UNIVERSITE BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE RENNES 1

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE RENNES 1
COMUE UNIVERSITÉ BRETAGNE LOIRE

ÉCOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : INFO

Par

Lucas BOURNEUF

A search space of graph motifs for graph compression : From Powergraphs to triplet concepts

Thèse présentée et soutenue à Inria Bretagne, Rennes, le 13 Décembre 2019
Unité de recherche : IRISA
Thèse N° :

Rapporteurs avant soutenance :

Marianne Huchard	Professeur en informatique à l'Université de Montpellier
Michael Schroeder	Professeur en bioinformatique à l'Université technique de Dresde

Composition du Jury :

Présidente :	Marianne Huchard	Professeur en informatique à l'Université de Montpellier
Examinateurs :	Karell Bertet	Maître de conférence à l'Université de La Rochelle
	Sébastien Ferré	Maître de conférence à l'Université de Rennes 1
	Bernhard Ganter	Professeur émérite en algèbre des structures théoriques à l'Université technique de Dresde
Rapporteurs :	Marianne Huchard	Professeur en informatique à l'Université de Montpellier
	Michael Schroeder	Professeur en bioinformatique à l'Université technique de Dresde
Dir. de thèse :	Jacques Nicolas	Directeur de Recherche à Inria Rennes - Bretagne Atlantique

Acknowledgements

I would like to thanks all the members of the jury for their participation, and for taking interest in my work. I am honored to have such great experts in my thesis jury, and hope it was for you as enjoyable as it was for me.

Special thanks to Sébastien Ferré, for following my work from year one, and to Bernhard Ganter for being here today, and at almost all conferences I attended, with questions and remarks.

I also deeply thanks Jacques, for his support, his encouragements, his (seemingly unlimited) patience, and for teaching me, among a lot of other things, what research is. I've come a long way from the ground, and now, after a 3 years liftoff, it seems that I'm reaching orbit.

Finally, thanks to those who have reread my manuscript and helped me improve my defence, including Alice, Anne, Arnaud, Catherine, Claire, Clara, Denis, François, Hugo, Lolita, Marie, Mikail, Nathalie, Nicolas, Rumen, Stéphanie, Téo, Wesley, and, my most dedicated reader, Jacques.

And, now, let me use my native language for a less formal acknowledgment.

INTRODUCTION

Qui aurait crû qu'un jour, j'écrirais, dans un contexte sérieux, quelque chose comme «*shifting the time/memory tradeoff*» ou «*the computation of their normal form is a simple post-process*» ? Il y a des moments dans cette thèse où j'ai l'impression d'être dans un film. Mais en fait, non, je suis sur un ordinateur à écrire du latex. Quoiqu'il en soit, l'occasion est trop belle pour ne pas s'épancher dans ces remerciements, que je suis sûr vous trouverez importants. D'abord parce que ce sont les seules pages de cette thèse dans lesquelles on peut trouver les mots «tintamarre» et «bigarrées», mais aussi parce que ma thèse et moi n'aurions jamais été possibles, ou aurions été très différents, sans la participation de très nombreuses personnes que je m'emplois, avec tout le sérieux qui me caractérise et pour le malheur du cinquième d'arbre qui servira à l'impression des 8 exemplaires finaux, à lister ici. Mais, évidemment, inéluctablement, dans la folie des derniers instants, j'ai immanquablement oublié des gens. Ils peuