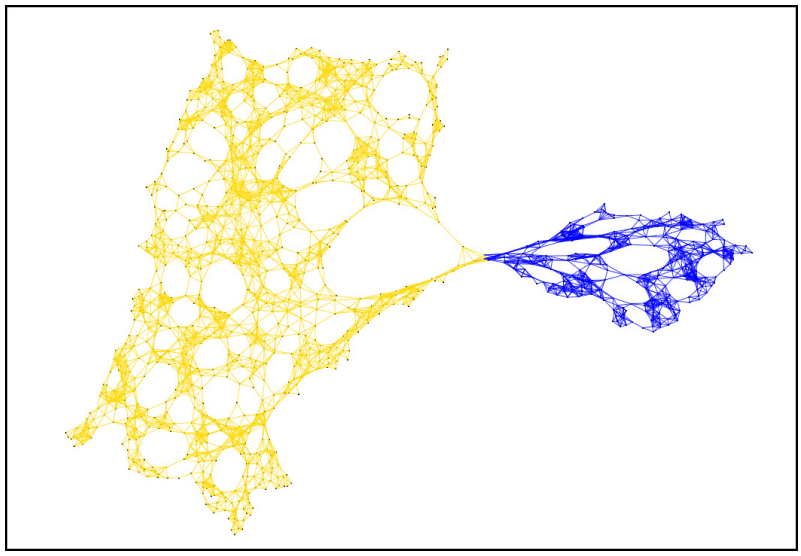

FIG. 5.8: Exemple de graphe arête-valué, la partie « gauche » du graphe (en jaune) ne contient que des arêtes faiblement valuées tandis que la partie « droite » (en bleu) ne contient que des arêtes fortement valuées. Pour obtenir un bon échantillonnage des sommets de ce graphe, il ne faudrait pas utiliser la même distance pour les deux parties du graphe.

Une autre approche pourrait consister à trouver pour chaque V_i la distance permettant d'obtenir un bon échantillonnage. Il semble cependant que trouver, pour chaque V_i, une distance *unique* qui permettrait d'avoir de bons représentants n'est pas possible. La figure 5.8 montre un graphe arête-valué dont les valuations des arêtes de la partie « gauche » sont faibles tandis que celles de la partie « droite » sont fortes. Afin d'obtenir un bon échantillonnage, la distance utilisée pour filtrer les sommets ne doit pas être la même pour la partie « gauche » et pour la partie « droite ».

La solution optimale à ce problème est de trouver pour chaque graphe et pour chaque V_i, les distances permettant d'optimiser la distribution des sommets dans les niveaux. Trouver ces distances reste un problème ouvert. Afin d'échantillonner les sommets du graphe, nous utilisons une approche permettant de se repositionner dans le cas non-valué.

Notre méthode consiste à calculer le *MISF* sans prendre en compte les valuations des arêtes sur un arbre couvrant du graphe. Lors de la construction de l'arbre couvrant, les arêtes « peu » importantes du graphe sont exclues. Une arête « peu » importante est typiquement une arête dont la valuation est forte. Supposons que les sommets u et v soient reliés par une de ces arêtes dans le graphe original, si l'arête $\{u, v\}$ est présente dans l'arbre couvrant, le sommet u pourrait être le représentant de v dans le *MISF* bien que u ne soit pas représentatif de v. Malgré l'exemple précédent, toutes les arêtes fortes ne doivent pas être retirées, certaines de ces arêtes étant centrales et les liens entre leurs extrémités importants (e.g nous souhaitons conserver la connexité du graphe). Ceci revient à faire un compromis entre valuation des arêtes et topologie du graphe original.

Calcul de l'arbre couvrant

Un arbre t-couvrant est un arbre couvrant qui approxime les distances (valuées ou non) dans le graphe par un facteur t (*cf.* section 2.2). De nombreux résultats ont été trouvés sur les arbres t-couvrant (pour un résumé de ces travaux le lecteur peut se référer à [28, 29]). Notamment, il a été montré que pour tout $t > 1$, le problème qui consiste à déterminer l'existence d'un arbre t-couvrant dans un graphe arête-valué est *NP-complet*. Etant donnée la complexité de ce problème, nous devons donc utiliser une méthode heuristique.

Afin d'obtenir un arbre permettant d'abstraire le mieux possible le graphe original, nous cherchons donc un arbre couvrant tel que les distances dans l'arbre soient aussi proches que possible des distances valuées dans le graphe : la distance dans l'arbre entre deux noeuds doit donc être aussi petite que possible. D'autre part, cet arbre doit aussi prendre en compte la topologie du graphe.

Pour prendre en compte la topologie du graphe, nous utilisons une métrique sur les arêtes : la métrique *Strength* [13, 32]. Cette métrique permet de quantifier l'importance d'une arête en fonction de la cohésion de son voisinage. Pour plus de détails, le lecteur peut se référer à la section 3.1.2.

L'intuition que nous avons est que moins de cycles (de tailles 3 ou 4) passent par une arête, plus cette arête est susceptible d'appartenir à des plus courts chemins. Par exemple, un *isthme* e (i.e. une arête telle que $G' = (V, E \setminus \{e\})$ n'est pas connexe) a une valeur de métrique Strength égale à zéro, or e est contenue dans tous les plus courts chemins reliant deux sommets des deux composantes connexes de $G' = (V, E \setminus \{e\})$.

Algorithme de Kruskal [91] : L'algorithme de Kruskal [91] est un algorithme bien connu qui calcule un arbre couvrant de poids minimum en temps $O(|E| \cdot log(|V|))$. Cet algorithme consiste à agglomérer petit à petit des sous-arbres du graphe jusqu'à obtenir un arbre couvrant. Plus précisément, l'algorithme construit tout d'abord une pile d'arêtes triées par ordre croissant de valuation. Puis, partant d'un état où chaque sommet est considéré comme un sous-arbre du graphe, l'algorithme insère l'arête la plus faible restant dans la pile si elle « relie » deux sous-arbres distincts et agglomère ses sous-arbres en un seul. Cette arête est ensuite enlevée de la pile. Lorsque toutes les arêtes ont été parcourues ou qu'il ne reste plus qu'un seul sous-arbre, un arbre couvrant de poids minimum est construit et l'algorithme s'arrête.

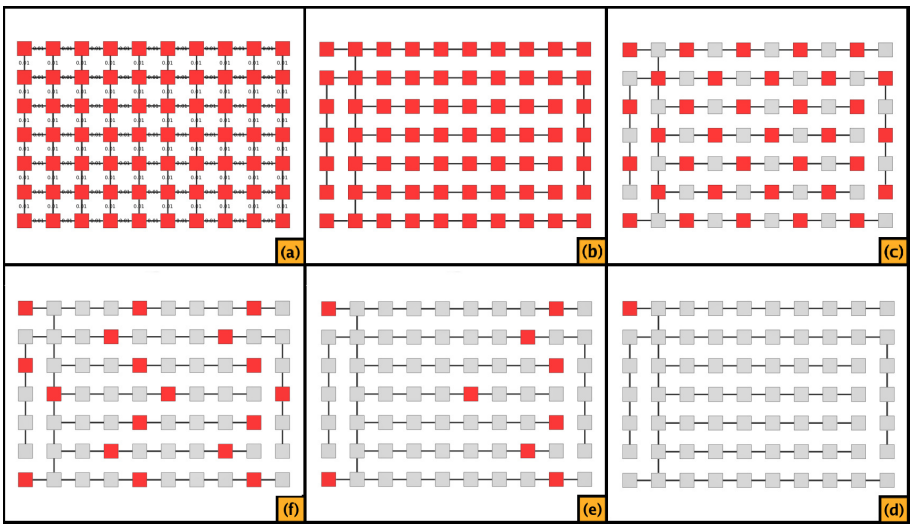

FIG. 5.9: (a) Exemple du graphe arête-valué de la figure 5.8 ; (b) Arbre couvrant calculé en utilisant notre approche ; (c), (d), (e) et (f) Les sommets en rouge indiquent respectivement les ensembles V_1, V_2, V_3 et V_4.

Comme décrit précédemment, nous voulons faire un compromis entre topologie du graphe et valuation des arêtes. Pour ce faire, nous commençons par calculer une nouvelle valuation w' des arêtes décrite dans [6] et définie comme suit :

$$\forall e \in E : w'(e) = w(e) \cdot w_s(e) \tag{7}$$

où $w(e)$ est la valuation de l'arête e et $w_s(e)$ sa valeur de Strength. Enfin, notre arbre couvrant, noté A est obtenu en appliquant l'algorithme de Kruskal sur notre graphe et en considérant la valuation w'. Finalement, nous calculons le *MISF* sur l'arbre couvrant A sans prendre en compte la valuation des arêtes. La figure 5.9 montre le résultat de cette approche sur l'exemple de la figure 5.8.(a). On obtient dans cet exemple un *MISF* $\mathcal{V}= \{V_0, V_1, V_2, V_3, V_4\}$ avec $|V_1| = 35$, $|V_2| = 18$, $|V_3| = 9$ et $|V_4| = 1$.

On peut noter que d'autres métriques pourraient être utilisées pour prendre en compte la topologie du graphe, notamment l'inverse de la « *Betweenness Centrality* » [53, 24, 103], l'indice de Jaccard [75] ou encore toute autre métrique mettant en évidence la topologie du graphe.

5.5.1.2 Placement des sommets

Nous présentons ici comment adapter l'étape de placement de l'algorithme GRIP [59, 58] afin de prendre en compte les valuations des arêtes.

Le positionnement initial

Nous avons vu que pour effectuer le positionnement initial des sommets, les auteurs de [59, 58] résolvent les systèmes d'équations (1) et (2-4). Pour prendre en compte les valuations des arêtes le placement initial de V_k se fait en résolvant le système suivant :

$$\begin{cases} dist_{\mathbb{R}}(v_1, v_2) = d_{G,w}(v_1, v_2) \\ dist_{\mathbb{R}}(v_1, v_3) = d_{G,w}(v_1, v_3) \\ dist_{\mathbb{R}}(v_2, v_3) = d_{G,w}(v_2, v_3) \end{cases} \tag{8}$$

où $dist_{G,w}(u, v)$ est la distance valuée dans le graphe entre les sommets u et v. Et le positionnement de tout autre V_i :

$$\begin{cases} dist_{\mathbb{R}}(u, v_1) = d_{G,w}(u, v_1) \\ dist_{\mathbb{R}}(u, v_2) = d_{G,w}(u, v_2) \end{cases} \tag{9}$$

$$\begin{cases} dist_{\mathbb{R}}(u, v_1) = d_{G,w}(u, v_1) \\ dist_{\mathbb{R}}(u, v_3) = d_{G,w}(u, v_3) \end{cases} \tag{10}$$

$$\begin{cases} dist_{\mathbb{R}}(u, v_2) = d_{G,w}(u, v_2) \\ dist_{\mathbb{R}}(u, v_3) = d_{G,w}(u, v_3) \end{cases} \tag{11}$$

L'adaptation aux graphes arête-valués se fait donc naturellement en considérant les distances valuées dans le graphe en lieu et place des distances non-valuées.

Affinement des positions

Comme décrit dans la section 5.4.1.2, l'algorithme GRIP [59, 58] utilise des algorithmes par modèle de forces pour affiner le positionnement des sommets. A chaque itération de ces algorithmes par modèle de forces, le temps de calcul est accéléré en ne prenant en compte qu'un nombre restreint de plus proches voisins lors du calcul des forces exercées sur un sommet. Dans un graphe non-valué, trouver les plus proches voisins d'un sommet u peut être fait simplement en effectuant un parcours en largeur du graphe (*BFS*) depuis le sommet u. Lorsque l'on considère que le graphe est arête-valué, un *BFS* ne permet pas de trouver les plus proches voisins.

Pour résoudre ce problème, il existe deux approches : la première consiste à calculer les distances entre toutes les paires de sommets (problème appelé *all-pair shortest paths problem*, *APSPP*), la seconde consiste à calculer n fois la distance entre un sommet et tous les autres (problème connu sous le nom de *single-source shortest paths problem*, *SSSPP*). Les algorithmes actuels permettant de calculer les distances entre toutes les paires de sommets [82, 100] ont une complexité en temps de $O(nm+n^2\log(n))$ et en espace de $O(n^2)$. Pour ce qui est de la deuxième approche, les algorithmes de [52, 38] permettent de calculer les distances entre un sommet et tous les autres sommets en temps $O(m + n\log(n))$. Répéter n fois, cela conduit à une complexité en temps de $O(nm + n^2\log(n))$. L'avantage

de cette méthode est que l'on peut ne pas stocker les distances entre toute paire de sommets mais seulement les distances entre chaque sommet et ses N voisins les plus proches. On a donc une complexité en espace de $O(N \cdot n)$. Pour plus de détails sur les problèmes *APSPP* et *SSSPP*, le lecteur peut se référer à [134].

Pour calculer les plus proches voisins de chaque sommet, nous préférons la solution, moins coûteuse en espace, consistant à calculer n fois les N plus proches voisins d'un sommet. Nous utilisons l'algorithme bien connu de Dijkstra [35] pour calculer les plus courts chemins dans un graphe arête-valué. Dans la mesure où nous ne recherchons que les N plus proches voisins de chaques sommets, nous pouvons ne maintenir à jour qu'une liste triée de N sommets. Ceci permet de réduire la complexité du problème *SSSPP* à $O(nm+n^2\log(N))$ en temps et à $O(N \cdot n)$ en espace. Le coût de la recherche des N voisins les plus proches d'un sommet dans l'arbre couvrant A est donc en temps $O(n\log(N))$ et en espace $O(N)$. Appliqué sur chaque sommet de l'arbre couvrant A, cela permet d'obtenir une « approximation » des distances et des plus proches voisins nécessaire pour prendre en compte la valuation des arêtes avec une complexité en temps de $O(n^2\log(N))$ et une complexité en espace de $O(N \cdot n)$.

5.5.2 Processus de dessin

Comme décrite dans la section 5.3, notre approche consiste à dessiner un à un les niveaux de la hiérarchie de manière descendante (i.e. du niveau 1 au niveau h). Dans cette partie, nous présentons comment notre approche dessine le premier niveau de la hiérarchie, puis comment tout autre niveau i est dessiné.

5.5.2.1 Dessin du niveau 1 de la hiérarchie

Le calcul du dessin du premier niveau de la hiérarchie se fait en deux étapes : la première étape permet le positionnement des sommets de Q_1 et la deuxième permet de respecter la contrainte 1 de la section 5.1.

Dessin du graphe quotient Q_1

Pour dessiner le graphe quotient Q_1 correspondant au premier niveau de la hiérarchie, nous utilisons la version modifiée de GRIP présentée dans la section 5.5.1. Cet algorithme permet de dessiner le graphe quotient en prenant en compte les valuations de ses arêtes. Etant donné que les valuations des méta-arêtes sont représentatives des valeurs des arêtes qu'elles représentent, le positionnement relatif des métanoeuds de Q_1 est une bonne « approximation » des relations liant les sommets qu'ils représentent.

Calculs des régions

La deuxième étape du dessin du premier niveau de la hiérarchie consiste à calculer le diagramme de Voronoï des positions des métanoeuds (et noeuds) de Q_1. Soit v un métanoeud de Q_1, on note S l'ensemble des sommets fils de v dans la hiérarchie. La cellule de Voronoï de v est la région dans laquelle les sommets de $Q_2[S]$ seront dessinés.

Par exemple, dans la figure 5.5.(a), le graphe représenté par le métanoeud 1 sera dessiné dans la cellule du métanoeud 1 (voir la figure 5.5.(b)).

Comme les cellules de Voronoï ne se recouvrent pas[7], l'utilisation du diagramme de Voronoï permet de garantir qu'il n'y aura pas de recouvrement entre les groupes représentés par les métanoeuds de Q_1 (cf contrainte 1 de la section 5.1).

5.5.2.2 Dessin du niveau i de la hiérarchie

Dessin du graphe quotient Q_i

Une fois dessiné le graphe quotient Q_1, l'algorithme descend progressivement pour dessiner les graphes Q_i, $1 < i \leq h$. Le processus utilisé pour dessiner Q_i est répété pour chaque niveau de la hiérarchie, c'est-à-dire, jusqu'à ce que le graphe entier ait été dessiné.

Pour dessiner le graphe quotient Q_i correspondant au niveau i de la hiérarchie, nous utilisons le dessin du niveau précédent (i.e. le dessin du graphe Q_{i-1}). Chaque sous-graphe induit $Q_i[S]$ de Q_i (représenté par un métanoeud v dans Q_{i-1}) est dessiné individuellement.

La première étape pour dessiner le graphe $Q_i[S]$ consiste à positionner ses sommets aux coordonnées du métanoeud v qui le représente dans Q_{i-1}. L'algorithme de dessin que l'on utilise ensuite est une version contrainte et modifiée de l'algorithme présenté dans la section 5.5.1.

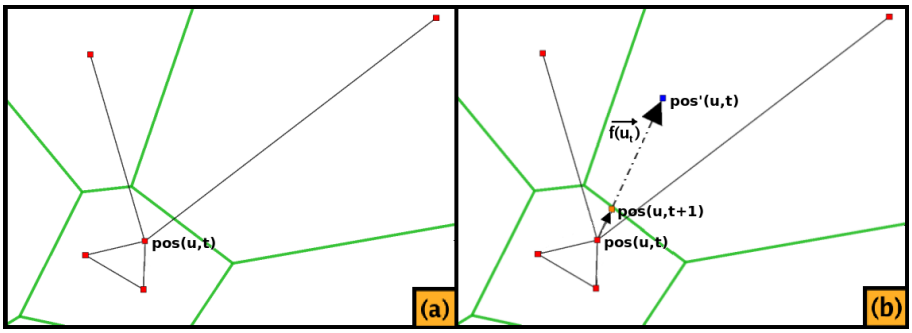

FIG. 5.10: (a) Dessin du sous-graphe $Q_i[S]$ situé dans la cellule centrale à l'itération t, u_t indique la position du noeud u à l'itération t ; (b) Pour calculer la force totale exercée sur u à l'itération t, un nombre constant de voisins dans Q_i sont pris en compte. u'_{t+1} indique la position que devrait avoir u à l'itération $t+1$ si l'on ne forçait pas les sommets du graphe $Q_i[S]$ à rester dans sa cellule. La position de u_{t+1} est trouvée en calculant l'intersection entre la cellule et le segment $[u_t u'_{t+1}]$.

La première de ces modifications concerne la contrainte 1 puisqu'elle permet de contraindre le positionnement des sommets de $Q_i[S]$ dans la cellule de v. On note $pos(u,t)$ la position du sommet u au début de l'itération t de l'algorithme par modèle de forces. Considérons qu'à l'itération t, le sommet u doit être déplacé à la position $pos'(u,t)$. Si $pos'(u,t)$ est

[7] A l'exclusion de leurs frontières

dans la cellule de $Q_i[S]$ (de v), alors $pos(u, t+1) = pos'(u, t)$. Sinon le sommet u est positionné à l'intersection de la cellule et du segment $[pos(u, t), pos'(u, t)]$ (voir la figure 5.10). Contrairement aux auteurs de [62], nous préférons donner plus d'importance aux arêtes *intra-groupes* qu'aux arêtes *inter-groupes*. C'est pourquoi la contrainte 1 est respectée à chaque itération de l'algorithme par modèle de forces.

La seconde modification permet d'améliorer la qualité du dessin en limitant le nombre d'arêtes excessivement longues. Lorsque les forces exercées sur un sommets de $Q_i[S]$ sont calculées, l'algorithme prend en compte un nombre constant de sommets de Q_i qui n'appartiennent pas à $Q_i[S]$ (voir la figure 5.10).

Calculs des sous-régions

Une fois tous les sous-graphes $Q_i[S]$ dessinés, l'algorithme calcule le diagramme de Voronoï des sommets de chacun des graphes $Q_i[S]$. Cela permet de garantir que chaque sous-graphe correspondant à un métanoeud dans $Q_i[S]$ est dessiné dans une région convexe et sans chevauchement avec d'autres sous-graphes représentés par un métanoeud dans $Q_i[S]$. Dans la figure 5.11.(a), les lignes bleues correspondent au diagramme de Voronoï du sous-graphe représenté par le métanoeud 1 dans la figure 5.5.(a).

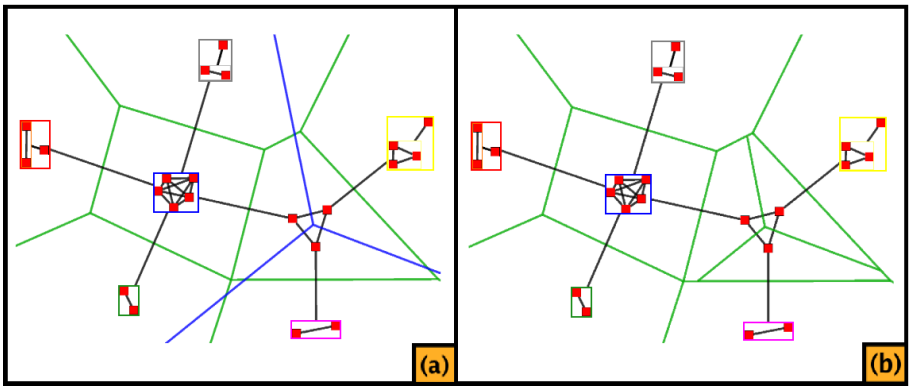

FIG. 5.11: (a) En bleu, le diagramme de Voronoï du sous-graphe représenté par le métanoeud 1 dans la figure 5.2.(a) ; (b) Les régions des métanoeuds (ici, de simples sommets) du sous-graphe sont le résultat de l'intersection du diagramme de Voronoï et de la cellule du métanoeud 1.

Enfin, la dernière étape consiste à calculer l'intersection du diagramme Voronoï du sous-graphe $Q_i[S]$ et de la région du métanoeud v (voir la figure 5.11.(b)). Cela permet de garantir les contraintes 1 et 2. En effet, les sous-graphes représentés par un métanoeud dans $Q_i[S]$ ne pourront pas chevaucher les régions des autres métanoeuds de Q_i. D'autre part, leur positionnement est ainsi contraint dans la région de v.

5.5.3 Post-traitement : Regroupement d'arêtes

La dernière étape de notre algorithme est basée sur le travail de Holten [72]. Dans cet article, l'auteur utilise une vue standard de l'arbre de partition (i.e. un algorithme

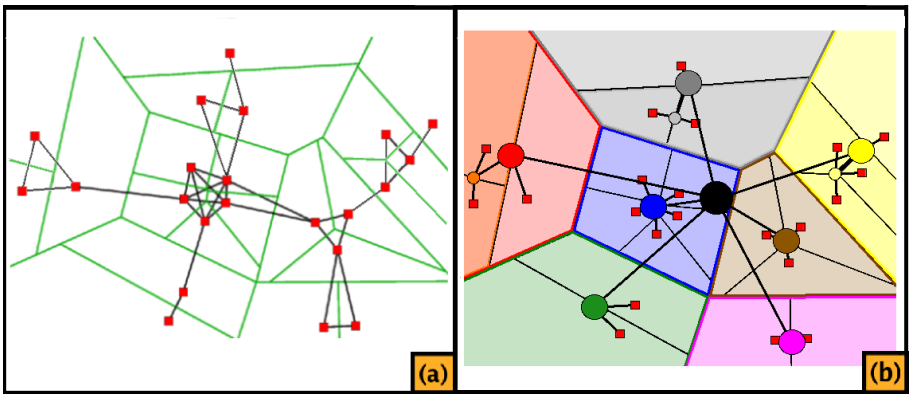

FIG. 5.12: (a) Positionnement final des sommets du graphe partitionné de la figure 5.2.(a) et (b) ; (b) Positions des sommets de l'arbre de partition de la figure 5.2.(b). Ces positions sont utilisées pour regrouper les arêtes en faisceaux.

classique de dessin d'arbre). Puis les arêtes du graphe sont ajoutées dans ce dessin. Par exemple, si une arête e relie un sommet u à un sommet v, alors e est dessinée de telle sorte qu'elle « suive » le chemin le plus court entre u et v dans l'arbre de partition. L'idée générale est de « fusionner » les arêtes reliant des groupes proches de la hiérarchie.

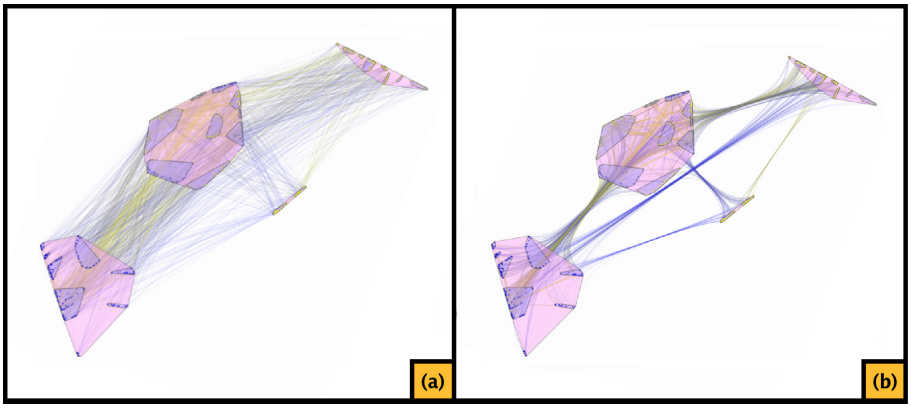

FIG. 5.13: Sous partie d'un réseau d'interactions gène-protéine dessiné par notre approche, avant le post-traitement (a) et après (b).

Nous avons adapté à notre algorithme cette méthode de regroupement d'arêtes. Les sommets de l'arbre de partition sont placés aux positions des métanoeuds auxquels ils correspondent dans les graphes quotients. La figure 5.12.(a) reprend l'exemple de la figure 5.2, la figure 5.12.(b) montre les positions de chaque noeud de l'arbre de partition.

Les figures 5.13.(a) et (b) montrent une sous-partie d'un réseau d'interactions gène-protéine de la mouche avant que ce post-traitement n'ait été effectué (a) et après (b). Cette technique permet d'augmenter la lisibilité du dessin et de mettre en évidence les relations d'adjacences de groupes.

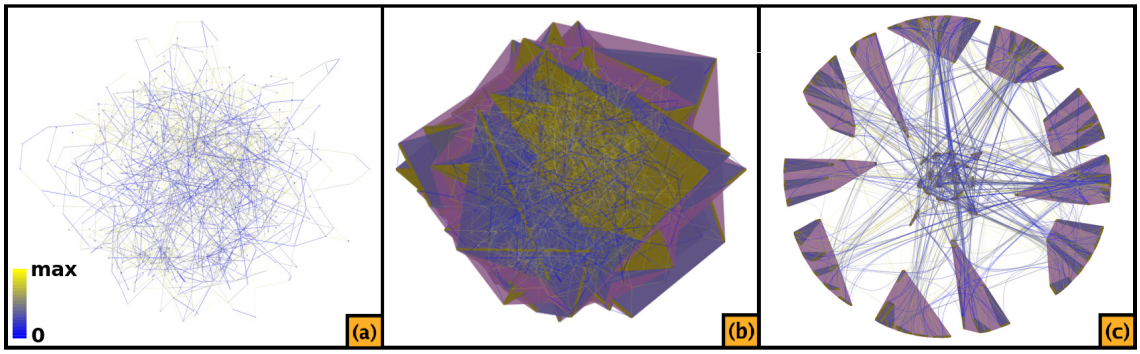

FIG. 5.14: (a) Réseau d'interactions gène-protéine de la mouche dessiné par un algorithme par modèle de forces classique [54], l'effet « brouillon » de ce dessin provient de la nature sans-échelle de ce graphe ; (b) Un algorithme de partitionnement a été appliqué sur ce graphe, et chaque groupe est entouré par une enveloppe convexe ; (c) Résultat de notre technique sur ce graphe partitionné.

5.5.4 Résultats

Notre approche a été implémentée sur la plateforme de visualisation de graphes Tulip [12]. Nous l'avons expérimentée sur un réseau d'interactions gène-protéine de la mouche. Nous avons d'autre part les niveaux d'expression de ces gènes et protéines lors d'expérimentations, ce qui nous a permis de construire un graphe arête-valué. Comme la plupart des réseaux d'interactions biologiques, ce graphe est un graphe sans-échelle[8]. La figure 5.14.(a) montre ce réseau dessiné par un algorithme classique par modèle de forces [54] permettant de prendre aisément en compte les valuations des arêtes.

Afin d'appliquer notre algorithme sur ce réseau, nous avons préalablement effectué un partitionnement multi-niveaux du graphe [9]. Ce partitionnement consiste à construire des boules de sommets similaires (dont les arêtes les reliant ont des valeurs faibles). Cet algorithme a cinq étapes :

1. Calculer une borne en fonction de la valeur moyenne des arêtes et de la densité du graphe
2. Tirer aléatoirement un sommet dans le graphe et mettre ce sommet dans le groupe courant
3. Pour chaque sommet u du groupe courant
   - Parcourir les arêtes (u, v) adjacentes à u
   - Si la valuation de l'arête est plus petite que la borne, ajouter v au groupe courant
4. Répéter 3 tant que l'on peut ajouter des sommets
5. Supprimer du graphe les sommets du groupe courant, s'il reste des sommets retourner à 2

Pour obtenir un partitionnement multi-échelles, cet algorithme de partitionnement est appliqué récursivement sur les groupes, sous-groupes, etc. Nous avons ainsi obtenu un graphe partitionné arête-valué.

[8] i.e. un graphe dont la distribution des degré suit une loi de puissance.

[9] La qualité de ce partitionnement n'est pas discuté dans ce chapitre.

Lorsque l'on entoure chaque groupe (et sous-groupe) de ce graphe partitionné dans le dessin de la figure 5.14.(a), on obtient un dessin complètement illisible (voir la figure 5.14.(b)). Le résultat de notre approche sur ces données est montré dans la figure 5.14.(c). Notre approche augmente clairement la lisibilité du dessin puisque l'on arrive à discerner chaque groupe (et sous-groupe) et que l'on voit aussi les relations fortes entre ces groupes.

5.6 Conclusion

Dans ce chapitre, nous avons présenté un algorithme de dessin de graphe partitionné arête-valué. Notre approche prend en compte un certain nombre de contraintes simultanément. En effet, elle permet de visualiser l'inclusion des groupes, grâce à la contrainte de non-chevauchement de groupes, et de respecter au mieux les valuations des arêtes. Ce processus offre des résultats esthétiquement plaisants qui facilitent l'identification des groupes et sous-groupes et met en évidence les relations d'adjacences. De plus, prendre en compte les valuations des arêtes lors du dessin permet de reconnaître visuellement les valuations des arêtes.

Nous avons présenté une expérimentation de notre approche sur des données réelles provenant d'expériences biologiques. Ce type de données possède habituellement la propriété de graphe sans-échelle, ce qui rend extrêmement difficile leur visualisation. Nous avons montré qu'utiliser notre méthode peut améliorer de manière significative la lisibilité des graphes d'interactions.

Dans ce chapitre, nous avons mis en place un algorithme par modèle de forces permettant de représenter une partition multi-échelles de graphe arête-valué. Certaines parties de ce travail soulèvent de nouvelles questions, notamment sur la définition d'un bon échantillonnage des sommets d'un graphe arête-valué. D'autre part, il semble intéressant d'étudier l'influence de la métrique utilisée lors du calcul de l'arbre couvrant sur la qualité de l'approximation des distances. Cela permettrait de faire un compromis entre temps de calcul et qualité de l'approximation des distances dans le graphe.

Chapitre 6

Visualisation de décompositions de graphes : application aux réseaux métaboliques

Nous abordons dans ce chapitre le problème de la visualisation d'une décomposition de graphe, c'est-à-dire, de la visualisation d'un graphe fragmenté en groupes chevauchants (*cf* définition en section 2.2). Plus particulièrement, nous nous intéressons à un problème bien connu en biologie et en bioinformatique : la visualisation de réseaux métaboliques. Les travaux présentés dans ce chapitre sont les résultats de recherches publiées dans [19, 22]. Dans ce chapitre, nous allons tout d'abord expliquer la problématique liée à une telle visualisation dans la partie 6.1, les contraintes spécifiques aux voies et réseaux métaboliques dans 6.2, puis nous présenterons les outils existants dans ce domaine dans la section 6.3. Dans la section 6.4, nous détaillerons le modèle ainsi que les données utilisés. Enfin, nous expliquerons notre approche dans la section 6.5 et comparerons les résultats obtenus dans la section 6.6.

6.1 Problématique

Une visualisation efficace d'un graphe décomposé doit permettre d'identifier aisément chaque groupe de la décomposition. Mettre en place un tel système de visualisation est un problème difficile. En effet dans le cas général, on ne peut pas garantir que deux groupes ne se s'intersectent (ou ne chevauchent) visuellement bien que cette intersection ne soit pas présente dans la décomposition. Dans la suite, nous appellerons croisements indésirables de tels chevauchements ou intersections.

Les figures 6.1.(a) et (b) montrent une paire (G, H) représentant une décomposition de graphe. On peut observer sur la figure 6.1.(a) qu'il existe un croisement indésirable entre les groupes jaune et bleu. La figure 6.1 montre le graphe de dépendance du graphe décomposé (G, H). Ce graphe est un graphe complet à 5 sommets (K_5) et ne peut donc pas être dessiné dans le plan sans croisement d'arêtes (d'après le théorème de Kuratowski). Etant donné que le nombre minimal de croisements d'arêtes dans un dessin (dans le plan) du graphe K_5 est égal à un, la représentation « optimale » du graphe décomposé de la figure 6.1.(a) et (b) contiendra un croisement indésirable.

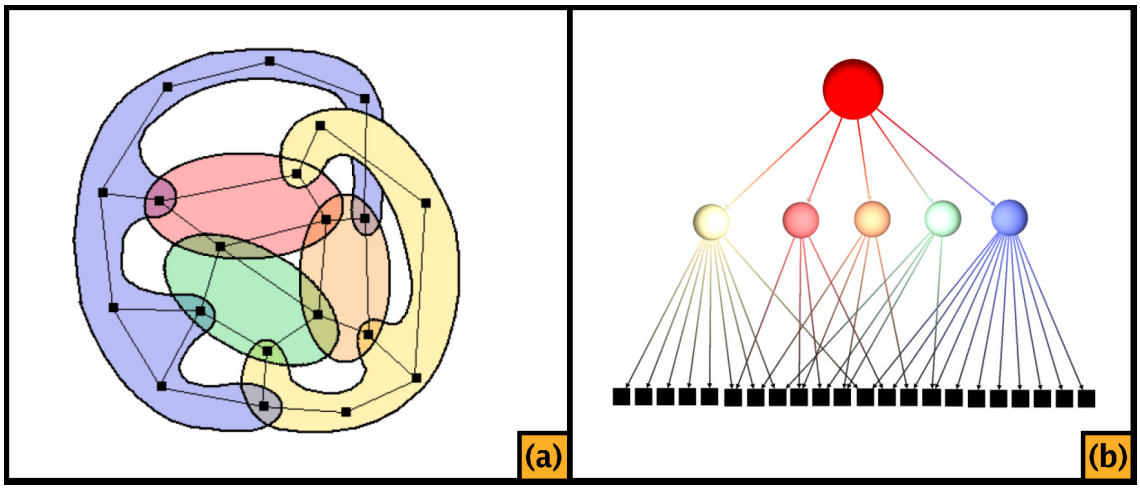

FIG. 6.1: Exemple de décomposition de graphe : (a) et (b) la paire (G,H) comme défini dans la section 2.2. Dans (a) chaque groupe est entouré par une couleur différente.

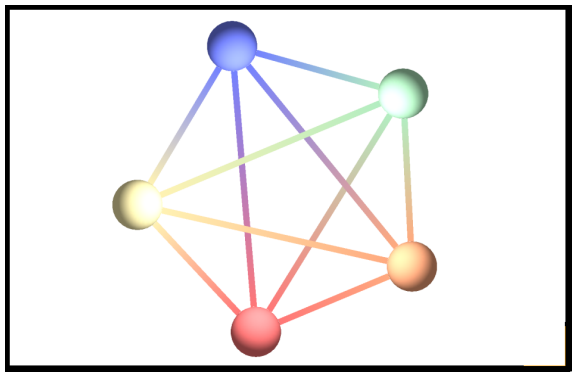

FIG. 6.2: Graphe de dépendance correspondant au graphe décomposé (G, H) de la figure 6.1.

Afin de représenter ce type de données, il existe deux principales approches. La première consiste à dupliquer un certain nombre de sommets (et arêtes) partagés par plusieurs groupes de manière à ce que le graphe de dépendance résultant soit planaire. La seconde approche consiste à se focaliser sur un certain nombre de groupes et de garantir qu'ils seront correctement représentés.

Dans ce chapitre, nous prendrons l'exemple des réseaux métaboliques. Ce type de réseau est l'un des exemples de décomposition de graphes classiques en biologie. Un réseau métabolique peut être décomposé en voies (et/ou super-voies) métaboliques. Ces voies métaboliques partagent des composés puisque les composés d'entrée de certaines voies sont les composés de sortie d'autres. De ce fait, un réseau métabolique peut être représenté comme un graphe décomposé avec chevauchements.

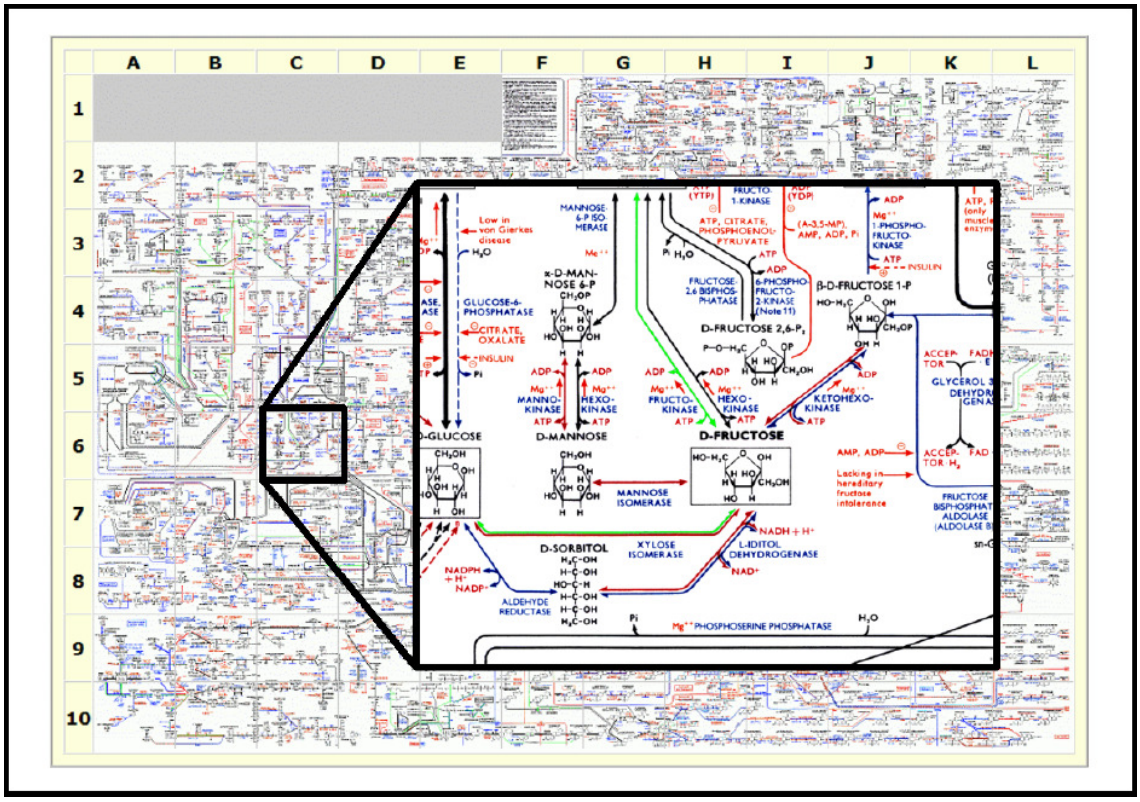

FIG. 6.3: Poster de Boehringer [102] représentant le réseau métabolique de l'*homo sapiens*.

6.2 Contraintes et objectifs spécifiques aux réseaux métaboliques

Appliqué aux réseaux métaboliques, le problème de la visualisation d'une décomposition de graphe revient à visualiser le réseau entier tout en conservant l'information liée aux voies métaboliques. La contrainte principale étant de visualiser le plus de groupes (ici de voies) mais aussi la plus grande partie du réseau correctement. Nous proposons une autre contrainte : les composés et réactions contenus dans une voie métabolique doivent être positionnés aussi proches que possible. La figure 6.3 montre le réseau métabolique entier de l'*homo sapiens* issue de [102] et la figure 6.4 montre les représentations de deux voies métaboliques (la glycolyse et la néoglucogénèse) données sur le site de KEGG [80]. Ces deux représentations ont été dessinées manuellement. Une analyse précise de ces représentations permet d'identifier les conventions tirées de manuels de dessin (pour une description détaillée, voir [17, 78]).

Dans [102], G. Michal a défini un certain nombre de ces contraintes pour la représentation de réseaux biochimiques, notamment la limitation du nombre de points de contrôle par arête ou encore la représentation circulaire de processus cycliques. La figure 6.5 montre le résultat de l'application de ces contraintes sur une sous-partie d'un réseau biologique.

Il est important de noter que les choix faits pour établir ces conventions correspondent généralement à une réalité biologique. Les processus de dégradation cyclique, par exemple

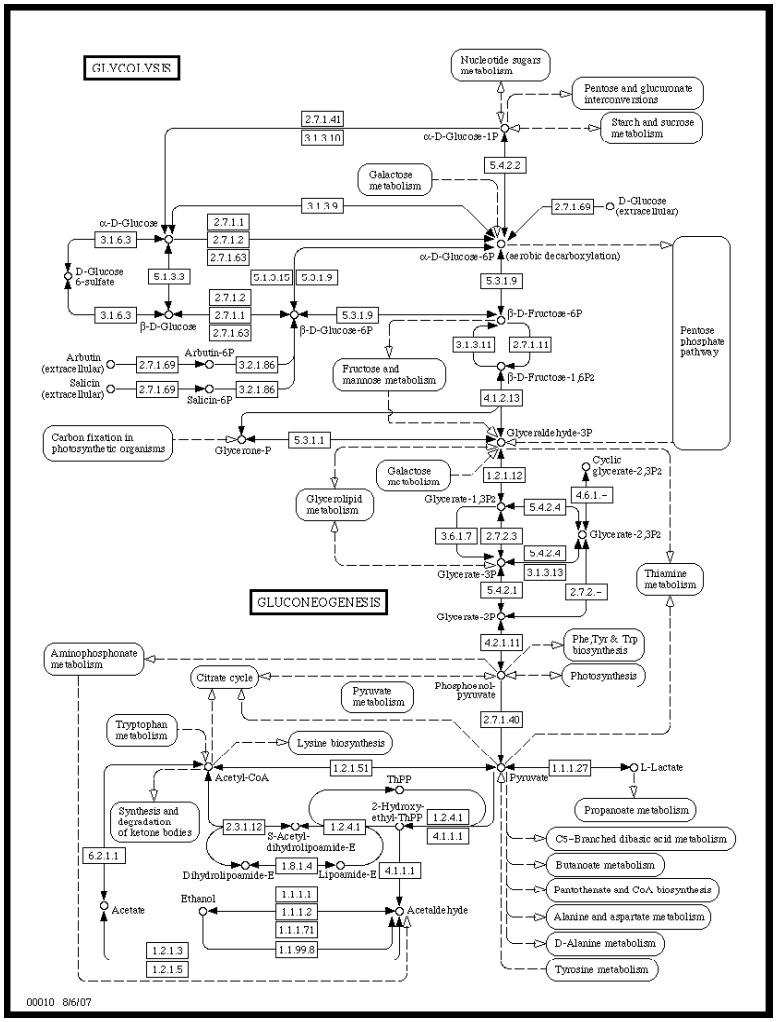

FIG. 6.4: Voies métaboliques de la Glycolyse et de la Gluconeogenèse comme elles sont données sur le site de KEGG [80].

le cycle de Krebs, sont mis en évidence par une représentation circulaire. Les cascades de réactions sont aussi considérées comme des représentations canoniques d'objets biologiques. Ces conventions de dessin correspondent aux critères esthétiques fréquemment utilisés par la communauté du dessin de graphes [16] comme le nombre de croisements d'arêtes, le nombre de points de contrôle (ou brisures) par arêtes.

Dans ce chapitre, les contraintes que nous nous sommes fixées sont :

1. Limiter le nombre de croisements d'arêtes
2. Maximiser l'angle minimum entre deux arêtes adjacentes (notion appelée *résolution angulaire*)
3. Limiter le nombre de points de contrôle par arête
4. Représenter chaque voie métabolique dans une « petite » région du dessin
5. Respecter les représentations circulaires des cycles et alignées des cascades de réactions

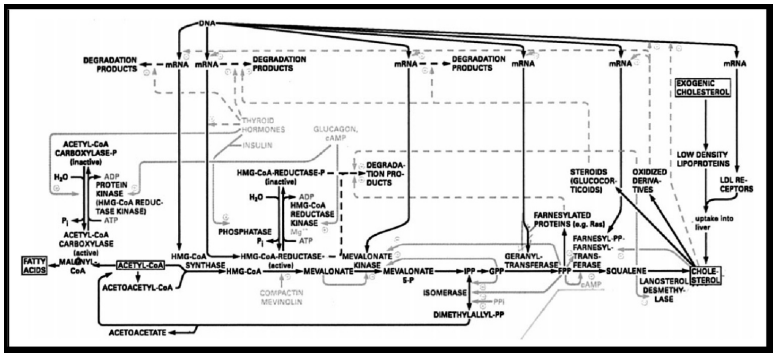

FIG. 6.5: Représentation manuelle respectant les contraintes de représentation proposées dans [102] d'une sous-partie d'un réseau biologique correspondant au mécanisme de régulation de la biosynthèse du cholestérol.

6.3 Outils existants de visualisation

Les études portant sur le métabolisme sont faites à différentes échelles, de la simple dégradation d'une molécule à l'étude du système complet. Par exemple, les toxicologues s'intéressent souvent à la dégradation d'une molécule et donc à un très petit nombre de réactions permettant cette dégradation. A un niveau plus large, certains biologistes ne travaillent que sur un certain nombre de voies métaboliques (e.g. la voie métabolique de la dégradation du glucose, d'un type de sucre particulier ou encore sur le cycle de Krebs). Enfin, le dernier niveau d'étude est celui du réseau métabolique entier. En effet, il est aussi intéressant de visualiser les interconnections des voies métaboliques, par exemple pour étudier les modifications de fonctionnement du réseau en réponse à un stimulus extérieur (e.g. infection par un parasite, un virus, injection d'une toxine). Il est important de noter que ces voies et réseaux métaboliques étaient représentés manuellement (e.g. figures 6.3, 6.4 et 6.5), ce qui explique que les travaux sur ce sujet sont très récents et que la plupart de ces travaux portent sur les deux plus petites échelles. Dans cette section, nous présentons succinctement [1] les travaux sur le dessin automatique de voies métaboliques, puis à une échelle plus large, de réseaux métaboliques.

Etant donné l'organisation hiérarchique des voies métaboliques (à l'exclusion des cycles de réactions), les travaux portant sur la visualisation des voies métaboliques utilisent généralement des algorithmes hiérarchiques pour dessiner ces voies (e.g. [121, 115, 25]). Cependant, ces techniques ne permettent pas de mettre en évidence les cycles de réactions. Pour répondre à cette attente des biologistes, Becker et Rojas [17] effectuent préalablement une recherche des plus longs cycles indépendants (i.e. ne partageant pas de sommet). Dans [131, 57], les auteurs ont affiné cette technique puisque leurs algorithmes permettent de représenter des cycles partageant des composés.

Selon Becker et Rojas [17], ces techniques permettent de représenter simultanément plusieurs voies métaboliques (jusqu'à cinq). Cependant elles ne peuvent être appliquées efficacement aux réseaux métaboliques entiers.

[1] Pour plus de détails, le lecteur peut se référer à la section 3.4

Une première approche pour représenter les réseaux métaboliques consiste à utiliser des algorithmes existants de dessin de graphe. En particulier, les algorithmes par modèle de forces sont souvent utilisés notamment par *Cytoscape* [118] ou encore *SBMLviewer* [1]. Cependant, comme mentionné dans [112], ces algorithmes ne produisent pas de dessin satisfaisant du point de vue des biologistes puisqu'ils ne respectent pas leurs conventions de dessin. Il existe deux outils principaux dédiés à la visualisation de réseau métabolique : *Reactome* [77] et *Pathway Tools cellular overview diagram* [108]. Afin de respecter les conventions de dessin et de conserver l'information liée au découpage en voies métaboliques, ces deux outils dessinent séparément chaque voie métabolique en prenant en compte ces conventions. La deuxième étape consiste ensuite à dessiner l'ensemble des voies dans une vue unique. Le problème de ces deux approches est qu'elles autorisent la duplication de sommets (un composé appartenant à k voies métaboliques est dupliqué k fois) et par conséquent offrent un ensemble de vues locales plutôt qu'une vue globale du réseau.

6.4 Modèle et données

6.4.1 Modèle

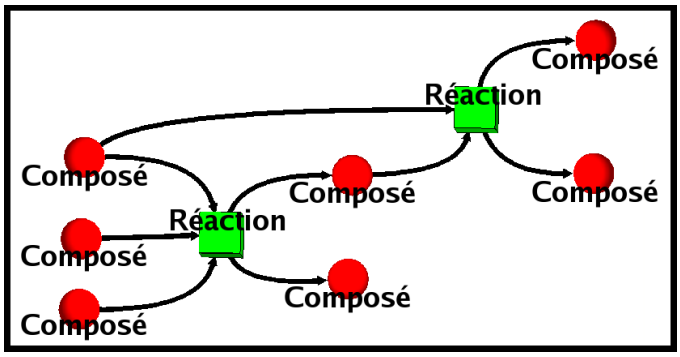

FIG. 6.6: Graphe biparti représentant deux réactions biochimiques.

Nous utilisons dans ce chapitre un modèle de graphe biparti (voir figure 6.6) pour représenter les réseaux métaboliques. Les deux ensembles de noeuds de ce graphe sont l'ensemble des réactions et l'ensemble des composés. Afin de respecter au mieux la réalité biologique, certaines réactions seront réversibles tandis que d'autre ne le seront pas. Les arêtes adjacentes à des réactions non-réversibles seront orientées (arcs) tandis que celles adjacentes à des réactions réversibles seront non-orientées (arêtes). Le modèle que l'on utilise est donc un modèle de graphe biparti mixte (i.e. un graphe biparti contenant des arc et des arêtes, cf définition d'un graphe mixte dans la section 2.2).

```
<reaction id="DIHYDROFOLATEREDUCT__45__RXN"
          name="DIHYDROFOLATEREDUCT-RXN" reversible="true">
        <notes>
                <html:p>GENE_ASSOCIATION: ( G6862 or EG10326 )</html:p>
                <html:p>PROTEIN_ASSOCIATION: ( G6862-MONOMER )
                                             or ( DIHYDROFOLATEREDUCT-MONOMER )</html:p>
                <html:p>SUBSYSTEM: tetrahydrofolate biosynthesis</html:p>
                <html:p>SUBSYSTEM: superpathway of chorismate</html:p>
                <html:p>SUBSYSTEM: formylTHF biosynthesis I</html:p>
                <html:p>PROTEIN_CLASS: 1.5.1.3</html:p>
        </notes>
        <listOfReactants>
                <speciesReference species="THF" stoichiometry="1"/>
        </listOfReactants>
        <listOfProducts>
                <speciesReference species="DIHYDROFOLATE" stoichiometry="1"/>
        </listOfProducts>
</reaction>
```

FIG. 6.7: Description d'une réaction dans un fichier SBML extrait de la base de donnée BioCyc [87] puis traitée afin d'intégrer à chaque réaction l'information d'appartenance à une ou plusieurs voies.

6.4.2 Données

Les données que nous utilisons dans ce chapitre proviennent de la version 10.0 de la base de données BioCyc [85]. Cette base de données regroupe 160 réseaux métaboliques et génomes. Notre collaboration avec l'équipe *BAOBAB* du LBBE (Laboratoire de Biométrie et Biologie Evolutive), Université Lyon 1, nous a permis d'obtenir des réseaux issus de cette base de données. L'un de nos collaborateurs, Ludovic Cottret, a extrait de cette base de données des réseaux métaboliques en utilisant les outils [83] et [90]. La figure 6.7 montre un exemple de description d'une réaction (ici la réaction « DIHYDROFOLATEREDUCT-RXN ») dans l'un de ces fichiers SBML. Cette description permet de connaître pour chaque réaction toutes les voies métaboliques auxquelles elle participe (« SUBSYSTEM »), la liste de ses réactants (« listOfReactants ») ainsi que la liste de ses produits (« listOfProduct »). Elle permet aussi de savoir si la réaction est réversible ou non (« reversible »). D'autre part, ces fichiers contiennent d'autres informations non utilisées dans notre approche mais qui s'avèrent nécessaires aux biologistes comme le(s) gène(s) associé(s) à cette réaction (« GENE_ASSOCIATION ») ainsi qu'un numéro permettant d'identifier sa fonction (« PROTEIN_CLASS »).

6.5 Méthode

L'algorithme que nous utilisons a deux phases principales. Premièrement, un partitionnement multi-échelles (cf section 2.2) est effectué permettant d'obtenir une abstraction du réseau sous forme de graphe quotient[2]. Deuxièmement, les groupes et le graphe quotient sont dessinés en utilisant des algorithmes de dessin appropriés. Dans cette section, nous allons tout d'abord expliquer notre algorithme de partitionnement, puis nous présenterons les algorithmes de dessin que nous avons utilisés, enfin nous comparerons les résultats ainsi obtenus avec ceux des outils existants.

[2]Le graphe quotient est construit en considérant les sommets isolés comme des singletons.

6.5.1 Partitionnement multi-échelles

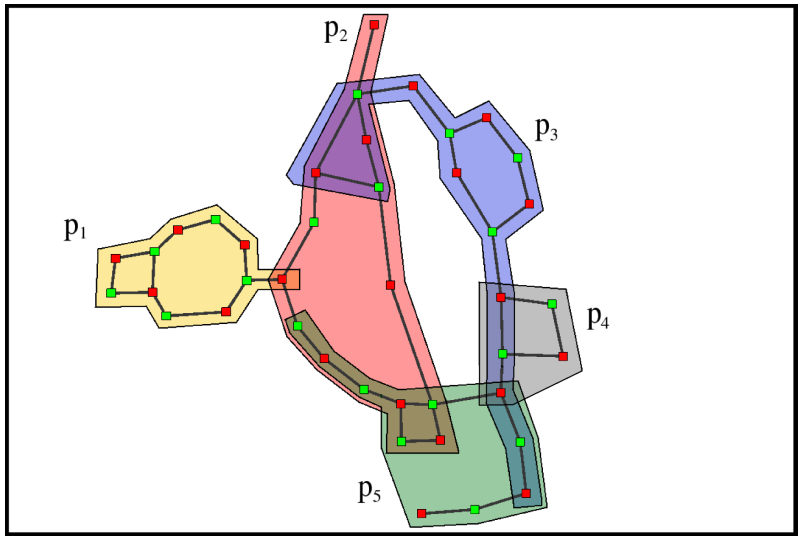

FIG. 6.8: Exemple fictif de réseau métabolique contenant 5 voies métaboliques de couleurs distinctes (p_1, p_2, p_3, p_4 et p_5).

L'une des difficultés principales lorsque l'on veut représenter un réseau métabolique tient au fait que les voies métaboliques partagent souvent des composés et/ou des réactions (e.g. dans le réseau métabolique de Escherichia Coli, une réaction appartient en moyenne à 2.4 voies métaboliques). La figure 6.8 illustre ce phénomène, les régions jaune, rouge et bleue représentent respectivement les voies p_1, p_2 et p_3. On peut voir que p_1 et p_2 partagent un noeud et p_2 et p_3 quatre noeuds. Etant donné que nous avons choisi de ne pas dupliquer de noeud et que les sommets d'une voie doivent être positionnés proches les uns des autres, notre algorithme devra décider si un noeud sera placé près d'une voie métabolique ou près d'une autre. Dans notre exemple, les noeuds partagés par p_2 et p_3 pourraient être placés « proches » de p_2 ou alors de p_3. Pour faire ce choix, nous calculons un ensemble indépendant de voies métaboliques (i.e. un ensemble de voies ne partageant pas de composés et de réactions). Les voies métaboliques de cet ensemble pourront être représentées en respectant la contrainte 4 (la contrainte de proximité définie dans la section 6.2). Ensuite, une seconde étape permettra de détecter les structures topologiques telles que les cycles et les chemins (cf contrainte 5).

6.5.1.1 Première étape : calcul de l'ensemble indépendant

Soit P_G l'ensemble des voies métaboliques du réseau métabolique. L'algorithme cherche un ensemble indépendant $P_{ind} = \{p_{i_1}, p_{i_2} ..., p_{i_r}\}$, r≥1, $P_{ind} \subseteq P_G$ afin de respecter la contrainte de proximité. L'ensemble P_{ind} doit donc respecter les propriétés suivantes :

1. P_{ind} est un ensemble maximal de voies métaboliques indépendantes,
2. $\sum_{k=1}^{k=r} |p_{i_k}|$ est maximisée.

Dans la figure 6.8, les ensembles indépendants sont $\{p_1\}$, $\{p_2\}$, $\{p_3\}$, $\{p_4\}$, $\{p_5\}$, $\{p_1, p_3\}$, $\{p_1, p_4\}$, $\{p_1, p_5\}$ et $\{p_2, p_4\}$. L'ensemble possédant le plus de composés et de réactions est $\{p_1, p_3\}$ puisque ces deux voies contiennent 27 composés et réactions.

Détection d'ensembles indépendants de voies métaboliques :
La recherche d'un ensemble indépendant de voies métaboliques se traduit directement en un problème de coloration propre dans le graphe de dépendance. En effet, une coloration propre consiste à associer une couleur à chaque sommet telle que deux sommets adjacents aient des couleurs différentes. Donc chaque classe de couleur[3] dans le graphe de dépendance représente un ensemble de voies métaboliques indépendantes. Le problème qui consiste à minimiser le nombre de couleurs dans une coloration propre est NP-difficile.

Nous utilisons une heuristique de coloration, celle de Welsh et Powel [132], pour colorer le graphe de dépendance. Cette méthode heuristique fonctionne comme suit :

1. Les sommets sont classés par degré décroissant
2. Les couleurs sont ordonnées
3. La couleur courante est attribuée au premier sommet non colorié, puis on parcourt la suite de la liste et on attribue la couleur courante à tous les sommets pour lesquels cela est possible
4. S'il reste des sommets non coloriés, on change la couleur courante et on revient à l'étape (3)

Choix de P_{ind} :
Pour chaque classe de couleur c, la somme $\sum_{p_i \in c} |p_i|$ est calculée. La classe de couleur qui la maximise est choisie comme ensemble indépendant P_{ind}. La stratégie suivie ici est une stratégie permettant de représenter « correctement » (i.e. en suivant la contrainte de proximité) la plus grande partie du réseau. D'autres stratégies sont envisageables, comme par exemple, représenter « correctement » le plus grand nombre de voies métaboliques ou alors les voies métaboliques les plus centrales.

On note P_{Nind} l'ensemble des voies métaboliques n'appartenant pas à P_{ind}, formellement, $P_{Nind} = P_G / P_{ind}$. Etant donné que nous souhaitons respecter au mieux la contrainte de proximité, nous cherchons ensuite des sous-parties des éléments de P_{Nind} que nous pouvons représenter correctement. En d'autres termes, nous cherchons dans chaque voie métabolique une sous-partie qui n'appartient qu'à cette voie. Plus formellement, l'algorithme calcule l'ensemble P'_{Nind} vérifiant les deux conditions suivantes :

(i) $\forall p \in P'_{Nind}$, $\exists! q \in P_{Nind} \mid p \subseteq q$, et

(ii) $\forall p \in P'_{Nind}$ et $q \in P'_{Nind}, p \neq q$, $p \cap q = \emptyset$.

Chaque élément p de P'_{Nind} est une sous-partie d'une voie métabolique. Représenter tous les sommets de p dans une « petite » région du dessin permet de respecter au mieux la contrainte de proximité.

[3] cf définition dans le chapitre 2

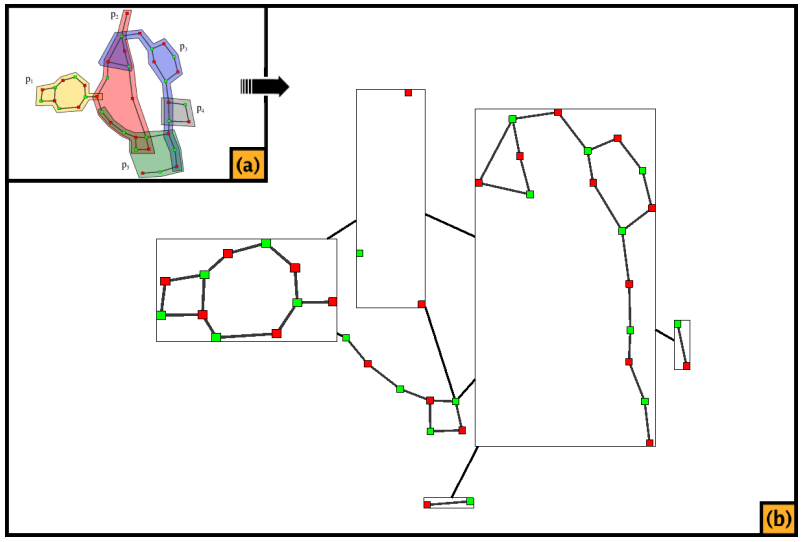

FIG. 6.9: Résultat de la première étape de partitionnement. (a) reprend l'exemple de la figure 6.8 ; (b) résultat de la première étape sur le réseau de la figure (a).

A la fin de cette étape de détection de sous-parties indépendantes du réseau, nous regroupons chaque élément de P_{ind} et de P'_{Nind} en métanoeud (groupe), i.e., nous remplaçons chaque sous-graphe induit par un élément de P_{ind} et de P'_{Nind} par un noeud le représentant. La figure 6.9.(a) reprend l'exemple de la figure 6.8 et la la figure 6.9.(b) montre le résultat de cette première étape de partitionnement.

6.5.1.2 Seconde étape : détection de cycles et de chemins

Afin de représenter les cycles et les cascades de réactions en suivant les conventions de dessin (i.e. contrainte 5 donnée dans la section 6.2), nous procédons ensuite à une détection de cycles mixtes indépendants (i.e. ne partageant pas de sommets) et de chemins mixtes.

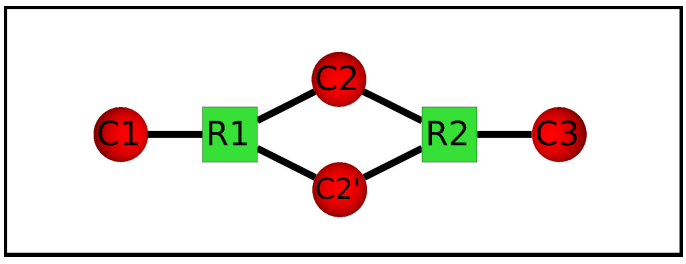

FIG. 6.10: . Supposons que la réaction $R1$ transforme le composé $C1$ en $C2$ et $C2'$ et que $R2$ transforme les composés $C2$ et $C2'$ en $C3$. Si les réactions $R1$ et $R2$ sont réversibles, alors une recherche « simple » de cycle le plus long trouverait le cycle $C2, R1, C2', R2$ bien que ce cycle ne soit pas biologiquement pertinent.

Etant donné que ces cycles doivent être pertinents d'un point de vue biologique(voir la figure 6.10), nous avons ajouté une contrainte dans cette recherche. Soit $c_1, r_1, c_2, r_2, ..., c_l, r_l, c_1$ un cycle où tout r_i est une réaction et tout c_j un composé. Si c_i est l'un des

réactants (respectivement produits) de r_i alors c_{i+1} est l'un de ses produits (respectivement réactants). Le problème classique de la recherche du plus grand cycle est connu pour être NP-complet[4]. Nous utilisons cependant une méthode exacte pour trouver ces plus longs cycles en prenant en compte cette dernière contrainte. Au cours de la recherche, si le temps de calcul dépasse une borne donnée, l'algorithme s'arrête et considère le plus long cycle qu'il a déjà trouvé comme le plus long cycle. Bien que cela soit étonnant, en pratique cette borne n'a jamais été atteinte (voir la section 6.6.1 pour le temps de calcul de cette recherche). En effet, les recherches de cycles ne sont effectuées que sur des sous-parties du réseau de petites tailles (voies métaboliques et sous-parties non regroupées lors de la première étape de l'algorithme de partitionnement).

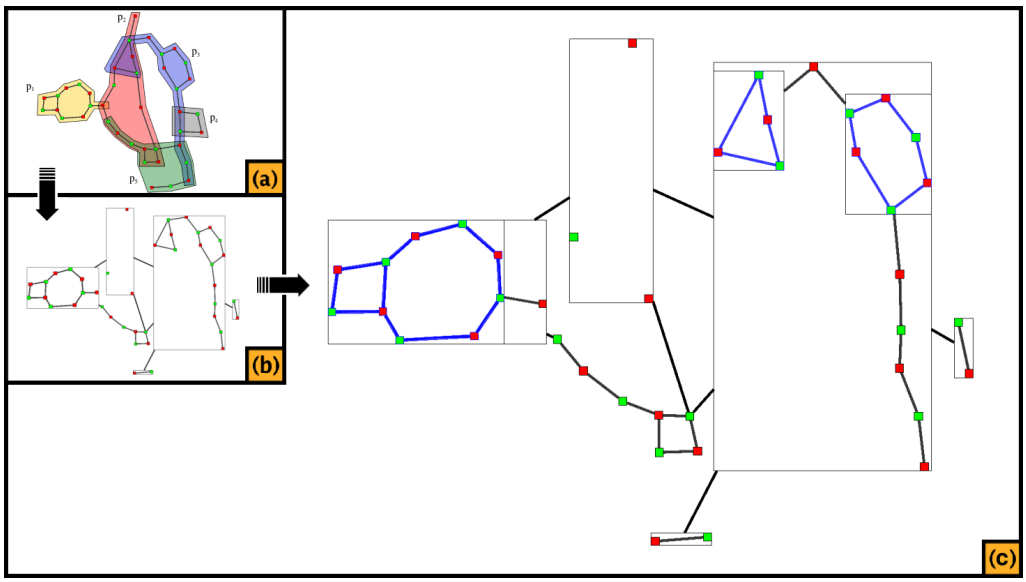

FIG. 6.11: Résultat de la recherche de cycles dans P_{ind} et P'_{Nind}. (a) et (b) reprennent l'exemple de la figure 6.9 ; (c) les arêtes des cycles trouvés lors de cette étape apparaissent en bleu.

Cette recherche est tout d'abord effectuée dans les groupes de P_{ind} et de P'_{Nind} et chaque cycle est regroupé en un métanoeud. La figure 6.11 reprend l'exemple de la figure 6.9. Dans la figure 6.11.(c), les plus longs cycles indépendants de chaque élément de P_{ind} et P'_{Nind} apparaissent en bleu.

Ensuite, l'algorithme cherche les plus longs cycles mixtes indépendants dans le graphe quotient obtenu lors de la première passe de l'algorithme de partitionnement, en excluant les métanoeuds. A chaque itération, nous regroupons le plus long cycle en un métanoeud et l'excluons des prochaines recherches.

Afin de représenter correctement les sommets (réactions et composés) d'une cascade de réactions, nous cherchons ensuite les plus longs chemins mixtes de sommets de degré inférieur ou égal à deux. La même contrainte réactants/produits est appliquée lors de cette recherche. Une fois encore chaque cascade de réactions trouvée est regroupée en un

[4]En effet, cela revient à chercher si le graphe est Hamiltonien.

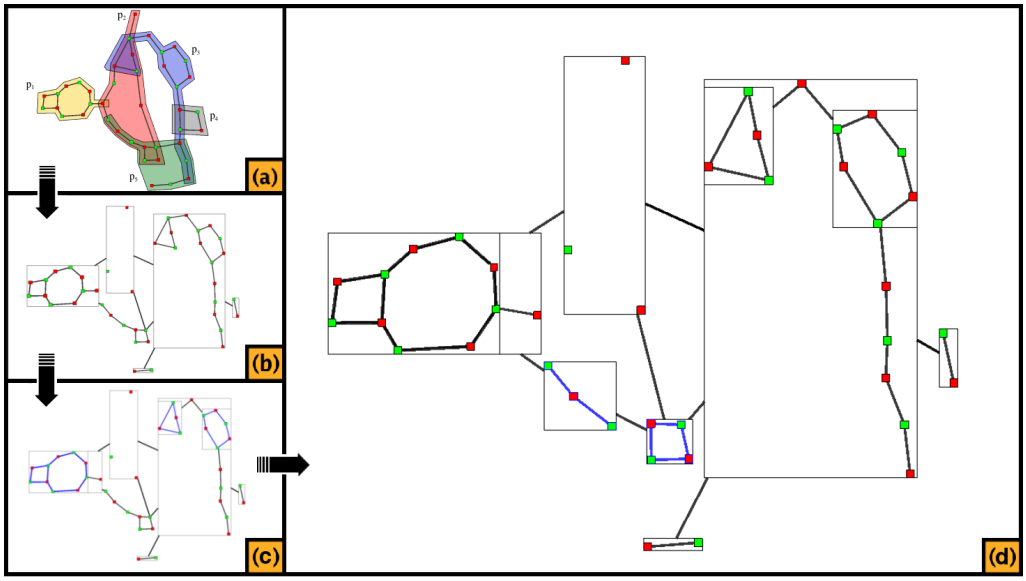

FIG. 6.12: Résultats de la recherche de cycles et de cascades de réactions dans le graphe de la figure 6.11.(c). (a), (b) et (c) reprennent l'exemple de la figure 6.11 ; (d) les arêtes des cycles et cascades de réactions trouvés lors de cette étape apparaissent en bleu.

métanoeud. Cette recherche n'est faite que dans le graphe quotient puisque l'algorithme utilisé pour dessiner les groupes permet la visualisation des cascades de réactions. La figure 6.12.(d), montre le résultat des recherches des plus longs cycles et des cascades de réactions dans le graphe de la figure 6.11.(c). Les cycles et cascades y apparaissent en bleu.

Le résultat de ces deux phases de partitionnement est un graphe quotient multi-échelles noté Q_G. Ce graphe quotient sera le paramètre d'entrée de l'algorithme de dessin.

6.5.2 Paramétrer les centres d'intérêts

Notre algorithme permet de mettre l'accent sur plusieurs voies métaboliques, i.e. l'utilisateur peut choisir un certain nombre de voies qui doivent être représentées en respectant la contrainte de proximité. Pour cela, il faut ajouter (si possible) ces voies métaboliques dans l'ensemble de voies indépendantes P_{ind}.

L'utilisateur peut donc donner en paramètre une liste ordonnée (par intérêt décroissant) de voies métaboliques. Si les sommets correspondants dans le graphe de dépendance ne forment pas un ensemble indépendant, l'ordre des voies dans la liste fixe la priorité associée à chacune de ces voies pour en extraire un ensemble indépendant de voies, noté $P_{utilisateur}$.

Les sommets représentant ces voies dans le graphe de dépendance ainsi que leurs voisins sont alors supprimés du graphe de dépendance. Un ensemble indépendant P'_{ind} est ensuite calculé (en utilisant la méthode décrite dans la partie 6.5.1) dans le graphe de dépendance résultant. Cela permet de garantir que les voies des ensembles indépendants P'_{ind} et $P_{utilisateur}$ ne partagent ni composé ni réaction.

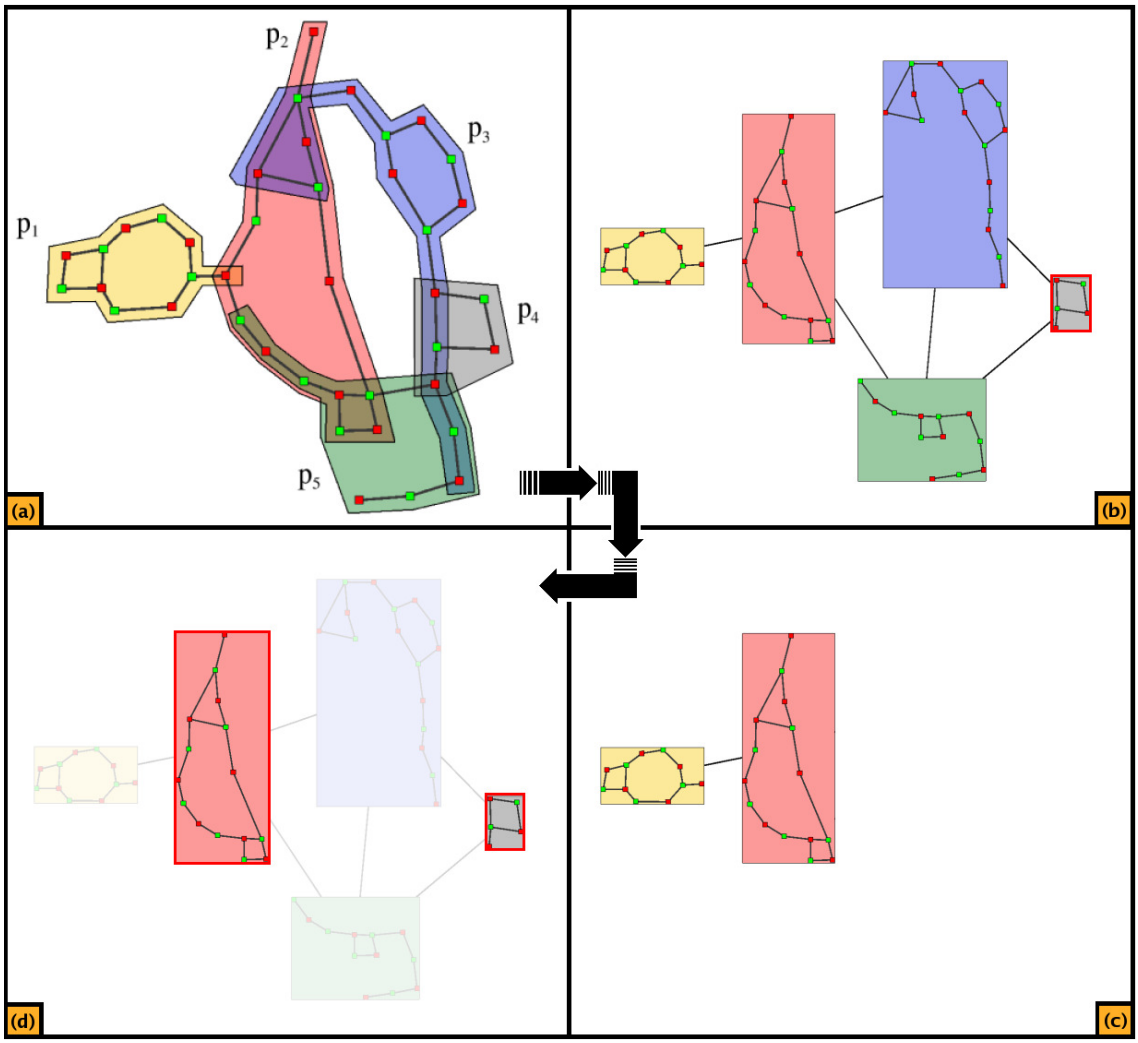

FIG. 6.13: (a) Exemple de la figure 6.8 ; (b) Graphe de dépendance correspondant ; (c) Le sommet centre d'intérêt, ici le sommet correspondant à $p4$ (entouré en rouge dans la figure (b)), ainsi que ces voisins sont retirés du graphe de dépendance ; (d) L'ensemble indépendant de cet exemple est $P_{ind} = \{p4, p2\}$.

Finalement, l'ensemble indépendant de voies métaboliques P_{ind} est l'union de $P_{utilisateur}$ et de P'_{ind}.

Reprenons l'exemple de la figure 6.8, la figure 6.13.(b) montre le graphe de dépendance de ce réseau. Supposons que l'utilisateur veuille voir la voie $p4$ correctement dessinée. L'ensemble $P_{utilisateur} = \{p4\}$ est bien un ensemble indépendant. Etant donné que l'on cherche à obtenir un ensemble indépendant P_{ind} contenant la voie $p4$, on supprime le sommet correspondant à $p4$ dans le graphe de dépendance. D'autre part, puisque $p4$ est dans l'ensemble P_{ind}, aucune voie partageant des sommets avec $p4$ ne doit être ajoutée à P_{ind} : on supprime donc du graphe de dépendance tous les voisins du sommet correspondant à $p4$ (voir la figure 6.13.(c)). En utilisant la technique décrite dans la partie 6.5.1, on calcule un ensemble indépendant sur le graphe résultant et on obtient l'ensemble $P'_{ind} = \{p2\}$.

Enfin, l'ensemble P_{ind} étant l'union de $P_{utilisateur}$ et de P'_{ind}, on obtient $P_{ind} = \{p4, p2\}$ (voir la figure 6.13.(d)). Les voies métaboliques $p2$ et surtout $p4$ seront donc dessinées en respectant la contrainte de proximité.

6.5.3 Algorithmes de dessin

Pour dessiner le réseau métabolique, nous utilisons plusieurs algorithmes : un pour le graphe quotient Q_G et deux pour les métanoeuds.

6.5.3.1 Dessin des métanoeuds

Pour dessiner les sous-graphes associés aux métanoeuds, nous utilisons un algorithme similaire à celui de [17]. Notre algorithme est un algorithme ascendant[5]. Etant donnée la méthode de partitionnement utilisée, un sous-graphe est soit un cycle soit une voie (ou une sous-partie d'une voie) métabolique. Dans le premier cas, nous utilisons un algorithme de dessin circulaire, et dans le second cas, l'algorithme de dessin hiérarchique présenté dans [12] (voir la figure 6.24.(a)).

6.5.3.2 Dessin du graphe quotient

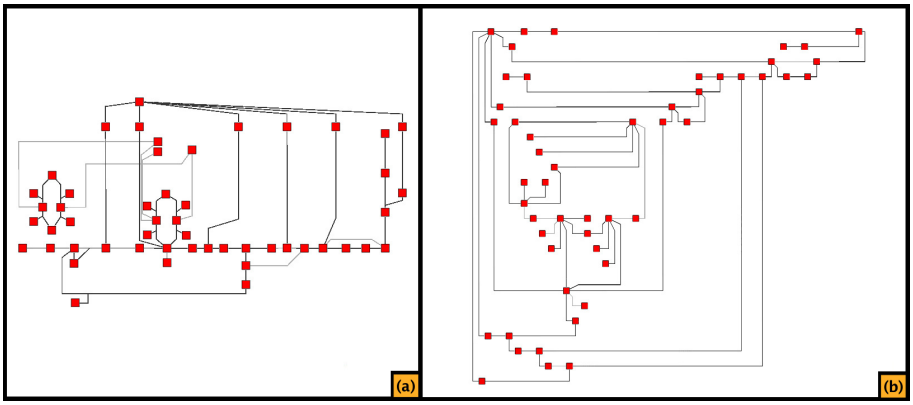

FIG. 6.14: (a) Sous-graphe extrait du réseau de la figure 6.5 ; (b) Dessin obtenu pas l'algorithme Mixed-Model de Gutwenger et Mutzel [64] sur ce sous-graphe.

Nous voulons que notre dessin optimise la résolution angulaire, le nombre de points de contrôle et offre un bonne visibilité. L'algorithme Mixed-Model de Gutwenger et Mutzel [64] est un compromis entre ces critères esthétiques. De plus, les dessins produits par cet algorithme sont visuellement similaires à ceux dessinés à la main (voir la figure 6.14).

[5]Algorithme dessinant tous les sous-graphes du plus bas dans la hiérarchie des métanoeuds au plus haut

Pré-traitements

Pour utiliser l'algorithme Mixed-Model, nous devons modifier le graphe quotient. En effet, cet algorithme ne peut être appliqué que sur les graphes planaires biconnexes. Par conséquent, nous devons tout d'abord rendre le graphe quotient planaire. Ce problème est bien connu pour être NP-difficile [97]. Il existe deux approches principales permettant de trouver une solution, celle par augmentation (un sommet virtuel est ajouté à chaque croisement d'arêtes) et celle par suppression d'arêtes (ou de noeuds), pour plus de détails le lecteur peut se référer à [96]. Le désavantage de la technique basée sur l'augmentation est qu'elle peut ajouter jusqu'à $|V|^4$ noeuds. Etant donnée la taille des réseaux métaboliques et les capacités des ordinateurs actuels, il est possible d'utiliser cette méthode, cependant le dessin produit serait alors extrêmement difficile à comprendre (e.g. il peut alors y avoir $O(|V|^2)$ points de contrôle par arête). C'est pourquoi nous utilisons une heuristique basée sur la suppression d'arêtes : les sommets de plus forts degrés sont supprimés un à un jusqu'à ce que le graphe soit planaire. Tous les noeuds sont ensuite réinsérés et les arêtes supprimées sont rajoutées une à une tant que le graphe reste planaire. Cet algorithme simple à mettre en place a l'avantage de conserver la biconnexité du graphe original (i.e. si le graphe quotient est biconnexe alors le sous-graphe planaire le sera aussi).
Dans le cas où le sous-graphe planaire n'est pas biconnexe, nous utilisons pour le rendre biconnexe une technique basée sur la méthode de décomposition de graphe en composantes biconnexes présentée dans la section 4. Pour cela, il suffit d'insérer une arête entre chaque composante biconnexe trouvée lors du parcours en profondeur.

Algorithme Mixed-Model et ses modifications

Le sous-graphe planaire du graphe quotient obtenu est dessiné par une version modifiée de l'algorithme Mixed-Model [64]. Pour résumer, cet algorithme a trois phases :

- Lors de la première phase, l'algorithme construit une partition ordonnée de l'ensemble des sommets appelée *« Shelling order »*.
- La seconde phase consiste à attribuer à chaque sommet des points de contrôle pour ses arêtes adjacentes.
- La dernière étape est une phase de positionnement.

Shelling order :
D'une manière informelle, construire le *Shelling order* consiste à supprimer successivement des sommets de la face extérieure du graphe tout en conservant la biconnexité du graphe. En traversant dans l'ordre inverse le *Shelling order*, l'algorithme peut alors positionner les sommets et les arêtes sans croisement d'arêtes.

Définition 6.1 (Shelling Order) *Soit* $SO = \{V_1, V_2, ..., V_N\}$ *une partition ordonnée de l'ensemble de sommets. Notons* $G_k = (V(G_k), E(G_k))$ *le graphe induit par les sommets de* $V_1 \cup ... \cup V_k$*, le graphe* G_N *représente donc le graphe entier. SO est un* Shelling order *si et seulement si :*

1. $V_1 = \{v_1, ..., v_s\}$*, où* $v_1, ..., v_s$ *est un chemin qui parcourt la face extérieure du graphe* G_N *dans le sens des aiguilles d'une montre avec* $s \geq 2$ *et* $E(G_1) \cap E(G_N) = \{(v_i, v_{i+1}) | 1 \leq i < s\}$.

2. *Chaque G_k est connexe et « intérieurement » biconnexe, i.e. la suppression d'un sommet interne de G_k ne le déconnecte pas.*
3. *Soit $C_k = \{c_1, ..., c_q\}$ les sommets d'un parcours dans le sens inverse des aiguilles d'une montre de la face extérieure de G_k de v_1 à v_s. Pour chaque $2 \leq k \leq N$, l'une des deux conditions suivantes est vraie :*
   - *$V_k = \{z\}$ et z est un sommet de C_k ayant au moins trois voisins dans G_{k-1}, ou*
   - *$V_k = \{z_1, ..., z_p\} \subset C_k$, $p \geq 1$, et il existe des sommets c_l et c_r, $l > r$ de C_k tels que $c_r, z_1, ..., z_p, c_l$ est un chemin qui parcourt la face extérieure de G_k dans le sens inverse des aiguilles d'une montre et $\forall i \in [1..p], deg_{G_k}(z_i) = 2$.*

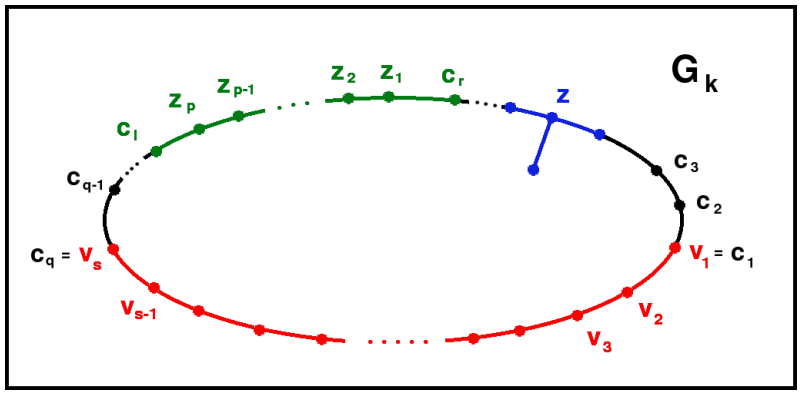

FIG. 6.15: Le graphe G_k d'un *Shelling order*. La partie rouge du dessin représente l'ensemble V_1 du *Shelling order*. Les parties bleue et verte représentent les deux cas possibles pour l'ensemble V_k.

La figure 6.15 illustre cette définition. Les sommets $v_1, v_2, ..., v_s$ en rouge forment l'ensemble V_1 du *Shelling order*. Les parties bleue et verte représentent les deux cas possibles : soit V_k ne contient qu'un sommet z de degré au moins 3, soit V_k contient un chemin de sommets de degré 2 de la face extérieure de G_k.

Calcul des points de contrôle :
La seconde étape consiste à calculer pour chaque sommet les points de contrôle des arêtes qui lui sont adjacentes, appelés *point entrant* et *point sortant* (notés *inpoint* et *outpoint*).

La figure 6.16 montre un exemple de positionnement des inpoints et des outpoints d'un sommet, ce placement est effectué de manière à optimiser la résolution angulaire (i.e. l'angle minimum entre deux arêtes adjacentes).

Positionnement :
La dernière étape de l'algorithme de Gutwenger et Mutzel [64] consiste à positionner les sommets et les arêtes. Pour ce faire, l'algorithme calcule les coordonnées des sommets de G_1, G_2, ..., G_N. Lors de cette phase de placement, l'algorithme maintient à jour les ordonnées absolues des sommets et des abscisses relatives. A l'itération k de l'étape de positionnement, l'algorithme positionne les sommets de l'ensemble V_k du Shelling order.

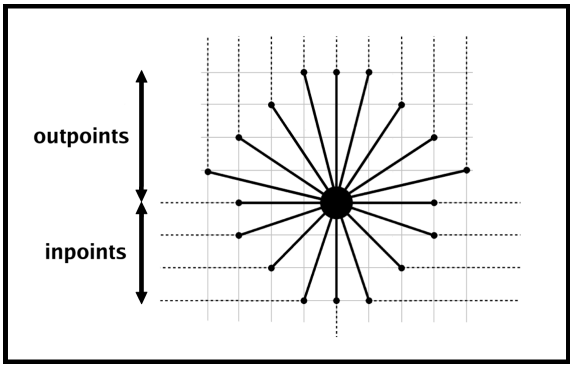

FIG. 6.16: Exemple de positionnement des in- et outpoints d'un sommet u ayant 9 inpoints et 9 outpoints. Les lignes en pointillé indique les directions des arêtes correspondantes.

Supposons que $C = \{c_1, c_2, ..., cq\}$ est le contour (i.e. les sommets de la face extérieure) de G_k, alors les abscisses relatives sont calculées comme suit :

$$x(v) = \begin{cases} x_{abs}(v) & \text{si } v = c_1 \\ x_{abs}(c_i) - x_{abs}(c_{i-1}) & \text{si } v = c_i \text{ avec } i \geq 2 \\ x_{abs}(v) - x_{abs}(father(v)) & \text{si } v \notin C \end{cases}$$

où $x_{abs}(v)$ est la position absolue de v et $father(v)$ est le sommet auquel le position de v est relative. L'idée est que lorsque l'on doit décaler un sommet c_i de C, on doit aussi décaler $c_{i+1}, c_{i+2}, ..., c_q$ mais aussi un certain nombre de sommets qui n'appartiennent pas à C. Ainsi pour chaque sommet c_i de C, l'algorithme maintient à jour l'ensemble $M(c_i)$ des sommets qui doivent être décalés si c_i est décalé. Les ensembles $M(c_i)$, $1 \leq i \leq q$, forment une forêt dont chaque $M(c_i)$ est un arbre. La relation $father$ permet de connaître l'unique prédécesseur d'un sommet dans ces arbres et ainsi d'utiliser des abscisses relatives.

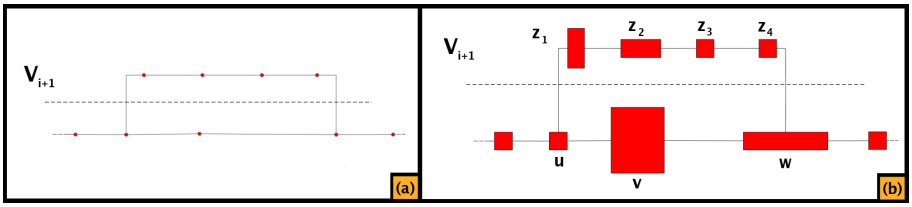

FIG. 6.17: (a) Positionnement des sommets de V_i et de V_{i+1} dans l'algorithme de [64] ; (b) Positionnement des sommets de V_i et de V_{i+1} lorsque ceux-ci ont des tailles variables.

Nous avons modifié cette étape afin que l'algorithme de [64] prenne les tailles des sommets en compte, la figure 6.17 illustre cette modification. Supposons que nous calculons lors de cette étape les positions des sommets de $V_{i+1} = \{z_1, z_2, z_3, z_4\}$. Notons $x(u)$ et $y(u)$ (resp. $x_{area}(u)$ et $y_{area}(u)$) les coordonnées du sommet u si les tailles des sommets ne sont pas prises en compte (resp. si les tailles des sommets sont prises en compte). Les

équations suivantes permettent de prendre en compte les tailles des sommets :

$$x_{area}(z_i) = \begin{cases} x(z_1) + largeur(z_i)/2 & \text{si } i = 1 \\ x(z_i) + largeur(z_{i-1})/2 + largeur(z_i)/2 & \text{si } 2 \leq i \leq 4 \end{cases}$$

et, $\forall i \in [1..4]$:

$$y_{area}(z_i) = y(z_i) + max_{z \in \{u,v,w\}}(hauteur(z))/2 + max_{1 \leq i \leq 4}(hauteur(z_i))/2$$

L'utilisation de la hauteur et de la largeur permet d'éviter tout chevauchement de sommets puisque la distance séparant les sommets n'est plus la distance entre les « centres » de ces sommets mais la distance entre leurs « contours ».

Post-traitement

La dernière étape de notre algorithme de dessin consiste à rajouter les arêtes supprimées lorsque l'on a extrait le sous-graphe planaire. Ces arêtes sont positionnées sur la face extérieure en utilisant un algorithme de dessin orthogonal avec trois points de contrôle par arête. Cette technique de placement des arêtes « non-planaires » est inspirée des représentations faites à la main (*cf* figure 6.5).

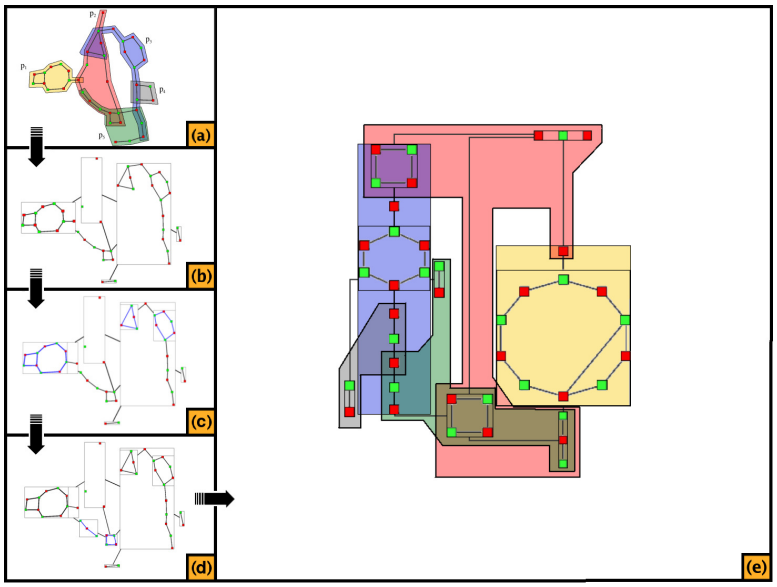

FIG. 6.18: Résultats de notre approche sur le graphe de la figure 6.8. (a), (b), (c) et (d) reprennent les figures 6.8, 6.9, 6.11 et 6.12 ; (e) Le résultat de notre algorithme de dessin sur le graphe de la figure (d). Chaque voie métabolique est entourée par une enveloppe de couleur.

Les figures 6.18 et 6.19 montrent respectivement les dessin obtenus par notre algorithme sur le graphe de la figure 6.8.(d) et sur le réseau métabolique de Escherichia Coli (E. Coli). Cet organisme a été longuement étudié, son métabolisme est composé de 198 voies, 1140 composés et réactions (i.e. noeuds) et 1321 liens (i.e. arêtes) entre eux.

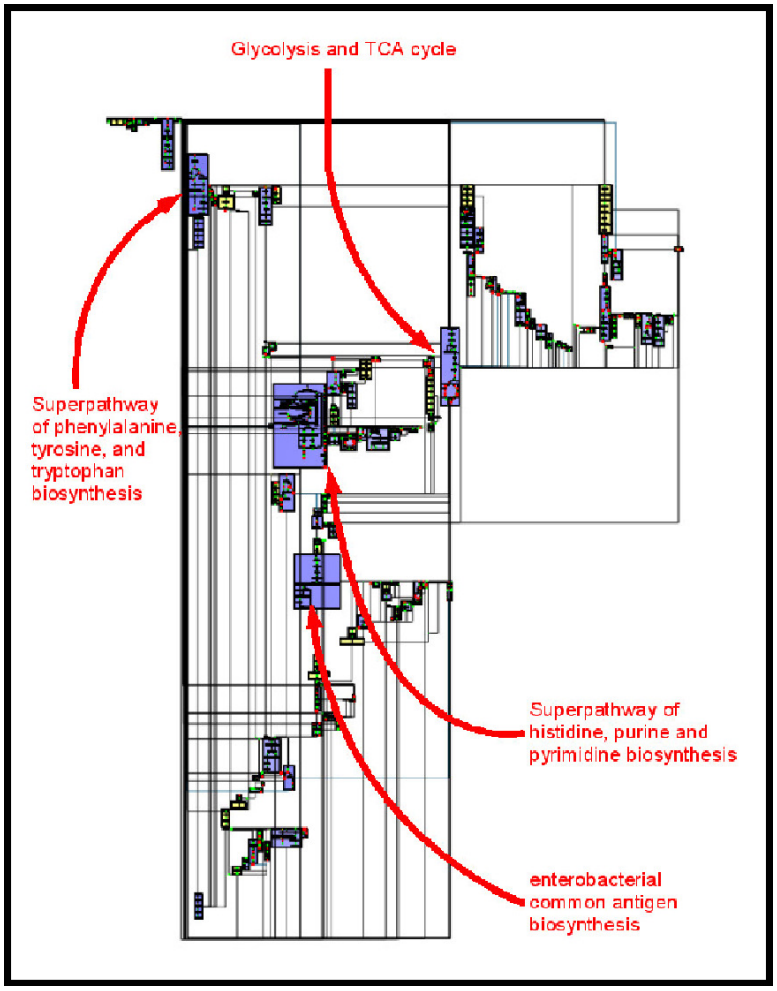

FIG. 6.19: Réseau métabolique complet de *E. Coli* dessiné par note approche. Les métanoeuds mauves représentent les voies métaboliques dessinées correctement. Les métanoeuds jaunes correspondent eux aux structures topologiques particulières (cycles et chaînes) détectées.

6.6 Résultats : comparaison aux méthodes existantes

Dans cette section, nous donnons tout d'abord le temps de calcul de notre approche sur un jeu de 27 organismes, puis nous comparons nos résultats (qualitatifs) à ceux obtenus par les outils les plus utilisés dans ce domaine, c'est-à-dire, Cytoscape [118] et Pathways Tool cellular overview diagram [108] (et par conséquent BioCyc). Cette comparaison est faite sur trois niveaux de visualisation :

- Visualisation du réseau entier ;
- Visualisation de voies métaboliques ;
- Visualisation d'une voie métabolique dans son contexte.

6.6.1 Temps de calcul

Il semble extrêmement difficile de calculer la complexité de notre algorithme. En effet, le nombre de noeuds et de métanoeuds du graphe quotient est étroitement lié aux données (e.g. le nombre de voies métaboliques, les chevauchements de voies, etc...). Toutefois, nous savons que la méthode utilisée pour extraire le sous-graphe planaire du graphe quotient a une complexité $O(|V_Q|^2 + |V_Q| \cdot |E_Q|)$, où V_Q et E_Q sont respectivement les nombres de sommets et d'arêtes du graphe quotient. D'autre part pour la recherche des plus longs cycles, nous utilisons une méthode exacte qui est exponentielle en temps. Il faut cependant nuancer ces complexités puisque la recherche des plus longs cycles mais aussi l'extraction du sous-graphe planaire ne sont effectuées que sur le graphe quotient et/ou sur les voies métaboliques. Pour évaluer l'efficacité « réelle » de notre algorithme, nous avons par conséquent mesuré le temps pris par chaque étape sur un jeu de 27 réseaux métaboliques.

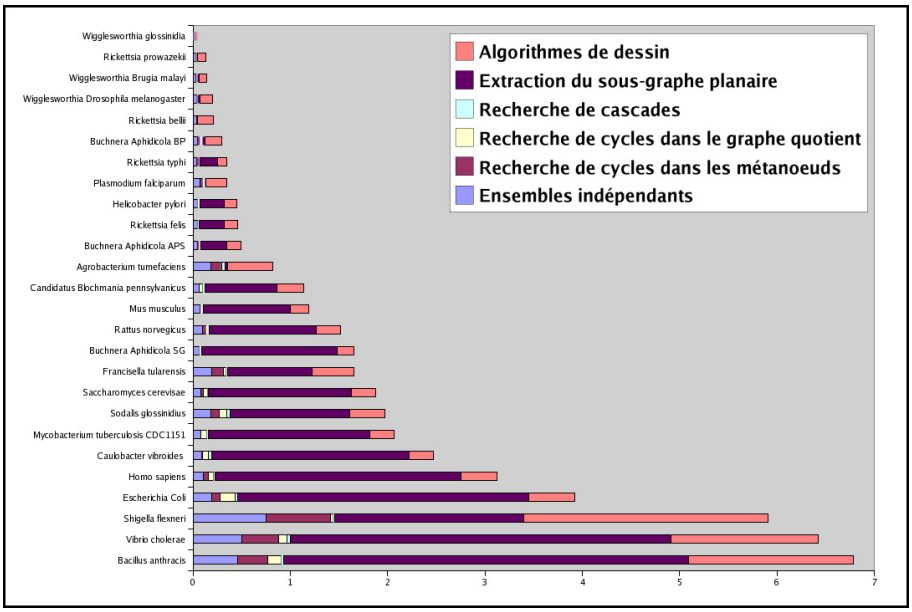

FIG. 6.20: Temps de calcul en secondes de notre algorithme pour chacun des 27 organismes.

La figure 6.20 montre le temps de calcul de notre algorithme sur chacun des 27 organismes du jeu de données. On peut remarquer que les temps varient beaucoup (de $0,05$ seconde pour *Wigglesworthia glossinidia* à $6,78$ secondes pour *Bacillus anthracis*), cela est dû aux tailles variables des réseaux métaboliques du jeu de données. Cette variation de taille est liée au fait que certains organismes ont des réseaux métaboliques très simples, mais aussi au fait que les réseaux de certains organismes ne sont pas complètement décrits dans les bases de données. Par exemple, *Wigglesworthia glossinidia* ne possède que 189 composés/réactions et 163 arêtes répartis sur 53 voies métaboliques tandis que *Bacillus anthracis* possède 892 composés/réactions et 1033 arêtes répartis sur 248 voies. Quoi qu'il en soit, même sur les organismes les plus complexes, notre algorithme permet d'obtenir une visualisation en des temps corrects (sur notre jeu de données, inférieurs à 7 secondes).

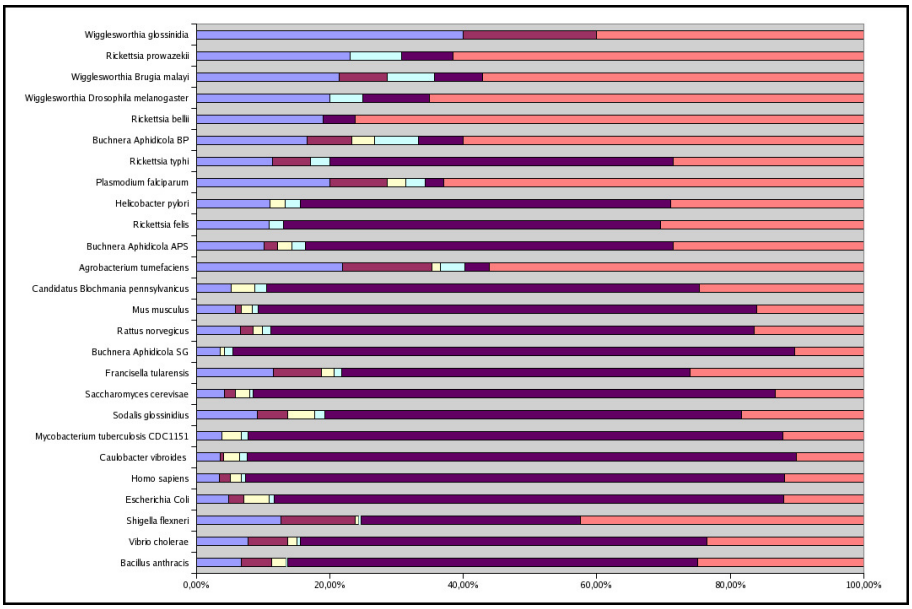

FIG. 6.21: Pourcentage de temps passé lors de chaque étape.

La figure 6.21 montre pour chaque organisme le pourcentage du temps pris par chaque étape comparé au temps total de l'algorithme sur cet organisme. Ce graphique met en évidence deux faits intéressants. Premièrement, pour une majeure partie des organismes du jeu de données, l'étape la plus longue est l'étape d'extraction du sous-graphe planaire. En effet, cette étape coûte en moyenne 61% du temps nécessaire à notre algorithme. Et ce, bien que les graphes quotients des organismes les plus simples soient « quasiment » planaires et que par conséquent le temps nécessaire à cette étape (pour ces organismes) soit très faible. Et deuxièmement, le temps nécessaire au calcul des plus longs cycles est relativement faible : en moyenne 4% pour la recherche dans les métanoeuds et 2% dans le graphe quotient. La recherche de ces cycles ne prend donc que 6% du temps total nécessaire à notre algorithme.

6.6.2 Visualisation du réseau entier

La figure 6.19 montre le dessin réseau complet de E. Coli calculé par notre approche. Contrairement au dessin obtenu par Cytoscape [118] sur les mêmes données (figure 6.22), notre algorithme permet d'organiser le réseau en métanoeuds. Les métanoeuds mauves sont ceux représentant les voies métaboliques de l'ensemble indépendant et sont par conséquent dessinés correctement (les sommets de ces voies sont dessinés proches les uns des autres). Ces voies métaboliques forment le squelette du dessin, et peuvent être changées en paramétrant les centres d'intérêts (voir la section 6.5.2).

La figure 6.23 montre le résultat obtenu par Pathway Tools cellular overview diagram sur ces mêmes données. La visualisation offerte par Pathway Tools cellular overview diagram est une vue sur chacune des voies métaboliques. Chaque sommet est dupliqué autant

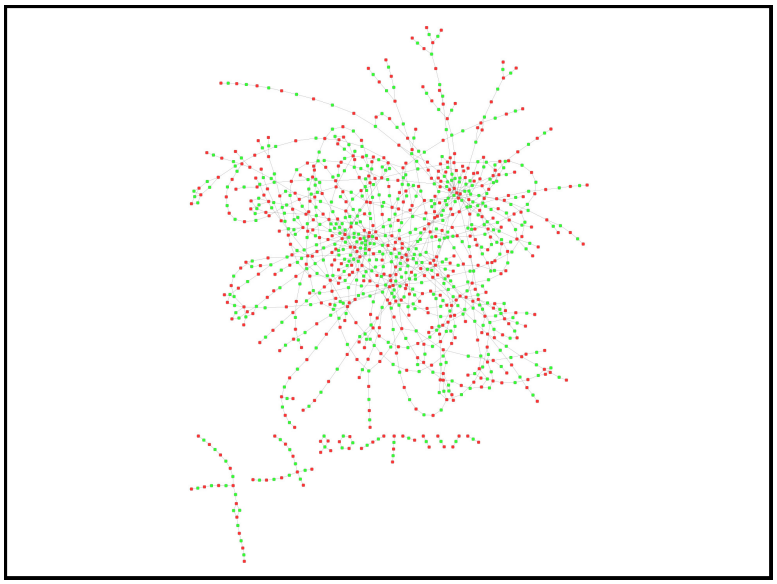

FIG. 6.22: Réseau métabolique complet de *E. Coli* dessiné par Cytoscape

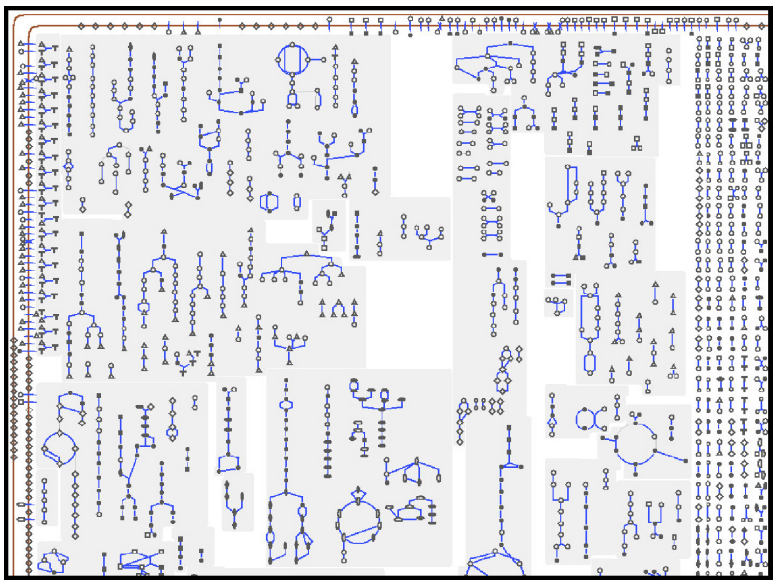

FIG. 6.23: Réseau métabolique complet de *E. Coli* dessiné par Pathway Tools cellular overview diagram

de fois qu'il existe de voies métaboliques qui le contiennent. Cette méthode de visualisation est donc une méthode complémentaire à la nôtre, puisqu'elle offre un ensemble de vues locales sur le réseau. Notre approche quant à elle permet de visualiser les voies métaboliques dans un contexte global. Par exemple, la figure 6.24.(a) est un agrandissement de la figure 6.19.

6.6.3 Visualisation de voies métaboliques

6.6.3.1 Cas d'étude : dessin du cycle de Krebs

En ce qui concerne la visualisation de voies métaboliques, nous ne comparons pas nos résultats à ceux de Cytoscape, en effet le but de ce logiciel n'est pas de dessiner des voies métaboliques mais des réseaux complets.

Dans les données extraites de BioCyc, le cycle de Krebs (aussi appelé *TCA cycle*) est contenu dans une super-voie métabolique : "glycolysis, pyruvate dehydrogenase, TCA, and glyoxylate bypass". Etant donné le nombre de sommets qu'elle contient, cette super-voie a été détectée par l'algorithme pour être correctement dessinée : tous les sommets (composés et réactions) impliqués dans cette super-voie sont regroupés en un seul méta-noeud (figure 6.24.(a)). Le dessin obtenu par notre méthode est extrêmement similaire à celui obtenu par le visualiseur de voie métabolique de BioCyc (figure 6.24.(b)). Les différences entre ces deux représentations sont principalement dues aux modèles utilisés pour représenter les voies/réseaux métaboliques : dans le modèle de BioCyc les réactions sont représentées par une arête et non un sommet (comme dans le modèle biparti).

6.6.3.2 Cas d'étude : dessin de la biosynthèse de la valine

Cette voie métabolique est composée d'une chaîne de quatre réactions partant du *pyruvate* et terminant par la *L-valine*.

Nous présentons ici deux cas :
– algorithme de partitionnement non paramétré, puis
– paramétré

Si l'algorithme de partitionnement n'est pas paramétré, cette voie métabolique n'est pas correctement dessinée. En effet, certains de ces composés et réactions appartiennent à de plus grandes voies métaboliques : la super-voie "glycolysis, pyruvate dehydrogenase, TCA, and glyoxylate bypass" et la super-voie de la *biosynthèse du coenzyme A et de la pantothenate*. La voie de la biosynthèse de la valine se retrouve donc séparée en trois parties, deux parties incluses dans ces deux voies et une partie partagée par plusieurs autres voies (et détectée comme une cascade de réactions). Dans la figure 6.25, la voie métabolique de la biosynthèse de valine est mise en surbrillance rose.

Quoi qu'il en soit, nous ne considérons pas que ce résultat soit négatif. En effet, cela signifie que cette voie métabolique partage plusieurs éléments (composés et réactions) avec d'autres voies, montrant ainsi l'interdépendance des ces voies.

Au contraire, si l'algorithme de partitionnement est guidé et que la voie métabolique de la biosynthèse de la valine est choisie comme centre d'intérêt, notre approche permet de la représenter efficacement (figure 6.26). Evidemment, ce choix amène à déconnecter les voies métaboliques partageant des composés (ou réactions) avec cette voie. Le paramétrage de notre approche est l'un de ces principaux intérêts puisqu'il permet de changer le squelette du dessin et le construire autour de voies métaboliques spécifiques.

Si l'on compare ce dessin à celui du visualiseur de voie métabolique de BioCyc (figure 6.27), on peut observer que l'ordre est inversé. Le *pyruvate* se trouve sur la gauche

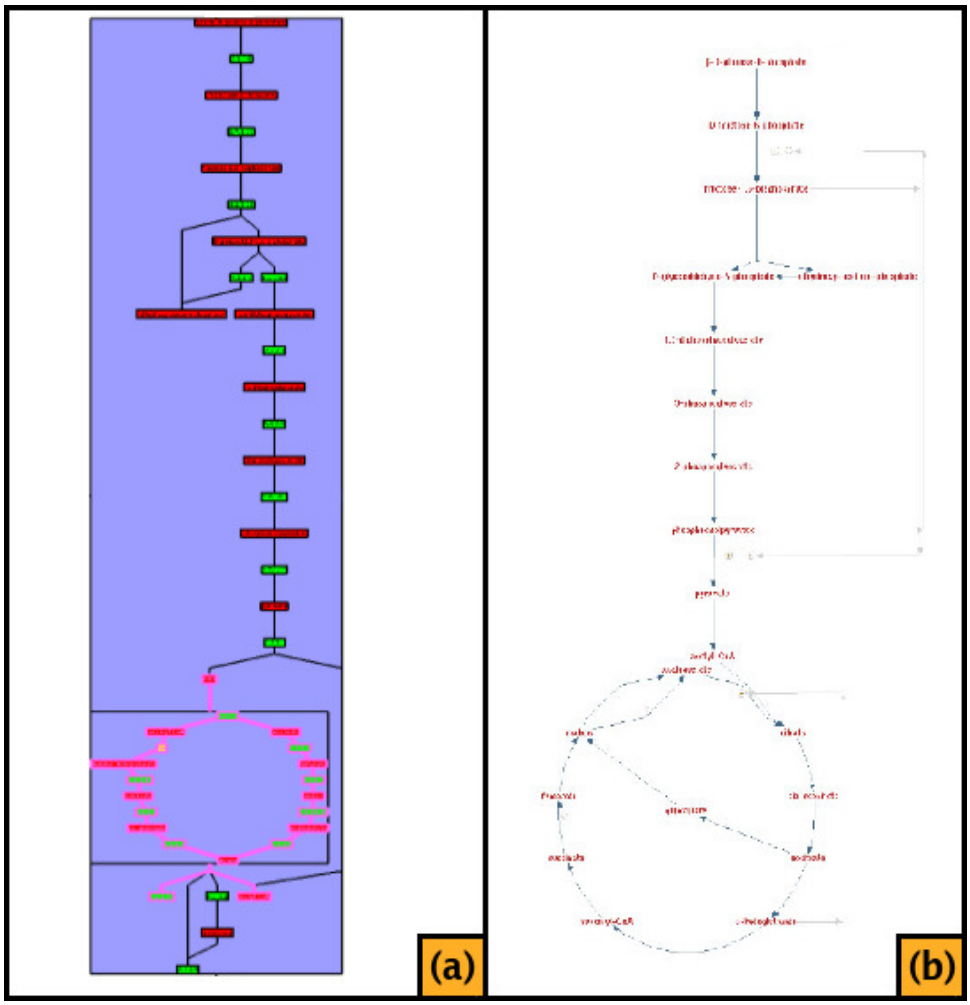

FIG. 6.24: La super-voie métabolique de « glycolysis, pyruvate dehydrogenase, TCA, and glyoxylate bypass ». (a) En utilisant notre méthode. Les sommets et les liens correspondants au cycle de Krebs sont entourés en rose ; (b) dans BioCyc [85]

du dessin de BioCyc tandis qu'il est en bas dans notre visualisation. Les composés d'entrée sont (généralement) positionnés en haut ou à gauche du dessin. Le *pyruvate* apparaît donc comme composé d'entrée de la voie métabolique dans le dessin BioCyc et sortie dans le nôtre. Cependant, dans BioCyc, toutes les réactions de la voie métabolique de biosynthèse de la valine sont décrites comme réversibles. Le *pyruvate* peut donc être considéré comme composé d'entrée aussi bien que comme composé de sortie de cette même voie.

6.6.4 Visualisation d'une voie métabolique dans son contexte

Dans notre approche, les liens entre les voies métaboliques sont représentés explicitement. Ces interconnections de voies sont ignorées lorsque l'on représente séparément chaque voie métabolique (comme dans BioCyc) ou lorsque l'on n'affiche pas d'information sur les voies métaboliques (comme dans Cytoscape). Une option de Pathway Tools Cellular Overview diagram permet d'ajouter ces liens sur le dessin du réseau. La limite

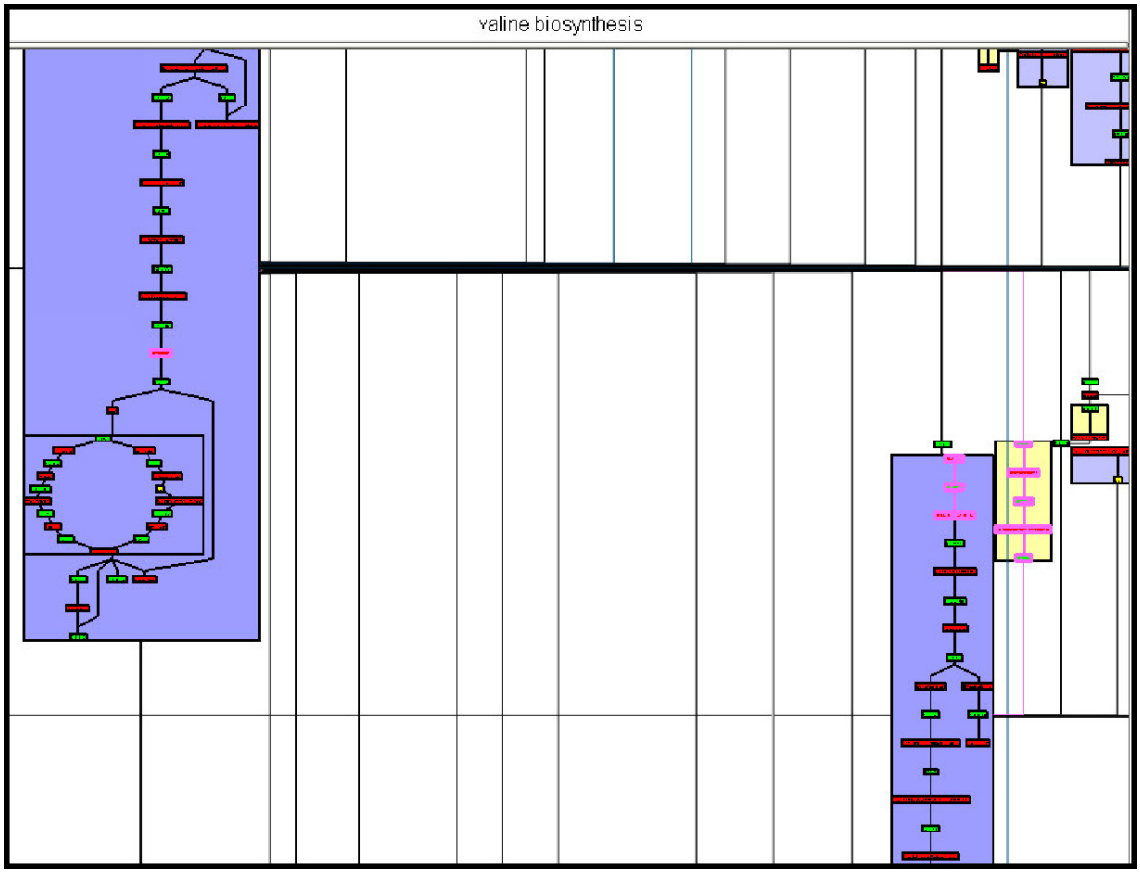

FIG. 6.25: Dessin de la voie de la biosynthèse de la valine en utilisant notre méthode et sans paramétrer l'algorithme de partitionnement. Les sommets correspondants sont entourés en rose. On peut voir qu'ils sont partagés sur trois métanoeuds.

de cette approche est que, étant donné que ces liens ne sont pas pris en compte lors du calcul du dessin original, le dessin final peut devenir très dense et difficile à lire.

Une tâche intéressante est de visualiser le voisinage d'un sommet. La figure 6.28 montre les voisins directs de la voie de la biosynthèse de la valine (entourés en rose). On peut facilement suivre chaque arête pour voir à quels sommets cette voie métabolique est connectée.

La figure 6.29 montre les liens entre la voie de la biosynthèse de la valine et le reste du réseau tels qu'ils sont affichés dans Pathway Tools cellular overview diagram. Etant donné qu'un certain nombre de sommets ont été dupliqués et que le dessin est statique (ne change pas en fonction d'un ou plusieurs centres d'intérêts), le nombre d'arêtes affichées rend extrêmement difficile la lecture de ce dessin.

6.7 Conclusion

Dans ce chapitre, nous présentons un nouvel algorithme permettant de représenter les réseaux métaboliques. Cette approche permet d'aborder un problème difficile qui consiste

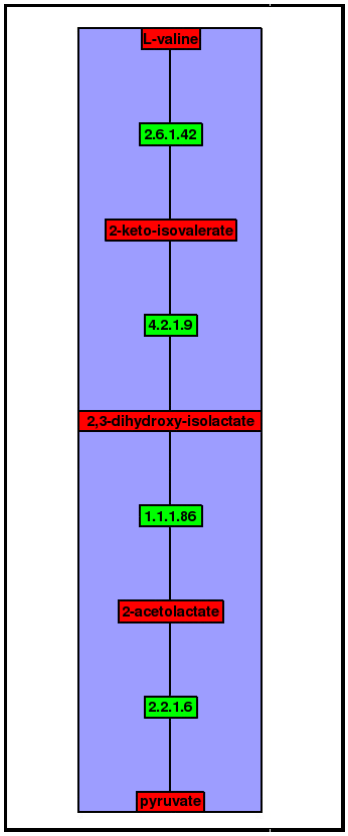

FIG. 6.26: Dessin de la voie de la biosynthèse de la valine en utilisant notre méthode et en guidant l'algorithme de partitionnement.

à représenter simultanément la topologie du réseau et la décomposition en voies métaboliques. Les voies métaboliques partageant souvent des composés et des réactions, les outils existants dupliquent ces sommets pour pouvoir représenter toutes ces voies dans une vue unique. Le principal désavantage d'une telle technique est que la connectivité dans le dessin n'est pas représentative de la connectivité réelle du réseau.

Contrairement à ces approches, notre algorithme permet de représenter sans duplication de sommet les réseaux métaboliques via un processus de partitionnement. Afin d'orienter ce partitionnement, l'utilisateur peut donner en paramètre une liste de voies métaboliques d'intérêt. De plus, l'algorithme de partitionnement permet de suivre les conventions de dessin fixées par les biologistes en détectant les cycles et cascades de réactions.

La deuxième étape de notre algorithme consiste à dessiner le graphe quotient et les métanoeuds calculés lors de l'étape de partitionnement. Nous utilisons un algorithme de dessin de graphe planaire pour dessiner le graphe quotient afin de respecter les conventions de dessin. D'autre part, les cycles et les cascades de réactions sont respectivement dessinés en cercle et alignées. Enfin, les voies et les sous-voies métaboliques sont dessinées en utilisant un algorithme hiérarchique mettant ainsi en évidence l'organisation hiérarchique de ces voies.

Nous avons prévu d'améliorer la qualité globale du dessin. L'une des améliorations

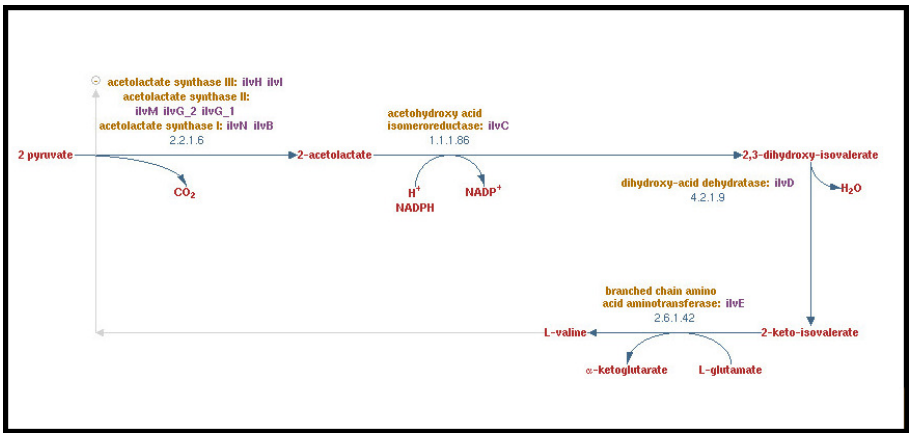

FIG. 6.27: Dessin de la voie de la biosynthèse de la valine dans BioCyc.

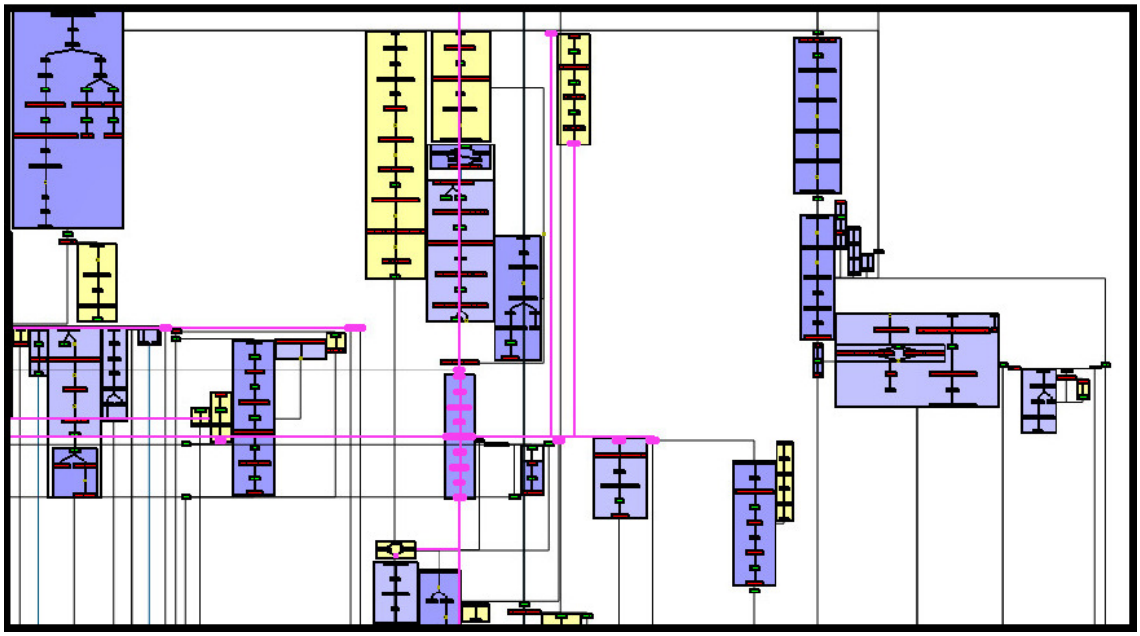

FIG. 6.28: Les sommets directement liés à la voie de la synthèse de la valine sont entourés en rose.

possibles est d'appliquer un algorithme de compaction tel que celui de Eiglsperger et Kaufmann [47], ou encore un algorithme permettant de supprimer les chevauchements de sommets tel que celui de Dwyer *et al.* [40, 41]. Cela aurait pour effet de « supprimer » les zones vides du dessin et ainsi d'augmenter la densité d'information.

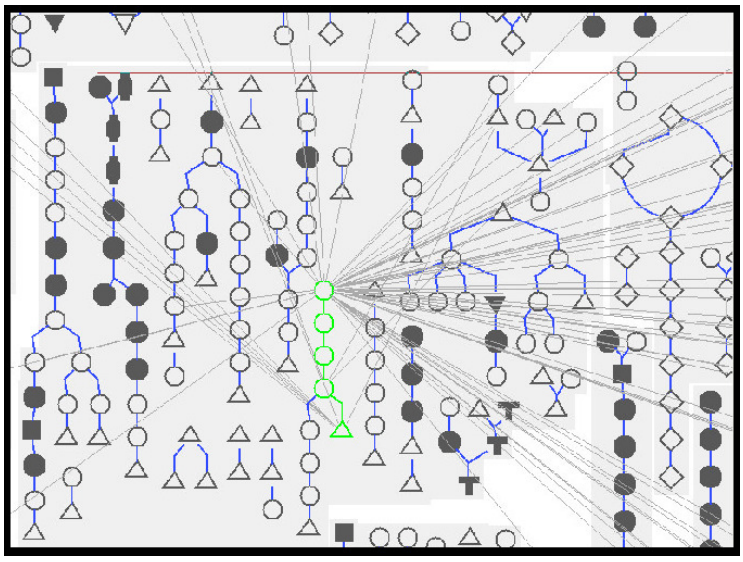

FIG. 6.29: Liens entre la voie de la biosynthèse de la valine dans Pathway Tools cellular overview diagram.

Chapitre 7

Algorithmes génériques ou dédié : évaluation sur une tâche biologique

Nous abordons dans ce chapitre la question de l'évaluation de dessin de graphe dans le cadre des réseaux métaboliques. Etant donnée une tâche précise, quelle représentation faut-il utiliser pour augmenter les performances de l'utilisateur ? Nous nous intéressons ici à l'évaluation de représentations de réseaux métaboliques. En particulier, nous comparons l'algorithme présenté dans le chapitre 6, appelé dans ce chapitre algorithme *MetaViz*, à deux algorithmes génériques fréquemment utilisés par la communauté de dessin de graphe. Nous posons dans un premier temps la problématique liée à la visualisation de réseaux métaboliques dans la section 7.1, puis nous donnons le modèle et les algorithmes de dessin que nous avons évalués dans la section 7.2. Dans la section 7.3, nous présentons la méthodologie et le protocole expérimental de notre évaluation. Enfin dans les section 7.4 et 7.5, nous donnons les résultats de l'expérimentation et l'analyse que nous en avons fait.

7.1 Problématique et motivations

Proposer un outil de visualisation utile pour les biologistes consiste souvent à trouver un compromis entre utilisabilité et respect de certaines conventions de représentation. Comme décrit dans le chapitre 6, les biologistes ont défini au cours des années un certain nombre de contraintes de dessin de réseaux biologiques [102]. Les performances des techniques utilisées et le nombre de projets de recherche sur les réseaux biologiques ayant explosé ces dernières années, le nombre de réseaux générés ainsi que la complexité[1] de ces réseaux ne permet plus d'en faire des représentations manuelles. De ce fait, un certain nombre de recherches a été fait pour trouver des algorithmes de dessin de réseaux biologiques respectant les conventions de représentation biochimique [17, 131, 22]

La plupart des algorithmes proposés par la communauté de dessin de graphe ne sont pas dédiés à un type de données particulier, mais sont plutôt des algorithmes génériques permettant de dessiner des données fictives (e.g. [16, 86]). En effet, nombre d'entre eux ne permettent pas par exemple de gérer les tailles des sommets, essentielles pour visualiser les

[1] Jusqu'à plusieurs milliers de sommets et arêtes [84].

étiquettes des sommets et/ou n'exploitent pas les informations intrinsèques aux données. La question traitée dans ce chapitre est l'évaluation des performances de tels algorithmes sur des réseaux métaboliques comparées à celle d'un algorithme dédié à la représentation de ces réseaux. En d'autres termes, les biologistes doivent-ils abandonner les conventions de dessin utilisées pour la représentation de réseaux biologiques ? Ou ces conventions permettent-elles de « mieux » visualiser ces réseaux ?

Dans ce chapitre, nous menons une étude empirique permettant de comparer les performances de trois algorithmes de dessin pour une tâche de recherche d'occurrences de *motifs* (cf définition partie 7.3.1). Nous comparons trois approches de dessin : une méthode par analogie physique, une méthode hiérarchique et une méthode utilisant un algorithme de dessin planaire (algorithme utilisé pour dessiner le graphe quotient dans la partie 6.5.3).

7.2 Modèle et algorithmes de dessins

Dans cette section, nous présentons les pré-traitements nécessaires pour cette étude ainsi que les trois algorithmes évalués.

7.2.1 Pré-traitements

Les données utilisées pour cette étude, décrites dans le chapitre 6, sont extraites de [85]. Un pré-traitement est appliqué sur ces données afin de détecter des voies métaboliques et certaines structures topologiques (i.e. cycles et cascades de réactions). Pour ce faire, l'algorithme de décomposition de [22] (décrit dans la section 6.5.1) est appliqué. Le résultat de cet algorithme est un graphe quotient dont les métanoeuds sont des voies métaboliques et/ou des structures topologiques. Dans ce type de représentation, deux métanoeuds sont reliés par une arête si au moins deux sommets (un dans chaque métanoeud) sont reliés dans le réseau original (cf définition du graphe quotient dans la section 2.2). Le principal désavantage de ce type de visualisation est qu'elle ne met pas en évidence la connectivité « réelle » du réseau. En effet, on ne sait pas combien d'arêtes représente une méta-arête et surtout quels sommets sont reliés par l'une de ces arêtes. Cependant elle permet d'avoir une abstraction du réseau et donc une vue globale des données.

En ce qui concerne le dessin des métanoeuds, chacun d'entre eux est dessiné en utilisant la technique présentée dans la section 6.5.3.1. Les trois algorithmes de dessin étudiés seront donc comparés sur la qualité du dessin du graphe quotient.

7.2.2 Algorithme par modèle de forces

Les algorithmes par modèle de forces [2] sont massivement utilisés puisqu'ils donnent des résultats visuellement plaisants. D'autre part, ces résultats mettent en évidence la structure générale du graphe. Ces algorithmes simulent des systèmes physiques où les sommets sont assimilés à des objets et les arêtes à des ressorts (physiques, électromagnétiques,...).

[2] Cf section 3.2.3 pour plus de détails sur les algorithmes par modèle de forces

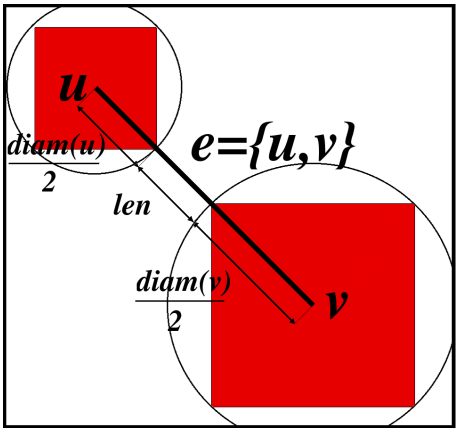

FIG. 7.1: Pour que GEM [54] prenne en compte la taille des sommets, la taille idéale de l'arête $e = \{u, v\}$ est la somme de len et des rayons des cercles englobant de u et de v.

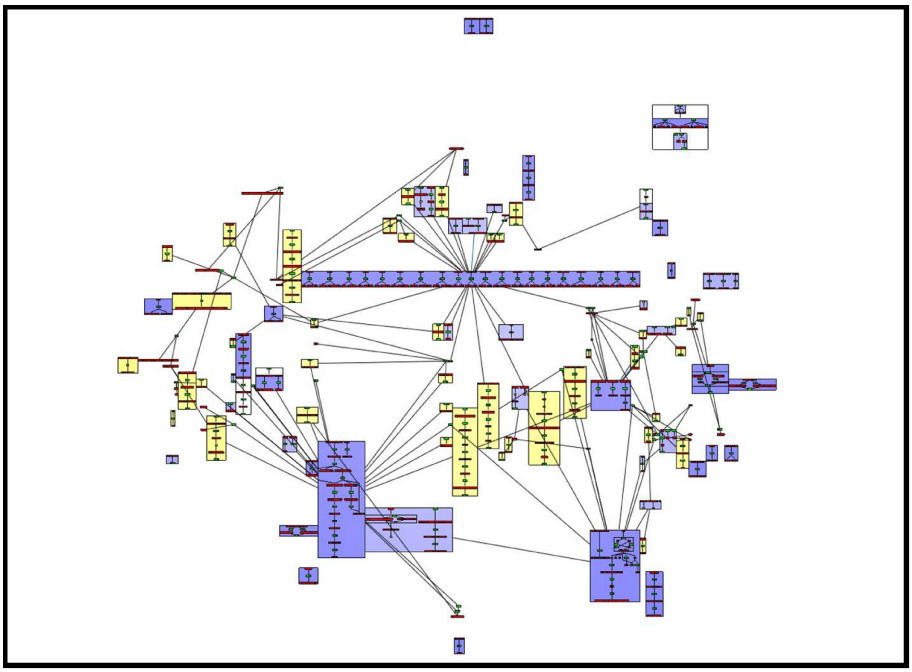

FIG. 7.2: Résultat de l'algorithme par modèle de forces sur le graphe quotient de Buchnera Aphidicola BP.

Dans les graphes ainsi dessinés, il existe donc une corrélation entre distance euclidienne et distance dans le graphe, et par conséquent ce type d'algorithmes donne une représentation intuitive du graphe. Il existe de nombreux algorithmes de dessin par modèle de forces (e.g. [42, 54, 59]). Nous avons choisi d'utiliser l'algorithme GEM [54] pour cette évaluation puisque cet algorithme donne des résultats particulièrement bons en terme d'étirement (i.e. le ratio entre les distances dans le dessin et distances dans le graphe). Le principal désavantage de cet algorithme est que sa complexité en temps est $O(|V|^3)$. Cependant les tailles des graphes quotients (de l'ordre de 120 sommets et 130 arêtes) permettent l'utilisation de cet algorithme.

Le système physique simulé par l'algorithme GEM [54] tente de mettre la taille des arêtes à une taille « idéale ». Etant donné que cet algorithme ne prend pas en compte les tailles des sommets, cette taille idéale est fixe pour toutes les arêtes. Pour prendre en compte la taille des sommets, nous utilisons une méthode similaire à celle décrite dans [67]. L'idée principale est que les arêtes ne doivent plus avoir des tailles uniformes. La notion de taille idéale devient donc propre à chaque arête. Soient $e = \{u, v\}$ une arête du graphe et $ideal(e)$ la taille idéale de e, on a :

$$ideal(e) = \frac{diam(u)}{2} + len + \frac{diam(v)}{2} \tag{1}$$

où len est la distance minimale désirée entre les contours des sommets et $diam(u)$ est le diamètre du cercle englobant du sommet u (voir la figure 7.1). Ceci ne permet pas d'interdire complètement les recouvrements de sommets mais en génère beaucoup moins que l'algorithme original.

Afin de supprimer les recouvrements de sommets restants, nous utilisons l'algorithme de [40, 41]. Dans cet algorithme, un système d'équations permet de représenter les contraintes de séparation (e.g. « un sommet u doit être positionné à droite d'un sommet v »). Un moteur de résolution de contraintes permet ensuite de trouver de nouvelles positions de sommets (sans recouvrement) tout en minimisant leurs déplacements.

La figure 7.2 montre le résultat de cet algorithme sur le graphe quotient obtenu par la méthode décrite dans la section 6.5 sur l'un des réseaux métaboliques utilisés dans cette évaluation (i.e. *Buchnera Aphidicola BP*).

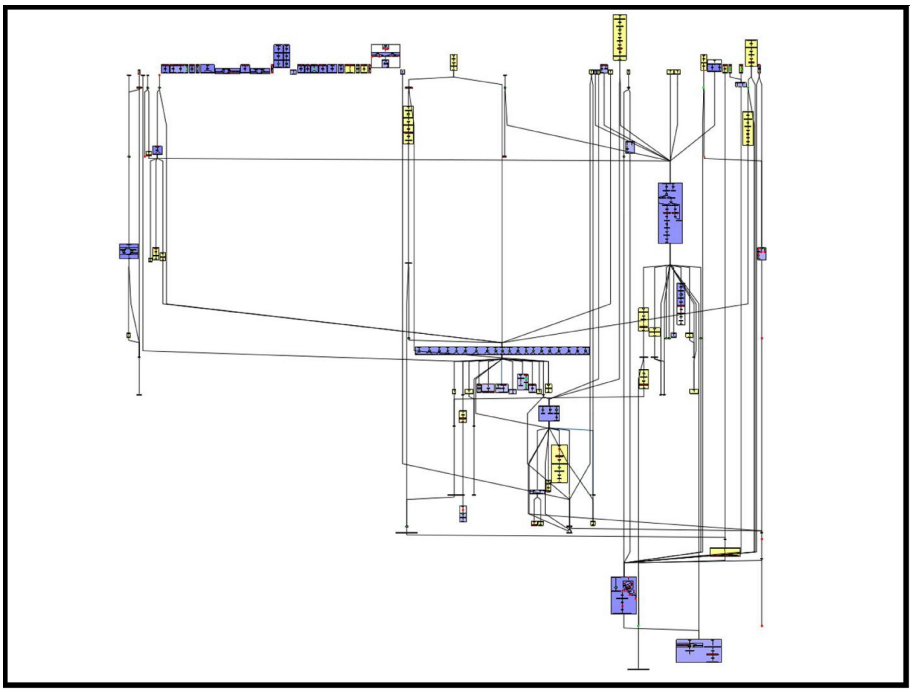

FIG. 7.3: Résultat de l'algorithme hiérarchique sur le graphe quotient de Buchnera Aphidicola BP.

7.2.3 Algorithme Hiérarchique

Le deuxième type d'algorithme évalué dans cette étude est un algorithme hiérarchique. Ces algorithmes organisent les sommets en couches horizontales et permettent ainsi de mettre en évidence l'organisation hiérarchique des données (cf section 3.2.2).

Ce type d'algorithme a été largement utilisé, notamment pour représenter des voies biologiques [83, 116, 36]. En effet, les voies biologiques (e.g. les voies métaboliques) contiennent une organisation hiérarchique puisqu'il existe un ordre dans les réactions, du (des) composé(s) d'entrée au(x) composé(s) de sortie.

Il existe de nombreuses versions d'algorithmes hiérarchiques (e.g. [123, 12, 48]). Nous avons choisi d'utiliser l'algorithme hiérarchique présenté dans [12]. Cet algorithme est une amélioration en terme de complexité en espace de l'algorithme de Sugiyama *et al.* [123] (pour plus de détails, voir le chapitre 3).

La figure 7.3 montre le résultat obtenu en appliquant l'algorithme de Auber [12] au graphe quotient de la figure 7.2.

7.2.4 Algorithme MetaViz

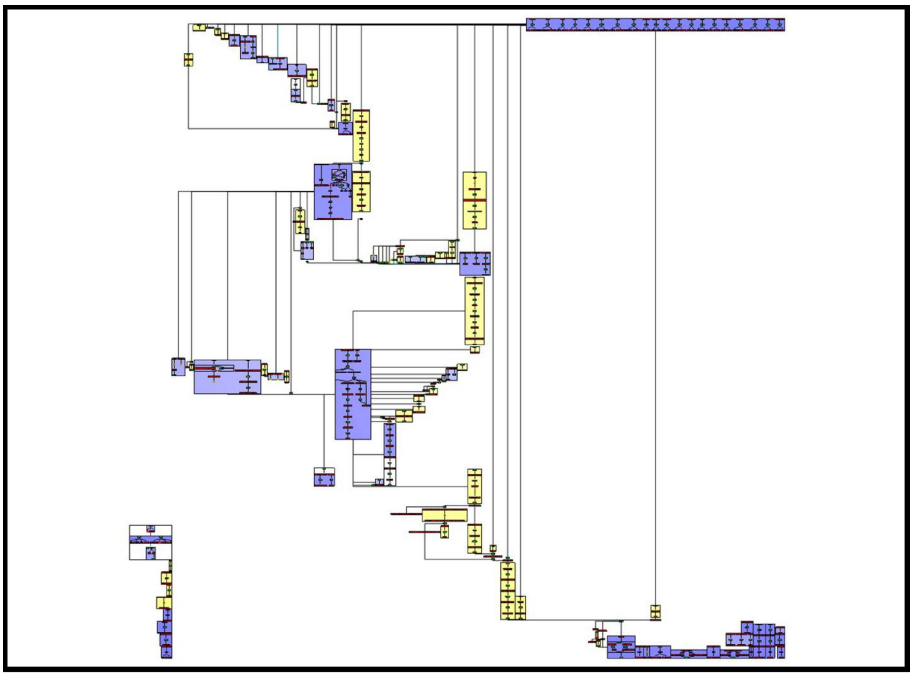

FIG. 7.4: Résultat de l'algorithme MetaViz sur le graphe quotient de Buchnera Aphidicola BP.

Le troisième algorithme dont nous avons évalué l'efficacité est l'algorithme de dessin utilisé dans le logiciel MetaViz [22]. MetaViz [22] est le logiciel intégrant le système de visualisation décrit dans le chapitre 6. Dans ce chapitre, nous appelons algorithme MetaViz, l'algorithme de dessin utilisé pour dessiner le graphe quotient obtenu après l'étape de partitionnement de l'algorithme de [19, 22].

La figure 7.4 montre le résultat obtenu en appliquant l'algorithme MetaViz au graphe quotient de la figure 7.2 et 7.3.

7.3 Méthodologie

7.3.1 Tâche et données biologiques

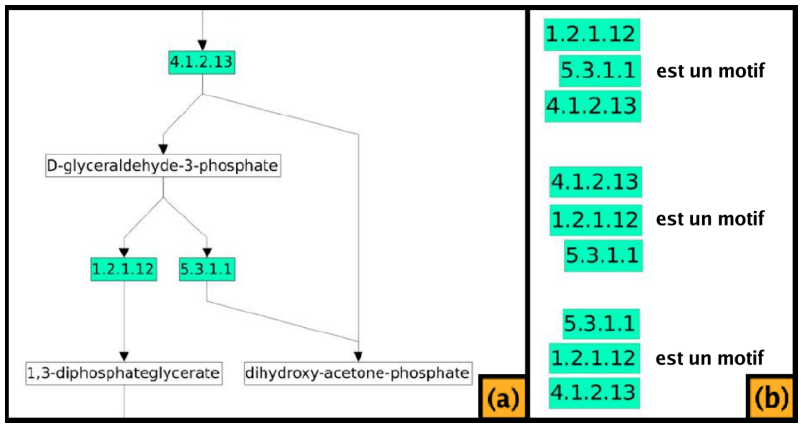

FIG. 7.5: (a) Exemple d'une occurrence du motif {1.2.1.12, 5.3.1.1, 4.1.2.13} ; (b) Ces trois ensembles représentent le même motif. Notez qu'un motif n'est pas nécessairement un chemin.

Pour mener à bien cette étude, nous avons choisi les réseaux métaboliques de trois organismes différents. Ces trois organismes sont très proches du point de vue biologique, puisque ce sont trois *genres* différents de bactéries *Buchnera Aphidicola* (un endosymbiote des aphidiens, e.g. les pucerons) : *Buchnera Aphidicola APS* (graphe A), *Buchnera Aphidicola BP* (graphe B) et *Buchnera Aphidicola SG* (graphe C). Ces organismes étant très proches dans la taxonomie des êtres vivants, ils sont aussi proches en terme de fonctionnalités (ces organismes peuvent dégrader ou synthétiser à peu près les mêmes composés). De plus, leurs réseaux métaboliques sont aussi proches en terme de taille (503 sommets/526 arêtes, 558 sommets/538 arêtes, et 562 sommets/559 arêtes). L'utilisation d'organismes similaires permet d'obtenir des temps de réponse des participants similaires sur les trois réseaux et donc pour un même motif, des tâches de difficultés similaires.

Comme décrit dans la partie 7.2.1, l'algorithme de partitionnement présenté dans la section 6.5 a été appliqué sur ces trois réseaux, donnant ainsi trois graphes quotients. Pour chacun des graphes quotients, nous avons trois représentations, une pour chaque condition de dessin (i.e. pour chaque algorithme de dessin). Nous avons donc 9 dessins de graphes au total.

La tâche que les participants à l'évaluation ont effectuée consiste à identifier dans un réseau des occurrences de motifs.

Définition 7.1 (Motif) *Un motif est un ensemble non-ordonné de réactions tel que chaque réaction du motif partage (au moins) l'un de ces réactants et/ou produits avec (au moins) une autre réaction du motif.*

La figure 7.5.(a) montre une occurrence du motif {1.2.1.12, 4.1.2.13, 5.3.1.1}. La figure 7.5 montre quant à elle trois manières d'écrire le même motif (puisqu'un motif est un ensemble non-ordonné).

L'intérêt majeur de cette expérimentation est qu'elle est faite sur une tâche biologique « réelle », la recherche d'occurrences de motifs. En effet, cette tâche est biologiquement pertinente puisque si un motif est répété dans le métabolisme d'un organisme, alors cela peut signifier qu'il y a eu duplication de gènes durant l'évolution de cet organisme. Cette information peut permettre d'aider à la reconstruction des réseaux métaboliques d'ancêtres de l'organisme en question.

Motifs	nb occurrences dans le graphe A		nb occurrences dans le graphe B		nb occurrences dans le graphe C	
	Contenues	Partagées	Contenues	Partagées	Contenues	Partagées
6.3.4.* 3.5.4.9 6.3.2.17	0	3	0	3	0	0
4.3.2.* 6.3.4.5 2.1.3.3	1	1	1	1	1	1
2.7.2.* 1.2.1.13 2.2.1.1	0	1	0	1	0	0

TAB. 7.1: Nombre d'occurrences de chaque motif contenu dans chaque réseau métabolique (graphes A, B et C) et indique combien sont contenues dans un métanoeud (contenues) et combien sont partagées sur plusieurs métanoeuds (partagées).

En utilisant le résultat de l'algorithme de Lacroix *et al.* [95], nous avons choisi trois motifs contenant chacun trois réactions. Afin d'éviter que les participants à l'évaluation n'apprennent les réponses durant l'expérience, le nombre d'occurrences de ces motifs varie entre 0 et 3 suivant l'organisme dans lequel est faite la recherche. Certaines occurrences des motifs choisis sont partagées par plusieurs métanoeuds (dans le graphe quotient) tandis que d'autres sont contenues entièrement dans un seul métanoeud. Le tableau 7.1 montre pour chaque motif choisi le nombre d'occurrences de ce motif dans chaque organisme et indique combien sont contenues dans un métanoeud et combien sont partagées sur plusieurs métanoeuds.

7.3.2 Hypothèse *a priori*

La figure 7.6 montre trois représentations d'une même sous-partie d'un réseau métabolique, celui de *Buchnera Aphidicola APS*. Le métanoeud central (la voie métabolique « superpathway of histidine, purine and pyrimidine biosynthesis ») est le même dans chacune des trois figures 7.6.(a), (b) et (c). On peut aussi remarquer que les facteurs d'agrandissement sont les mêmes puisque les métanoeuds centraux ont visuellement la même taille.

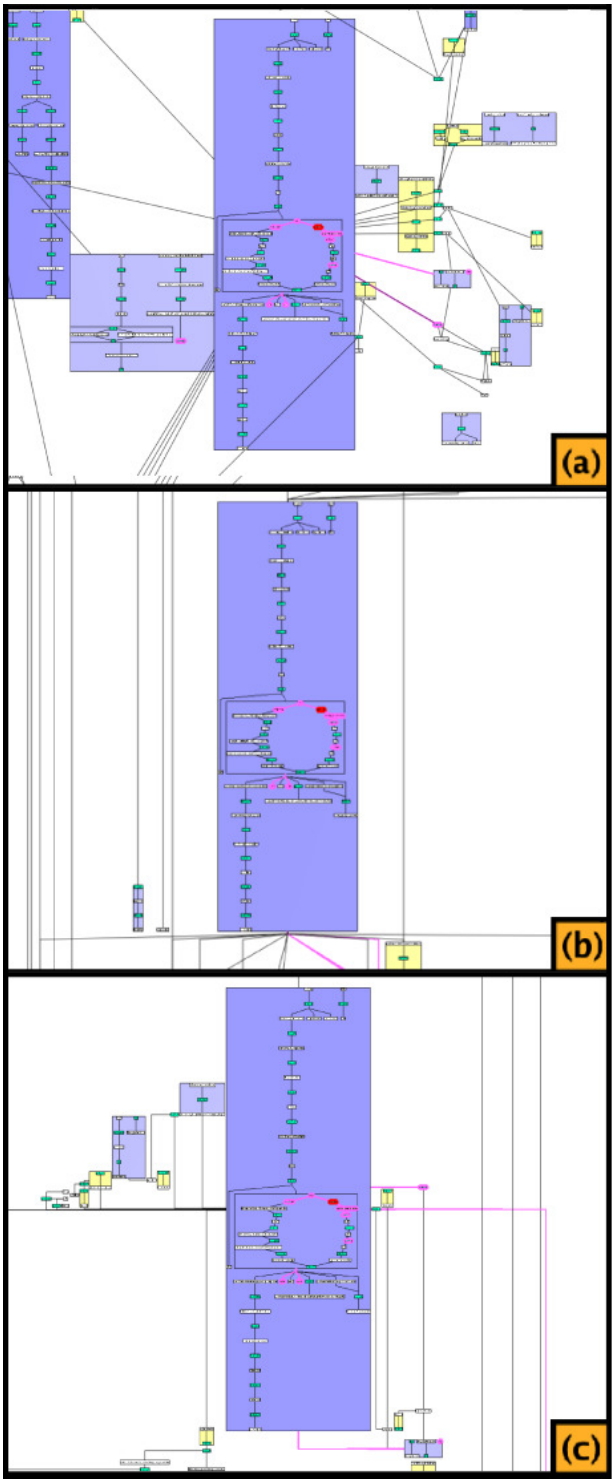

FIG. 7.6: Vue agrandie du réseau métabolique de *Buchnera Aphidicola APS* dessiné par (a) l'algorithme GEM, (b) l'algorithme hiérarchique et (c) l'algorithme MetaViz. Les chemins reliant le sommet entouré en rouge et les sommets à distance inférieure ou égale à deux du sommet entouré en rouge sont en surbrillance rose.

Dans chacune des trois représentations, les sommets, arêtes et méta-arêtes[3] qui sont à une distance inférieure ou égale à 2 du sommet entouré en rouge dans le réseau original ont été mis en évidence par une surbrillance rose. On remarque aisément que tous ces sommets et arêtes (et méta-arêtes) sont visibles dans la représentation du réseau faite par l'algorithme par modèle de forces (voir la figure 7.6.(a)). Au contraire, certains de ces sommets et arêtes ne sont pas visibles dans les dessins des algorithmes hiérarchiques et MetaViz (voir figures 7.6.(b) et (c)). Pour visualiser le voisinage (à distance 2) du sommet entouré en rouge, l'utilisateur devrait donc naviguer dans le graphe (« zoom and pan »).

Etant donné que la tâche consiste à chercher des motifs (cf section 7.3.1) et que cette tâche est étroitement liée à la connectivité, l'hypothèse que nous avions formulée *a priori* était que GEM donnerait de meilleurs résultats en terme de temps. Le principe des algorithmes par modèle de forces est de dessiner les sommets de telle sorte que les distances dans le dessin soient proches des distances dans le graphe. Les sommets voisins sont donc dessinés proches dans le dessin. Les algorithmes hiérarchiques et MetaViz n'ont quant à eux pas les mêmes objectifs puisqu'ils tentent de mettre en évidence une organisation structurelle du graphe (e.g. hiérarchique), d'obtenir une bonne distribution des sommets dans le plan et de limiter le nombre de croisements d'arêtes. Nous n'émettions donc aucune hypothèse sur les performances relatives de ces deux algorithmes.

7.3.3 Protocole expérimental

Notre évaluation a pour but de comparer trois algorithmes de dessin, sur trois réseaux et trois motifs différents. Chaque tâche étant une combinaison d'un algorithme de dessin, d'un réseau et d'un motif, nous avons donc 27 tâches différentes.

Pour permettre aux participants d'apprendre comment résoudre ce type de tâche, nous avons choisi aléatoirement 12 tâches d'apprentissage supplémentaires. Ces tâches étant les mêmes pour tous les participants, ils ont tous acquis la même expérience avant de commencer l'expérimentation « réelle ». Lors des cinq premières tâches d'apprentissage, nous avons aidé les participants à résoudre la tâche en leur indiquant comment trouver les réactions pertinentes. Ils ont d'autre part eu un retour sur la réponse à la tâche précédente (cf figure 7.8). Lors des sept dernières tâches expérimentales, ils n'ont pas été tenus au courant que ces tâches ne faisaient pas partie de l'expérimentation. Ces tâches ont été nécessaires pour que les participants acquièrent suffisamment d'expérience et soient par conséquent suffisamment performants (et ce, quelle que soit la condition de dessin). En effet, les performances sont étroitement liées à l'expérience acquise précédemment. La figure 7.7 montre l'évolution typique des performances d'un participant lambda au cours d'une expérimentation en fonction du nombre de tâches déjà effectuées.

Les participants ont ensuite effectué les 27 tâches expérimentales. Afin d'éviter les biais liés à l'expérimentation, les tâches ont été présentées à chaque participant dans un ordre aléatoire. Si les tâches avaient été présentées dans un ordre fixe, les performances sur les dernières tâches auraient été accrues (cf figure 7.7). D'autre part, pour éviter les

[3] Si une arête d'un de ces chemins n'appartient pas au graphe quotient, alors la méta-arête la représentant est elle mise en surbrillance.

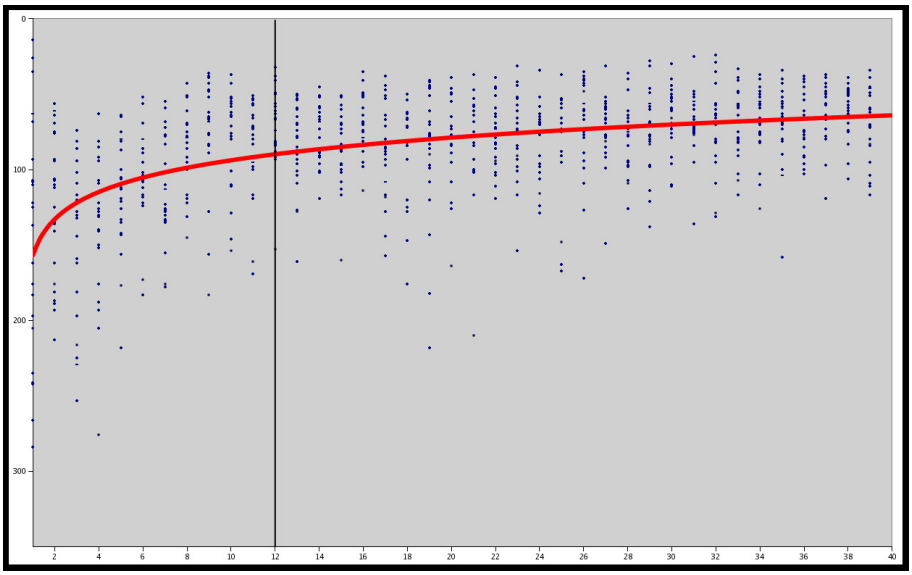

FIG. 7.7: Temps de réponse des participants en fonction des tâches précédemment effectuées. La courbe rouge est une régression logarithmique du temps moyen de réponse. Au delà de la douzième tâche, les participants ont acquis assez d'expérience pour commencer l'expérimentation « réelle ».

problèmes de fatigue, une courte pause a été donnée aux participants de manière régulière (toutes les cinq tâches).

7.3.4 Outils d'évaluation

La figure 7.8 montre une capture d'écran du logiciel d'évaluation utilisé lors de cette étude. Ce logiciel est organisé comme suit : la zone de visualisation du réseau se trouve sur la partie droite et le motif recherché sur la partie gauche (entouré en bleu dans la figure 7.8). Afin d'aider le participant dans la recherche des occurrences, trois sommets ont été entourés en rouge dans le dessin du réseau. Ces sommets sont des réactions appartenant potentiellement à une (au moins) occurrence du motif recherché. Dans l'exemple de la figure 7.8, ces sommets sont trois réactions dont l'étiquette commence par « 6.3.4. ». La tâche consiste à trouver parmi ces trois réactions lesquelles appartiennent à au moins une occurrence du motif recherché (dans la figure 7.8, le motif $\{6.3.4.*, 3.5.4.9, 6.3.2.17\}$). Entourer ces trois réactions en rouge s'est révélé nécessaire afin d'éviter aux participants de rechercher les occurrences du motif dans le réseau entier. Des tests pilotes nous ont permis de vérifier que malgré cette aide, la tâche restait suffisament compliquée et permettait d'obtenir des résultats « significatifs ».

Pour résoudre la tâche, l'utilisateur dispose d'un interacteur de « voisinage » (bouton 2 dans la figure 7.8). Cet interacteur permet de mettre en surbrillance rose les chemins de longueur 2 dans le réseau original partant d'un sommet donné[4] (dans la figure 7.9, les sommets et arêtes à distance au plus 2 du sommet entouré en rouge sont mis en

[4]Si une arête d'un tel chemin n'appartient pas au graphe quotient, alors la méta-arête la représentant est en surbrillance

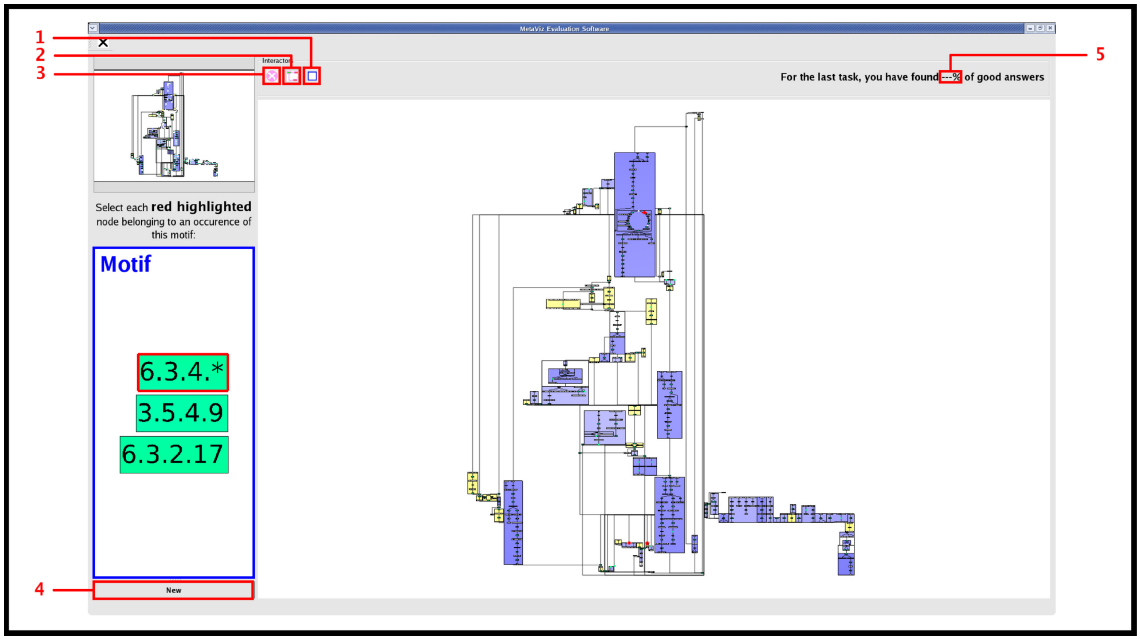

FIG. 7.8: Capture d'écran du logiciel d'évaluation. Les boutons 1 et 2 permettent de sélectionner et de trouver les réactions pertinentes. Le bouton 2 est un interacteur permettant de mettre en surbrillance le voisinage à distance 2 d'un sommet, et le bouton 1 de sélectionner les réactions pertinentes. Le bouton 3 permet quant à lui de supprimer toute surbrillance. Le bouton 4 permet de valider la réponse et d'obtenir la tâche suivante. Enfin, 5 montre la zone où le retour sur les réponses précédentes est donné lors des cinq premières tâches d'apprentissage.

surbrillance rose). Si l'utilisateur « clique » sur une réaction R, les extrémités des chemins en surbrillance seront les réactions partageant au moins un composé (d'entrée ou de sortie) avec R (voir la figure 7.9).

Supposons que l'on cherche une occurrence du motif $\{6.3.4.*, 3.5.4.9, 6.3.2.17\}$ et que la réaction R entourée en rouge porte l'étiquette 6.3.4.3. L'utilisateur doit alors vérifier des réactions étiquetées 3.5.4.9 et 6.3.2.17. Dans le cas le plus simple, « cliquer » sur R en utilisant l'interacteur 2 mettra en surbrillance rose les deux réactions recherchées (voir la figure 7.9). Il est cependant possible qu'une seule des deux autres réactions (par exemple 3.5.4.9) du motif soit mise en surbrillance. Ce cas arrive lorsque R est la première réaction d'une cascade de réactions : la dernière réaction de la cascade se trouve alors à une distance 4 de R. Il faut alors regarder le voisinage à distance 2 de 3.5.4.9 pour éventuellement trouver la dernière réaction du motif (ici 6.3.2.17).

Le participant peut ensuite sélectionner les réactions pertinentes en utilisant l'interacteur du bouton 1 (voir la figure 7.8). Lorsque le participant a vérifié que toutes les réactions entourées en rouge appartiennent ou non à une occurrence du motif recherché (et a sélectionné les réactions pertinentes), il « clique » sur le bouton 4 pour valider sa sélection et demander la tâche suivante.

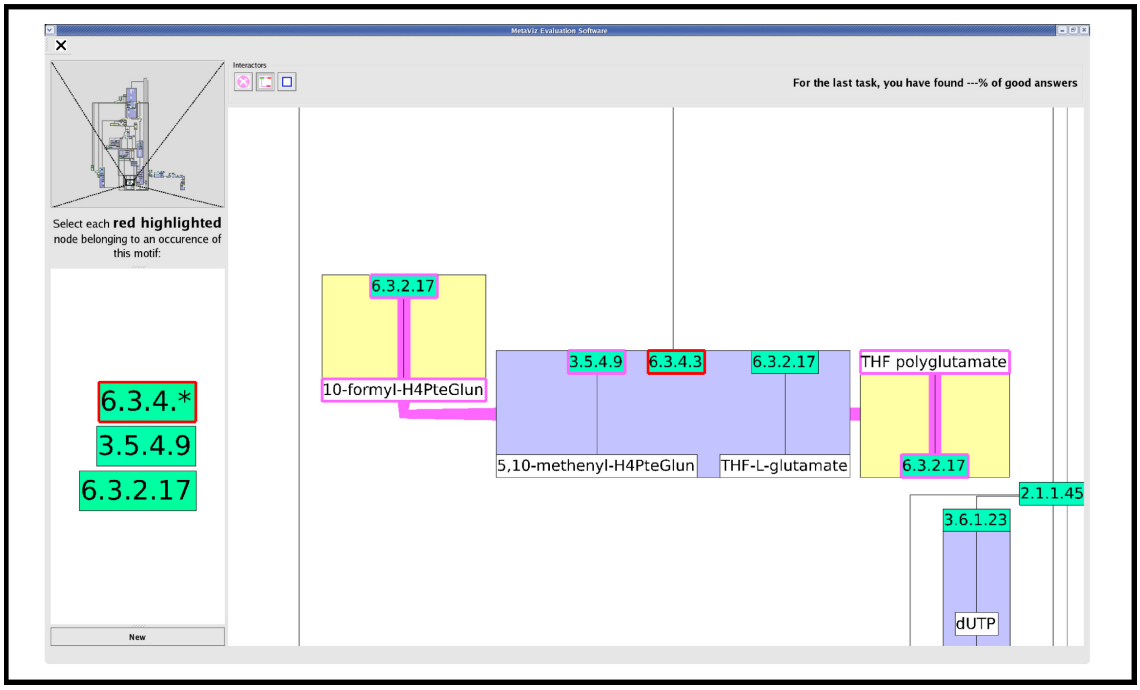

FIG. 7.9: En utilisant l'interacteur du bouton 2, « cliquer » sur la réaction étiquetée 6.3.4.3 met en surbrillance les sommets à distance au plus 2 dans le graphe original ainsi que les arêtes (et méta-arêtes) les reliant. Ici, la réaction 6.3.4.3 appartient bien au motif recherché ($\{6.3.4.*, 3.5.4.9, 6.3.2.17\}$) puisque les autres réactions du motif (3.5.4.9 et 6.3.2.17) ont été mises en surbrillance rose.

7.3.5 Processus expérimental

Pour mener à bien cette étude, nous avons choisi un panel de 22 participants (des universités de Glasgow et de Bordeaux 1). Tous ces participants sont issus de la communauté informaticienne, et parmi eux, sept ont des connaissances en bioinformatique. Nous avons délibérément choisi de ne pas prendre de biologistes dans cet échantillon. En effet, les biologistes que nous avons rencontrés nous ont indiqué que l'algorithme de dessin de MetaViz leur semblait plus familier. Nous ne voulions pas que cette familiarité puisse biaiser les résultats de notre étude.

Lors des tests d'évaluation aucun problème n'est survenu, le protocole mis en place s'est donc déroulé correctement. Le temps pris par chaque test d'évaluation (comprenant les tâches d'apprentissage, l'évaluation réelle ainsi qu'un questionnaire à la fin du test) a varié de 40 minutes à 1 heure et 35 minutes, la moyenne étant approximativement 1 heure. Le temps total de l'évaluation est de plus de 26 heures.

7.4 Résultats et analyse

Pour chaque participant, nous avons mesuré l'intervalle de temps nécessaire à la réalisation de chaque tâche, c'est-à-dire le temps entre l'affichage de la tâche et la validation de la réponse.

Nous avons aussi mesuré le taux d'erreurs de chaque réponse comme le pourcentage de réponses incorrectes par rapport au nombre total de réponses. Etant donné que les participants n'ont pas eu de limite de temps pour répondre à chaque tâche, le taux d'erreurs dans les résultats de l'évaluation est extrêmement faible (de l'ordre de 1.4%) et par conséquent le taux de réponses correctes est très élevé (plus de 98.5%).

Dans cette section, nous nous intéressons donc à l'analyse des résultats quantitatifs des participants (i.e. les performances en temps) sur chaque algorithme de dessin, puis aux résultats qualitatifs (i.e. questionnaire).

7.4.1 Résultats quantitatifs

7.4.1.1 Par condition de dessin

Les temps moyens de réponse (de tous les participants) pour chaque condition de dessin, sans distinction sur le réseau ni le motif recherché, sont donnés dans la figure 7.10.

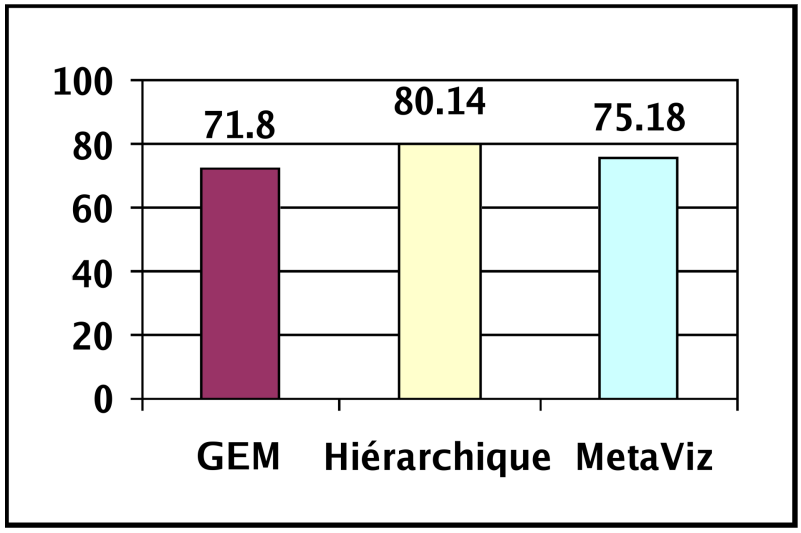

FIG. 7.10: Temps moyen de réponse pour les trois algorithmes de dessin, tout réseau et tout motif confondus.

Afin de déterminer s'il existe une différence statistiquement significative entre les performances des trois algorithmes, nous avons effectué une analyse statistique des données. Lorsque l'on effectue ces tests, il nous faut généralement spécifier ce que différence *significative* signifie en fixant le niveau de confiance. On considère généralement qu'un niveau de confiance de 95% est suffisant [71] puisque l'on a alors qu'une chance sur vingt que les différences trouvées soient dues au hasard.

Le premier test effectué est une analyse standard de type ANOVA (pour *ANalyse Of Variance*) [71], basée sur les valeurs critiques de la distribution F. La distribution F permet de savoir si la différence observée de variance entre les différents algorithmes de dessin a une probabilité d'existence inférieure à une probabilité α (ici, nous avons utilisé $\alpha = 0.05$ ce qui peut être interprété comme un niveau de confiance de 95%). D'une manière simplifiée, le test ANOVA consiste à comparer les variances « intra-groupe » (ici

les résultats moyens des participants pour chaque algorithme de dessin) à la variance « inter-groupe ». Pour qu'il y ait différence significative, il faut que la variance « intra-groupe » soit plus petite que la variance « inter-groupe ». Ce test permet donc de savoir s'il existe une différence significative dans les données, et ce avec un niveau de confiance donné. Par contre, il ne permet pas de savoir entre quelles conditions se trouve(nt) la (les) différence(s).

En utilisant un niveau de confiance de 95% ($\alpha = 0.05$), ce test a montré qu'il y a une différence significative entre les différentes conditions. En effet, la valeur de F trouvée est $F = 9.14$ ce qui est supérieur à la valeur attendue $F(2.42, \alpha = 0.05) = 3.23$.

Ayant trouvé une différence significative dans les données, il faut alors déterminer entre quelle(s) paire(s) de conditions se trouvent cette (ces) différence(s). Le méthode statistique utilisée pour trouver ces différences est le test Tukey [71], basé sur les valeurs critiques de HSD (pour *Honestly Significant Difference*). Ce test effectue des comparaisons de paires de conditions (ici, d'algorithmes de dessin). Comme pour le test ANOVA, les résultats du test sont donnés avec un taux de confiance.

Différences	GEM	Hiérarchique	MetaViz
GEM	#	8.34	3.38
Hiérarchique	8.34	#	4.95
MetaViz	3.38	4.95	#

TAB. 7.2: Différences trouvées lors du test Tukey entre chaque paire de conditions. Dans cette étude la valeur attendue est $HSD(3.44, \alpha = 0.05) = 4.77$.

Nous avons effectué le test Tukey avec une valeur $\alpha = 0.05$. Le tableau 7.2 montre les résultats obtenus lors du test Tukey entre chaque paire d'algorithmes de dessin. On peut interpréter les résultats du test Tukey comme suit :

1. Les différences de performances en temps entre l'algorithme de dessin hiérarchique et les algorithmes GEM et MetaViz sont statistiquement significatives : un temps moyen par tâche de 80.14 secondes pour l'algorithme hiérarchique contre 71.8 secondes pour GEM et 75.18 secondes pour MetaViz.
2. Il n'y a pas de différence statistiquement significative entre les performances de GEM et de MetaViz, malgré la différence constatée entre les temps moyens de GEM (71.8 secondes) et de MetaViz (75.18 secondes).

Dans notre étude, il est important de noter que si l'on augmente le niveau de confiance à 99% ($\alpha = 0.01$), le test ANOVA nous indique qu'il existe bien une différence significative dans ces résultats, mais le test Tukey ne trouve qu'une seule différence significative : l'algorihme hiérarchique offre de moins bons résultats que l'algorithme GEM. Cependant, augmenter le niveau de confiance à 99% augmente aussi la probabilité de ne pas trouver de différence significative alors qu'il y en a une. Nous considérons donc dans ce chapitre qu'un niveau de confiance de 95% est suffisamment élévé.

7.4.1.2 Autres conditions

Nous avons effectué d'autres tests statistiques afin de vérifier certaines attentes. Tout d'abord, comme nous l'espérions, il n'y a aucune différence significative entre les résultats obtenus sur les différents réseaux. Par contre, nous avons trouvé une différence entre les trois motifs (la valeur trouvée $F = 17.4$ est supérieure à la valeur attendue, 3.23), le motif $\{6.3.4.*, 3.5.4.9, 6.3.2.17\}$ étant plus difficile que les deux autres motifs. Cette différence n'est pas surprenante : ce motif est celui dont le nombre d'occurrences partagées sur plusieurs métanoeuds dans les graphes quotients est le plus important (voir le tableau 7.1), ce qui rend la tâche plus difficile à réaliser.

7.4.2 Résultats qualitatifs

En fin de test d'évaluation, les participants ont répondu à un questionnaire. Ce questionnaire nous a permis de mettre en évidence un certain nombre de points positifs et négatifs de chaque algorithme et d'émettre une explication sur les performances relatives de ces algorithmes. Les questions posées sont les suivantes :

Q1. Quel est l'algorithme de dessin le plus efficace pour cette tâche ?

Q2. Quel est l'algorithme de dessin le moins efficace pour cette tâche ?

Q3. Dans quel type de dessin les arêtes en surbrillance sont-elles le plus facile à suivre ?

Q4. Dans quel type de dessin les arêtes en surbrillance sont-elles le plus difficile à suivre ?

Q5. Dans quel type de dessin le voisinage d'un sommet est-il le plus facile à identifier ?

Q6. Dans quel type de dessin le voisinage d'un sommet est-il le plus difficile à identifier ?

D'autre part, les participants ont aussi écrit un commentaire succinct sur chaque algorithme de dessin, nous décrivant ainsi les points positifs et négatifs (s'il en existe) de chacun d'entre eux.

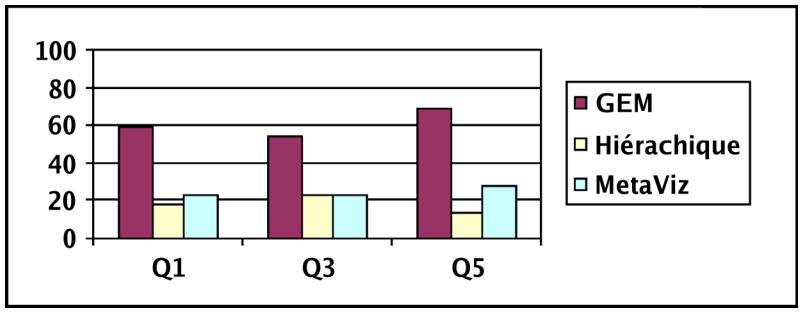

FIG. 7.11: Réponses des participants aux questions Q1, Q3 et Q5 (algorithmes préférés) données en pourcentages.

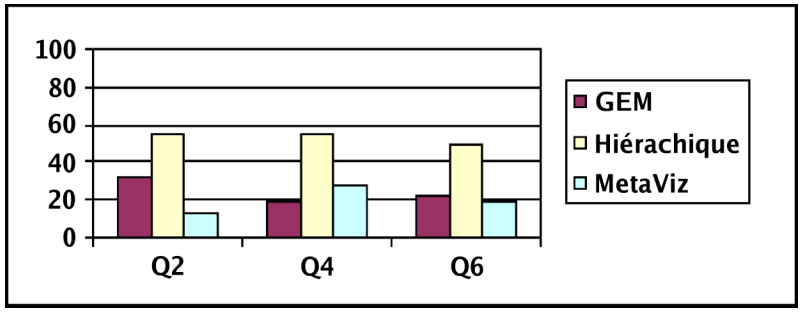

FIG. 7.12: Réponses des participants aux questions Q2, Q4 et Q6 (algorithmes les moins appréciés) données en pourcentages.

Les figures 7.11 et 7.12 montrent les résultats du questionnaire donné aux participants (donnés en pourcentages). Un résumé des commentaires est donné dans le tableau 7.3. Il se dégage des résultats qualitatifs que GEM est l'algorithme préféré par la majorité des participants, suivi de l'algorithme MetaViz puis de l'algorithme hiérarchique.

	Commentaires positifs		Commentaires négatifs	
GEM	17	« arêtes courtes »	9	« recouvrements sommet/arête » « arêtes trop espacées »
Hiérarchique	6	« pas de recouvrement d'arêtes »	21	« arêtes longues » « dessin grand »
MetaViz	5	« arêtes orthogonales »	16	« arêtes longues » « certaines arêtes trop proches »

TAB. 7.3: Nombre et résumé des commentaires positifs et négatifs sur les trois algorithmes de dessin.

Afin de vérifier s'il existe une corrélation entre la préférence d'un algorithme et la performance en temps sur cet algorithme, les préférences (sur les algorithme de dessin) des participants ont été associées à un score de 1 à 3. En utilisant les performances en temps de chaque participant, nous avons effectué une analyse de la corrélation entre préférence et performance. Cette analyse a conduit à affirmer qu'il n'y a aucune corrélation entre la performance d'un participant sur un algorithme et sa préférence. Cela indique que les participants n'ont pas été plus performants (en temps) sur l'algorithme qu'ils ont préféré.

7.5 Discussion

Les résultats quantitatifs ont en partie vérifié l'hypothèse *a priori* que nous avions formulée, puisque l'algorithme GEM permet de résoudre la tâche de recherche de motif plus efficacement que l'algorithme hiérarchique. Cependant, nous avons été surpris par les résultats de l'algorithme MetaViz dont les performances sont aussi bonnes que celles de GEM. Une explication probable de ce succès est d'après nous que le dessin obtenu en appliquant cet algorithme a une apparence claire, structurellement organisée. D'autre part, même si l'algorithme de [64] n'est pas un algorithme de dessin orthogonal, les modifications

apportées pour qu'il prenne en compte les tailles variables des sommets (cf section 6.5) font de l'algorithme MetaViz un algorithme « visuellement orthogonal ». Cette organisation orthogonale ainsi que le fait qu'il n'y ait pas de recouvrement sommet/arête et arête/arête permet une meilleure lisibilité et une bonne visualisation du voisinage. Une autre explication plausible est que les positions des sommets sont relativement bien distribuées dans le plan, avec quelques zones de forte densité d'information.

Quant aux données qualitatives, il apparaît clairement que GEM est l'algorithme jugé comme le plus efficace pour cette tâche et l'algorithme hiérarchique comme le moins bon. MetaViz se positionne quant à lui entre ces deux extrêmes. On peut aussi remarquer une anomalie dans les résultats aux questions sur les algorithmes les moins efficaces. En effet, pour la question Q2, GEM est considéré comme le « second » pire algorithme par 31.8% des participants (contre 13.6% pour MetaViz) bien qu'il soit aussi considéré comme le meilleur par plus de 59% des participants. L'analyse des questionnaires a montré que les participants qui ont classé GEM comme algorithme le moins efficace, ont mis en évidence un certain nombre de problèmes tels que les croisements sommet/arête dans les commentaires libres.

Nous avions anticipé les bons résultats de l'algorithme GEM notamment à cause de la qualité esthétique offerte par les algorithmes par modèle de forces et la proximité dans le dessin des sommets voisins dans le graphe. Cette hypothèse a été validée dans les commentaires libres des participants qui ont commenté favorablement la « petite » taille des arêtes et la bonne répartition des sommets et arêtes dans le dessin.

L'ordre du classement préférentiel des trois algorithmes est donc clairement l'algorithme GEM puis l'algorithme MetaViz et enfin l'algorithme hiérarchique. Ce qui contraste avec les performances en temps des participants puisque les algorithmes GEM et MetaViz produisent des résultats similaires.

Le résultat le plus intéressant de cette étude est que l'algorithme MetaViz offre d'aussi bonnes performances que l'algorithme GEM malgré le fait que les dessins qui en résultent ne semblent pas favorables à une tâche liée à la connectivité. Un aspect important de ces résultats est qu'ils ne peuvent être attribués ni à des connaissances biologiques ni au fait que ce type de représentation puisse être familière à certains participants puisqu'aucun d'entre eux n'est biologiste.

7.6 Conclusion

Lorsque l'on conçoit un outil de visualisation de réseaux métaboliques, le choix de l'algorithme de dessin est très important. En effet, les biologistes ont certaines attentes en particulier en ce qui concerne les conventions de dessin. Les algorithmes classiques de dessin de graphe tels que les algorithmes par modèle de forces ou encore les algorithmes hiérarchiques peuvent se montrer utiles. Notre hypothèse était que les algorithmes par modèle de forces seraient plus efficaces sur une tâche liée à la connectivité (comme la recherche d'occurrences de motifs). Cependant, notre étude expérimentale a montré qu'il n'y

a aucune différence entre un dessin respectant les conventions biologiques et un algorithme par modèle de forces.

Nous pouvons conclure de cette étude que les efforts pour mettre en place des algorithmes respectant les conventions de dessin de données biologiques ne sont pas inutiles. Et cela, non seulement parce que ces algorithmes collent mieux aux attentes des biologistes mais aussi parce qu'ils peuvent être tout aussi efficaces que les algorithmes génériques.

Ces résultats doivent bien sûr être interprétés dans le contexte de cette expérimentation, ses limitations et les données biologiques utilisées. En effet, cette étude a été menée avec trois réseaux de tailles particulières ainsi que trois motifs particuliers. Utiliser plus de réseaux et plus de motifs pourrait permettre de généraliser ces résultats même s'il y aurait toujours certaines limitations liées au protocole expérimental.

Etant donné que nous souhaitions évaluer l'effet des algorithmes de dessin sur l'efficacité d'un utilisateur sur la tâche biologique connue qu'est la recherche d'occurrences de motifs, nous avons volontairement exclu tout biologiste de cette expérimentation. Nous pouvons par conséquent espérer que ces résultats seraient confirmés voire même que les résultats de MetaViz soient améliorés en effectuant une étude similaire sur un échantillon de participants biologistes.

Les données qualitatives de cette étude expérimentale nous ont permis de mettre en évidence un certain nombre de points positifs et négatifs de l'algorithme MetaViz. Intégrer ces commentaires dans l'algorithme de dessin permettra d'améliorer nettement la qualité de la prochaine version de MetaViz. L'une de ces améliorations pourrait consister à augmenter la densité de l'information et diminuer la taille globale du dessin en supprimant les zones « blanches » et ainsi répondre à bon nombre de commentaires négatifs. Nous espérons que ces améliorations de l'algorithme MetaViz permettront d'obtenir de meilleurs résultats expérimentaux.

Chapitre 8
Conclusion et perspectives

Nous avons abordé dans cette thèse trois des différents aspects de la Visualisation d'Information : la décomposition de graphe, le dessin de graphe et en particulier de graphe sur lequel une décomposition a été appliquée, et enfin l'évaluation par un panel d'utilisateurs des choix fait pour la visualisation de réseaux métaboliques.

Dans le chapitre 4, nous avons présenté un algorithme de décomposition en composantes 4-connexes. Cet algorithme de décomposition donne de meilleurs résultats qualitatifs que les autres approches auxquelles il a été comparé. En effet, la décomposition en composante 4-connexes permet d'obtenir simultanément de bonnes valeurs de modularité et de MQ contrairement aux autres algorithmes évalués. D'autre part, la densité moyenne de groupes détectés par la décomposition en composantes 4-connexes est très proche de 1. Bien que la complexité théorique de la décomposition en composantes 4-connexes soit $O(|V| \cdot |E|)$, le temps de calcul moyen de l'algorithme est tout à fait acceptable puisque proche de celui des autres algorithmes comparés.

Dans le chapitre 5, nous avons décrit le travail présenté dans [20] sur la visualisation de graphe partitionné arête-valué. L'algorithme que nous avons proposé permet non seulement de visualiser aisément le partitionnement multi-niveaux du graphe mais aussi de rendre compte des valuations des arêtes. Pour ce faire, nous avons présenté un algorithme descendant par modèle de forces. Afin d'interdire le chevauchement de groupes dans le dessin, notre approche utilise une subdivision du plan en cellules de Voronoï. Enfin, l'adaptation de l'algorithme par modèle de forces GRIP aux graphes arête-valués permet de respecter au mieux les valuations des arêtes.

Nous avons décrit un système de visualisation de graphe décomposé, publié dans [19, 22], dans le chapitre 6. Dans ce chapitre, nous prenons l'exemple de réseaux métaboliques et présentons un système permettant la visualisation de tels réseaux. La difficulté principale dans ce type de visualisation est de prendre en compte la décomposition du réseau en voies métaboliques tout en interdisant la duplication de sommet. Un autre aspect pris en compte dans notre travail est le respect des conventions de dessin des biologistes, notamment l'utilisation d'un dessin circulaire pour représenter les cycles de réactions. Notre approche permet de prendre en compte ces contraintes en effectuant tout d'abord une partition du graphe. D'une part, cette partition permet de trouver les voies métaboliques d'intérêt pour l'expert, d'autre part, elle permet de trouver les structures topologiques dont les représentations doivent respecter les conventions de dessin.

Enfin, dans le chapitre 7, nous avons évalué la qualité de l'algorithme de dessin utilisé dans le chapitre 6 pour représenter le graphe quotient du plus haut niveau de la partition. Nous avons pour cela effectué une évaluation expérimentale avec 22 participants. Lors de cette étude, nous avons comparé l'efficacité de trois algorithmes de dessin : un algorithme par modèle de forces, un algorithme hiérarchique et l'algorithme MetaViz (présenté dans le chapitre 6). Les résultats obtenus montrent que l'algorithme hiérarchique offre une efficacité moindre comparé aux deux autres algorithmes testés. Etant donné que la tâche utilisée lors de cette évaluation est liée à la connectivité, l'hypothèse que nous avions émise était que l'algorithme par modèle de forces donnerait de meilleurs résultats que MetaViz. Cependant, après une étude statistique, il semble qu'il n'existe pas de différence significative entre les résultats obtenus par ces deux approches. Nous avons intentionnellement exclu les biologistes de notre panel de participants afin d'éviter tout biais lié à leur familiarité avec le type de représentation qu'offre l'algorithme MetaViz. Répéter cette évaluation sur un panel de biologistes permettrait donc certainement d'obtenir des résultats meilleurs pour cet algorithme.

8.1 Perspectives

Les travaux et résultats présentés dans cette thèse offrent de nombreuses perspectives, notamment en ce qui concerne la décomposition de graphe et la visualisation de réseaux métaboliques.

8.1.1 Décompositions

Il semble intéressant d'étendre la comparaison d'algorithmes de décomposition. Le premier aspect de cette généralisation consiste à comparer d'autres algorithmes, notamment les méthodes spectrales mais aussi les méthodes locales. Puis, il semble intéressant d'utiliser d'autres mesures de qualité pour évaluer les différentes approches. Et dans un dernier temps, nous voulons mesurer l'efficacité de ces algorithmes sur des classes de graphes particulières afin de classifier ces algorithmes.

La deuxième perspective offerte par cette étude est la mise au point d'un algorithme *hybride* de décomposition. En effet, la plupart des algorithmes tentent de maximiser une mesure de qualité particulière. Utiliser conjointement plusieurs algorithmes pourrait permettre d'obtenir de bons résultats pour plusieurs de ces mesures. Afin de permettre de décomposer de grands graphes (contenant plusieurs dizaines de milliers d'éléments), le processus consisterait à utiliser des algorithmes peu coûteux en temps, puis à affiner ce résultat par des algorithmes plus coûteux mais aussi plus efficaces.

8.1.2 Visualisation de réseaux métaboliques

L'expérimentation menée dans le chapitre 7, a permis de mettre en évidence un certain nombre de problèmes dans la visualisation que nous offrons dans le chapitre 6. L'une des remarques les plus fréquentes est que le dessin réalisé par MetaViz contient de trop

nombreuses « zones vides ». Il semble donc important de réduire l'espace occupé par le dessin et par conséquent la taille et le nombre de ces « zones ».

La deuxième perspective offerte par la visualisation de réseaux métaboliques consiste à utiliser de récents travaux portant sur la visualisation de diagramme d'Euler [120]. Cependant, cet algorithme ne permet pas de respecter les conventions de dessin dictées par les biologistes. Il faudrait donc adapter ces travaux aux attentes des biologistes.

Enfin, la dernière piste de recherche consiste à permettre une comparaison visuelle de réseaux métaboliques. Dans [21], nous présentons un prototype de système de comparaison visuelle de réseaux métaboliques. Dans ce système, le résultat de la comparaison de deux réseaux métaboliques est un réseau contenant les voies présentes dans les deux réseaux comparés. Ce système ne permet pour l'instant de visualiser que l'intersection des réseaux, il semble intéressant d'étendre ces résultats afin de pouvoir visualiser non seulement l'intersection mais aussi les parties propres de chacun des réseaux comparés.

8.1.3 Autre perspective

Lors de cette thèse, nous avons porté un intérêt particulier aux algorithmes de décomposition ainsi qu'aux algorithmes de dessin. Combiner ces algorithmes permettrait de mettre en place des systèmes de visualisation de très grands graphes. Le principe serait d'appliquer des algorithmes de décomposition peu coûteux en temps (et éventuellement d'affiner cette décomposition par des algorithmes plus coûteux). Puis l'utilisation d'algorithmes de dessin adaptés à la topologie et/ou à la taille de chaque groupe permettrait de construire dans des temps très faibles une visualisation de ces graphes.

Chapitre 8
Bibliographie

[1] Sbml viewer. http ://sbw.kgi.edu/layout/.

[2] IEEE InfoVis 2007 Contest : InfoVis goes to the movies. 2007. http ://www.apl.jhu.edu/Misc/Visualization/.

[3] J. Abello, S. G. Kobourov, and R. Yusufov. Visualizing Large Graphs with Compound-Fisheye Views and Treemaps. In *Proc. Graph Drawing 2004 (GD'04)*, pages 431–441, 2004.

[4] J. Abello, F. Van Ham, and N. Krishnan. ASK-GraphView : A Large Graph Visualisation System. *IEEE Transactions on Visualization and Computer Graphics*, 12(5) :669–676, 2006.

[5] A.T. Adai, S.V. Date, S. Wieland, and E.M. Marcotte. Lgl : creating a map of protein function with an algorithm for visualizing very large biological networks. *Journal Mol Biol*, 340(1) :179–190, 2004.

[6] M. Amiel, G. Melançon, and C. Rozenblat. Réseaux multi-niveaux : l'exemple des échanges aériens mondiaux de passagers. *Mappemonde*, 79, 2005. http ://mappemonde.mgm.fr/num7/articles/art05302.html.

[7] D. Archambault, T. Munzner, and D. Auber. Topolayout : Multi-level graph layout by topological features. In *IEEE Information Visualization 2005 Poster Compendium*, pages 3–4, 2005.

[8] D. Archambault, T. Munzner, and D. Auber. Smashing Peacocks Further : Drawing Quasi-Trees from Biconnected components. *IEEE Transactions on Visualization and Computer Graphics (Proc. Vis/InfoVis 2006)*, 12(5) :813–820, 2006.

[9] D. Archambault, T. Munzner, and D. Auber. Grouse : Feature-Based and Steerable Graph Hierarchy Exploration. In Ken Museth, Torsten Möller, and Anders Ynnerman, editors, *Eurographics/ IEEE-VGTC Symposium on Visualization*, pages 67–74, Norrköping, Sweden, 2007. Eurographics Association.

[10] D. Archambault, T. Munzner, and D. Auber. Topolayout : Multi-level graph layout by topological features. *IEEE Transactions on Visualization and Computer Graphics*, 13(2) :305–317, 2007.

[11] D. Archambault, T. Munzner, and D. Auber. GrouseFlocks : Steerable Exploration of Graph Hierarchy Space. *IEEE Transactions on Visualization and Computer Graphics*, à paraître.

[12] D. Auber. Tulip : A huge graph visualisation framework. In P. Mutzel and M. Jünger, editors, *Graph Drawing Softwares*, Mathematics and Visualization, pages 105–126. Springer-Verlag, 2003.

[13] D. Auber, Y. Chiricota, F. Jourdan, and G. Melançon. Multiscale Visualization of Small-World Networks. In S. C. North and T. Munzner, editors, *Proc. of IEEE Information Visualization Symposium*, pages 75–81, Seattle, USA, 2003. IEEE Computer Press.

[14] D. Auber and F. Jourdan. Interactive Refinement of Multi-scale Network Clusterings. In *Proc. of the Ninth Int. Conf. on Information Visualisation (IV'05)*, pages 703–709, Washington, DC, USA, 2005. IEEE Computer Society.

[15] S. Baase and A. V. Gelder. *Computer algorithms : introduction to design and analysis (3rd ed.)*. Addison-Wesley, 2000.

[16] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing : Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.

[17] M. Becker and I. Rojas. A Graph Layout Algorithm for Drawing Metabolic Pathways. *Bioinformatics*, 17 :461–467, 2001.

[18] C. Berge. *Graphes*. Gauthier Villars, 1983.

[19] R. Bourqui, D. Auber, V. Lacroix, and F. Jourdan. Metabolic network visualization using constraint planar graph drawing algorithm. In *Pro. of the 10th Int. Conf. on Information Visualisation (IV'06)*, pages 489–496, Washington, DC, USA, 2006. IEEE Computer Society.

[20] R. Bourqui, D. Auber, and P. Mary. How to Draw Clustered Weighted Graphs using a Multilevel Force-directed Graph Drawing Algorithm. In *Proc. the 11th Int. Conf. on Information Visualisation (IV'07)*, pages 757–764, Washington, DC, USA, 2007. IEEE Computer Society.

[21] R. Bourqui and F. Jourdan. Revealing subnetwork roles using contextual visualization : comparison of metabolic networks. In *Proc. the 12th Int. Conf. on Information Visualisation (IV'08)*, pages 638–643, Washington, DC, USA, 2008. IEEE Computer Society.

[22] R. Bourqui, V. Lacroix, L. Cottret, D. Auber, P. Mary, M.-F. Sagot, and F. Jourdan. Metabolic network visualization eliminating node redundance and preserving metabolic pathways. *BMC Systems Biology*, 1(29), 2007. http ://www.biomedcentral.com/1752-0509/1/29/.

[23] F. Brandenburg, M. Forster, A. Pick, M. Raitner, and F. Schreiber. Biopath : Exploration and visualization of biochemical pathways. In P. Mutzel and M. Jünger, editors, *Graph Drawing Softwares*, Mathematics and Visualization, pages 215–236. Springer-Verlag, 2003.

[24] U. Brandes. A faster algorithm for betweenness centrality. *Journal of Methmatical Sociology*, 25 :163–177, 2001.

[25] U. Brandes, T. Dwyer, and F. Schreiber. Visualizing Related Metabolic Pathways in Two and Half Dimensions. In *Proc. Graph Drawing 2003 (GD'03)*, pages 11–122, 2003.

[26] C. Buchheim, M. Jünger, and S. Leipert. Improving walker's algorithm to run in linear time. In *Proc. Graph Drawing 2002 (GD'02)*, pages 344–353, 2002.

[27] H. Byelas and A. Telea. Visualization of areas of interest in software architecture diagrams. In *Proc. of the 2006 ACM symposium on Software visualization (SoftVis '06)*, pages 105–114. ACM, 2006.

[28] L. Cai and D. G. Corneil. Tree spanners : an overview. In *Proc. of the Twenty-Third Southeastern Conference on Combinatorics, Graph Theory and Computing 1992, Congressus Numerantium*, number 88, pages 65–76, 1992.

[29] L. Cai and D.G. Corneil. Tree Spanners. *SIAM journal on Discrete Mathematics*, 8(3) :359–387, 1995.

[30] R. Caspi, H. Foerster, C. Fulcher, R. Hopkinson, J. Ingraham, P. Kaipa, M. Krummenacker, S. Paley, J. Pick, S.Y. Rhee, C. Tissier, P. Zhang, and P.D. Karp. MetaCyc : a multiorganism database of metabolic pathways and enzymes. *Nucleic Acids Research*, 34 :D511–D516, 2006.

[31] B. Cheswick, H. Burch, and S. Branigan. Mapping and visualizing the internet. In *ATEC '00 : Proc. of the annual conference on USENIX Annual Technical Conference*, pages 1–1, Berkeley, CA, USA, 2000. USENIX Association.

[32] Y. Chiricota, F. Jourdan, and G. Melançon. Software Components Capture using Graph Clustering. In *11th IEEE International Workshop on Program Comprehension*, 2003.

[33] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (Second Edition)*. MIT Press and McGraw-Hill, 2005.

[34] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geomtry : Algorithm and Application (Third Edition)*. Springer-Verlag, 2008.

[35] E. W. Dijkstra. *A short introduction to the art of programming*. Technische Hogeschool Eindhoven, 1971.

[36] U. Dogrusoz, E. Giral, A. Cetintas, A. Civril, and E. Demir. A compound graph layout algorithm for biological pathways. In *Proc. Graph Drawing 2004 (GD'04)*, pages 442–447, 2004.

[37] S. dos Santos and K. Brodlie. Gaining understanding of multivariate and multidimensional data through visualization. *Computer & Graphics*, 28(3) :311–325, 2004.

[38] J.R. Driscoll, H.N. Gabow, R. Shrairman, and R.E. Tarjan. Relaxed heaps : an alternative to Fibonacci heaps with applications to parallel computation. *Communications of the ACM*, 31(11) :1343–1354, 1988.

[39] T. Dwyer and P. Eckersley. Wilmascope - a 3d graph visualization system. In P. Mutzel and M. Jünger, editors, *Graph Drawing Softwares*, Mathematics and Visualization, pages 55–75. Springer-Verlag, 2003.

[40] T. Dwyer, K. Marriott, and P. Stuckey. Fast Node Overlap Removal. In *Proc. Graph Drawing 2005 (GD'05)*, pages 153–164, 2005.

[41] T. Dwyer, K. Marriott, and P. Stuckey. Fast Node Overlap Removal - Correction. In *Proc. Graph Drawing 2006 (GD'06)*, pages 446–447, 2006.

[42] P. Eades. A heuristic for graph drawing. In *Congressus Numerantium*, volume 42, pages 149–160, 1984.

[43] P. Eades and Q.-W. Feng. Multilevel Visualization of Clustered Graphs. In *Proc. Graph Drawing 1996 (GD'96)*, pages 101–112, 1996.

[44] P. Eades, Q.-W. Feng, and X. Lin. Straight-Line Drawing Algorithms for Hierarchical Graphs and Clustered Graphs. In *Proc. Graph Drawing 1996 (GD'96)*, pages 113–128, 1996.

[45] P. Eades, Q.-W. Feng, and H. Nagamochi. Drawing Clustered Graphs on an Orthogonal Grid. *Journal of Graph Algorithms and Applications*, 3(4) :3–29, 1999.

[46] P. Eades and M.L. Huang. Navigating Clustered Graph using Force-Directed Methods. *journal Graph Algorithm and Applications*, 4 :157–181, 2000.

[47] M. Eiglsperger and M. Kaufmann. Fast compaction for orthogonal drawings with vertices of prescribed size. In *Proc. Graph Drawing 2001 (GD'01)*, pages 124–138, 2001.

[48] M. Eiglsperger, M. Siebenhaller, and M. Kaufmann. An Efficient Implementation of Sugiyama's Algorithm for Layered Graph Drawing. In *Proc. Graph Drawing 2004 (GD'04)*, pages 155–166, 2004.

[49] B. S. Everitt and G. Dunn. *Applied Multivariate Data Analysis*. Arnold, 1991.

[50] S. Fortune. A sweepline algorithm for Voronoi diagrams. In *SCG '86 : Proc. of the second annual symposium on Computational geometry*, pages 313–322, 1986.

[51] H. De Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10 :41–51, 1990.

[52] M.L. Fredman and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *journal of the ACM*, 34 :596–615, 1987.

[53] L.C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(1) :35–41, March 1977.

[54] A. Frick, A. Ludwig, and H. Mehldau. A Fast Adaptive Layout Algorithm for Undirected Graphs. In *Proc. Graph Drawing 1994 (GD'94)*, pages 388–403, 1994.

[55] T. M. J. Fruchterman and E. M. Reingold. Graph Drawing by Force-directed Placement. In *Software-Practice and Experience*, volume 21(11), pages 1129–1164. nov 1991.

[56] G. W. Furnas. The FISHEYE view : A new look at structured files. Technical Report #81-11221-9, Murray Hill, New Jersey 07974, U.S.A., dec 1981.

[57] S. D. Gabouje and E. Zimányi. Generic visualization of biochemical networks : A new compound graph layout algorithm. In *Poster Proc. of the 4th International Workshop on Efficient and Experimental Algorithms (WEA 05)*, 2005.

[58] P. Gajer, M. T. Goodrich, and S. G. Kobourov. A Multi-dimensional Approach to Force-Directed Layouts of Large Graphs. *Computational Geometry : Theory and Applications*, 29 :3–18, 2004.

[59] P. Gajer and S. G. Kobourov. GRIP : Graph dRawing with Intelligent Placement. In *Proc. Graph Drawing 2000 (GD'00)*, pages 222–228, 2000.

[60] E. R. Gansner, Y. Koren, and S. C. North. Topological Fisheye Views for Visualizing Large Graphs. *IEEE Transactions on Visualization and Computer Graphics*, 11(4) :457–468, 2005.

[61] E. R. Gansner, E. Koutsofios, S. C. North, and K.-P. Vo. A Technique for Drawing Directed Graphs. *Software Engineering*, 19(3) :214–230, 1993.

[62] M. Granitzer, W. Kienreich, V. Sabol, K. Andrews, and W. Klieber. Evaluating a System for Interactive Exploration of Large, Hierarchically Structured Document Repositories. In *Proc. of IEEE Information Visualization Symposium*, pages 127–134, Washington, DC, USA, 2004. IEEE Computer Society.

[63] S. Grivet, D. Auber, J. Domenger, and G. Melançon. Bubble tree drawing algorithm. In *International Conference on Computer Vision and Graphics*, pages 633–641, 2004.

[64] C. Gutwenger and P. Mutzel. Planar Polyline Drawings with Good Angular Resolution. In *Proc. Graph Drawing 1998 (GD'98)*, volume 1547, pages 167–182, 1998.

[65] S. Hachul and M. Jünger. Drawing Large Graphs with a Potential-Field-Based Multilevel Algorithm. In *Proc. Graph Drawing 2004 (GD'04)*, pages 285–295, 2004.

[66] S. Hachul and M. Jünger. An Experimental Comparison of Fast Algorithms for Drawing General Large Graphs. In *Proc. Graph Drawing 2005 (GD'05)*, pages 235–250, 2005.

[67] D. Harel and Y. Koren. Drawing graphs with non-uniform vertices. In *Proc. Working Conference on Advanced Visual Interfaces (AVI'02)*, pages 157–166. ACM Press, 2002.

[68] D. Harel and Y. Koren. Graph drawing by high dimensional embedding. In *Proc. Graph Drawing 2002 (GD'02)*, pages 207–219, 2002.

[69] D. Harel and Y. Koren. Graph drawing by high dimensional embedding. *Journal of Graph Algorithms and Applications*, 8(2) :195–214, 2004.

[70] I. Herman, M. S. Marshall, and G. Melançon. Graph Visualisation and Navigation in Information Visualisation : A Survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1) :24–43, 2000.

[71] P. R. Hinton. *Statistics Explained : A Guide for Social Science Students*. Routledge, 1995.

[72] D. Holten. Hierachical Edge Bundles : Visualization of Adjacency Relations in Hierarchical Data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5) :805–812, 2006.

[73] J. E. Hopcroft and R. E. Tarjan. Dividing a Graph into Triconnected Components. *SIAM journal on Computing*, 2(3) :135–158, 1973.

[74] M.L. Huang and P. Eades. A Fully Animated Interactive System for Clustering and Navigating Huge Graphs. In *Proc. Graph Drawing 1998 (GD'98)*, pages 374–383, 1998.

[75] P. Jaccard. Distribution de la flore alpine dans la Bassin de Dranses et dans quelques régions voisines. *Bulletin de la Société Vaudoise des Sciences Naturelles*, 37 :241–272, 1901.

[76] T. J. Jankun-Kelly and K.-L. Ma. MoireGraphs : Radial Focus+Context Visualization and Interaction for Graphs with Visual Nodes. In S. C. North and T. Munzner, editors, *Proc. of IEEE Information Visualization Symposium*, pages 59–66, Seattle, USA, 2003. IEEE Computer Press.

[77] G. Joshi-Tope, M. Gillespie, I. Vastrik, P. D'Eustachio, E. Schmidt, B. de Bono, B. Jassal, G. R. Gopinath, G. R. Wu, L. Matthews, S. Lewis, E. Birney, and L. Stein. Reactome : a knowledgebase of biological pathways. *Nucleic Acids Research*, 33 :D428–D432, 2005.

[78] F. Jourdan and G. Melançon. A Tool for Metabolic and Regulatory Pathways Visual Analysis. In *Visualization and Data Analysis, VDA*, pages 46–55. SPIE, jan 2003.

[79] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. In *Information Processing Letters*, volume 31, pages 7–15, apr 1989.

[80] M. Kanehisa and S. Goto. KEGG : Kyoto Encyclopedia of Genes and Genomes. *Nucleic Acids Research*, 28(1) :27–30, 2000.

[81] G. Kant. Drawing planar graphs using the canonical ordering. *Algorithmica*, 16 :4–32, 1996.

[82] D.R. Karger, D. Koller, and S.J. Phillips. Finding the hidden path : time bounds for all-pairs shortest paths. *SIAM journal on Computing*, 22 :1199–1217, 1993.

[83] P. Karp, S. Paley, and P. Romero. The Pathway Tools software. *Bioinformatics*, 18(90001) :S225–S232, 2002.

[84] P. Karp, M. Riley, M. Saier, and I. Paulsen. The ecocyc and metacyc databases. *Nucleic Acids Research*, 28(1) :56–59, 2000.

[85] P.D. Karp, C.A. Ouzounis, C. Moore-Kochlacs, L. Goldovsky, P. Kaipa, D. Ahren, S. Tsoka, N. Darzentas, V. Kunin, and N. Lopez-Bigas. Expansion of the BioCyc collection of pathway/genome databases to 160 genomes. *Nucleic Acids Research*, 19 :6083–6089, 2005.

[86] M. Kaufmann and D. Wagner. *Drawing Graphs : Methods and Models*. Springer, 2001.

[87] I.M. Keseler, J. Collado-Vides, S. Gama-Castro, J. Ingraham, S. Paley, I.T. Paulsen, M. Peralta-Gil, and P.D. Karp. Ecocyc : A comprehensive database resource for Escherichia coli. *Nucleic Acids Research*, 33 :D334–D337, 2005.

[88] P.-Y. Koenig, G. Melancon, C. Bohan, and B. Gautier. Combining DagMaps and Sugiyama Layout for the Navigation of Hierarchical Data. In *Proc. the 11th Int. Conf. on Information Visualisation (IV'07)*, pages 447–452, Washington, DC, USA, 2007. IEEE Computer Society.

[89] Y. Koren, L. Carmel, and D. Harel. ACE : A Fast Multiscale Eigenvectors Computation for Drawing Huge Graphs. In *Proc. IEEE Symposium on Information Visualization*, pages 137–144, 2002.

[90] M. Krummenacker, S. Paley, L. Mueller, T. Yan, and P. D. Karp. Querying and computing with biocyc databases. *Bioinformatics*, 21(16) :3454–3455, 2005.

[91] J. B. Kruskal. On the shortest spanning subtree and the traveling salesman problem. In *Proc. of the American Mathematical Society*, number 7, pages 48–50, 1956.

[92] G. Kumar and M. Garland. Visual Exploration of complex Time-varying Graphs. *IEEE Transactions on Visualization and Computer Graphics*, 12(5) :805–812, 2006.

[93] P. Kuntz, B. Pinaud, and R. Lehn. Minimizing crossings in hierarchical digraphs with a hybridized genetic algorithm. *Journal of Heuristics*, 12(1-2) :23–36, 2006.

[94] V. Lacroix, L. Cottret, P. ThÃ©bault, and M.-F. Sagot. An introduction to metabolic networks and their structural analysis. *IEEE/ACM Trans. Comput. Biol. Bioinform.*, à paraître.

[95] V. Lacroix, C. G. Fernandes, and M.-F. Sagot. Motif search in graphs : application to metabolic networks. *IEEE/ACM Trans. Comput. Biol. Bioinform.*, 3(4) :360–368, 2006.

[96] A. Liebers. Planarizing Graphs - A Survey and Annotated Bibliography. *journal of Graph Algorithms and Applications*, 5(1) :1–74, 2001.

[97] P.C. Lui and R.C. Geldmacher. On the deletion of nonplanar edges of a graph. In *Proc. on the 10th conf. on Comb., Graph Theory, and Comp.*, pages 727–738, 1977.

[98] S. Mancoridis, B. S. Mitchell, Y. Chen, and E. R. Gansner. Bunch : A Clustering Tool for the Recovery and Maintenance of Software System Structures. In *Proc. of the IEEE International Conference on Software Maintenance (ICSM '99)*, pages 50–59, 1999.

[99] S. Mancoridis, B. S. Mitchell, C. Rorres, Y. Chen, and E. R. Gansner. Using Automatic Clustering to Produce High-Level System Organizations of Source Code. In *IEEE Proc. Int. Workshop on Program Understanding (IWPC'98)*, pages 45–53, 1998.

[100] C.C. McGeoch. All-pairs shortest paths and the essential subgraph. *Algorithmica*, 13(5) :426–441, 1995.

[101] F. McSherry. *Spectral methods for the data analysis.* PhD thesis, University of Washingtown, 2004.

[102] G. Michal. On representation of metabolic pathways. *BioSystems*, 47 :1–7, 1998.

[103] M. E. J. Newman. Scientific collaboration networks : II. Shortest paths, weighted networks, and centrality. *Physical Review E*, 64 :016132, 2001.

[104] M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Physical Review E*, 69 :066133, 2004.

[105] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69 :026113, 2004.

[106] M. E. J. Newman, D. J. Watts, and S. H. Strogatz. Random graph models of social networks. *PNAS*, 99(90001) :2566–2572, 2002.

[107] T. Nishizeki and M. S. Rahman. *Planar Graph Drawing*. Lecture Notes Series on Computing. World Scientific Publishing Company, 2004.

[108] Suzanne Pailey and Peter Karp. The Pathway Tools cellular overview diagram and Omics Viewer. *Nucleic Acids Research*, 34(13) :3771–3778, august 2006.

[109] G. Parker, G. Franck, and C. Ware. Visualization of Large Nested Graphs in 3D : Navigation and Interaction. *Journal of Visual Languages & Computing*, 9(3) :299–317, 1998.

[110] D. Reniers and A. Telea. Extreme simplification and rendering of point sets using algebraic multigrid. *Computing and Visualization in Science*, page à paraître, 2008.

[111] S. Rusinkiewicz and M. Levoy. Qsplat : a multiresolution point rendering system for large meshes. In *Proc. of the 27th annual conference on Computer graphics and interactive techniques (SIGGRAPH '00)*, pages 343–352. ACM Press/Addison-Wesley Publishing Co., 2000.

[112] P. Saraiya, C. North, and K. Duca. Visualizing biological pathways : requirements analysis, systems evaluation and research agenda. *Information Visualization*, 4(3) :1–15, 2004.

[113] S. E. Schaeffer. Stochastic local clustering for massive graphs. In T. B. Ho, D. Cheung, and H. Liu, editors, *Proc. of the Ninth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD-05)*, volume 3518 of *Lecture Notes in Computer Science*, pages 354–360, Berlin/Heidelberg, Germany, 2005. Springer-Verlag GmbH.

[114] S. E. Schaeffer. Graph clustering. *Computer Science Review*, 1(1) :27–64, 2007.

[115] F. Schreiber. High Quality Visualization of Biochemical Pathways in BioPath. *In Silico Biology*, 2(2) :59–73, 2002.

[116] F. Schreiber. Comparison of metabolic pathways using constraint graph drawing. In *APBC 03 : Proc. of the First Asia-Pacific bioinformatics conference on Bioinformatics*, pages 105–110. Australian Computer Society, Inc., 2003.

[117] S.B. Seidman. Network Structure and Minimum Degree. *Social Networks*, 5 :269–287, 1983.

[118] P. Shannon, A. Markiel, O. Ozierand, N. Baliga, J. Wang, D. Ramage, N. Amin, B. Schwikowski, and T. Ideker. Cytoscape : A Software Environment for Integrated Models of Biomolecular Interaction Networks. *Genome Research*, 13 :2498–2504, 2003.

[119] B. Shneiderman. The eyes have it : A task by data type taxonomy for information visualization. In Boulder, editor, *Proc. of IEEE Conference on visual languages*, pages 336–343, 1996.

[120] P. Simonetto and D. Auber. Visualise Undrawable Euler Diagrams. In *Proc. the 12th Int. Conf. on Information Visualisation (IV'08)*, pages 594–599, Washington, DC, USA, 2008. IEEE Computer Society.

[121] M. Sirava, T. Schäfer, M. Eiglsperger, M. Kaufmann, O. Kohlbacher, E. Bornberg-Bauer, and H.-P. Lenhof. BioMiner - modeling, analyzing, and visualizing biochemical pathways and networks. *Bioinformatics*, 18 :S219 – S230, 2002.

[122] J. M. Six and I. C. Tollis. A Framework for Circular Drawings of Networks. In *Proc. Graph Drawing 1999 (GD'99)*, pages 107–116, 1999.

[123] K. Sugiyamaa, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2) :109–125, 1981.

[124] R. Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM journal on Computing*, 1(2) :146–160, 1972.

[125] S. T. Teoh and K. Ma. Rings : A technique for visualizing large hierarchies. In *Proc. Graph Drawing 2002 (GD'02)*, pages 268–275, 2002.

[126] S. van Dongen. *Graph Clustering by Flow Simulation*. PhD thesis, Universiteit Utrecht, 2000.

[127] J. J. van Wijk and A. A. Nuij Wim. Smooth and efficient zooming and panning. In S. C. North and T. Munzner, editors, *Proc. of IEEE Information Visualization Symposium*, pages 75–81, Seattle, USA, 2003. IEEE Computer Press.

[128] G.F. Voronoi. Nouveles applications des paramètres continus à la théorie de formas quadratiques. *J Reine Angew Math*, 134 :198–287, 1908.

[129] X. Wang and I. Miyamoto. Generating Customized Layouts. In *Proc. Graph Drawing 1995 (GD'95)*, pages 504–515, 1996.

[130] C. Ware. *Information Visualization : Perception for Design*. Morgan Kaufmann Publishers, 2000.

[131] K. Wegner and U. Kummer. A new dynamical layout algorithm for complex biochemical reaction networks. *BMC Bioinformatics*, 6(212), 2005.

[132] D. J. Welsh and M. B Powell. An upper bound to the chromaticnumber of a graph and its application to timetabling problems. *The Computer journal*, 10 :85–86, 1967.

[133] R. Wiese, M. Eiglsperger, and M. Kaufmann. yFiles : Visualization and automatic layout of graphs. In P. Mutzel and M. Jünger, editors, *Graph Drawing Softwares*, Mathematics and Visualization, pages 173–192. Springer-Verlag, 2003.

[134] U. Zwick. Exact and approximate distances in graphs - A Survey. In *Algorithms - ESA'01, 9th Ann European Symposium*, pages 33–48. Springer, Berlin, 2001.

Décomposition et Visualisation de graphes : Applications aux Données Biologiques

Résumé : La quantité d'informations stockée dans les bases de données est en constante augmentation rendant ainsi nécessaire la mise au point de systèmes d'analyse et de visualisation. Nous nous intéressons dans cette thèse aux données relationnelles et plus particulièrement aux données biologiques. Cette thèse s'oriente autour de trois axes principaux : tout d'abord, la décomposition de graphes en groupes d'éléments "similaires" afin de détecter d'éventuelles structures de communauté ; le deuxième aspect consiste à mettre en évidence ces structures dans un système de visualisation, et dans un dernier temps, nous nous intéressons à l'utilisabilité de l'un de ces systèmes de visualisation via une évaluation expérimentale.

Les travaux de cette thèse ont été appliqués sur des données réelles provenant de deux domaines de la biologie : les réseaux métaboliques et les réseaux d'interactions gènes-protéines.

Mots-clef : Visualisation de graphe, décomposition de graphe, évaluation, bioinformatique
Discipline : Informatique

Graph Clustering and Visualization : Application to Biological Data

Abstract : The amount of information stored in databases is constantly increasing making necessary to develop systems for analysis and visualization. In this thesis, we are interested in relational data and in particular, in biological data. This thesis focuses on three main axes : firstly, the decomposition of graph into clusters of "similar" elements in order to detect the community structures ; the second aspect is to highlight these structures in a visualization system ; and thirdly, we are interested in the usability of one of these visualization systems through an experimental evaluation.

The work presented in this thesis was applied on real data from two fields of biology : the metabolic networks and the gene-protein interaction networks.

Keywords : Graph visualization, graph decomposition, evaluation, bioinformatic
Field : Computer Science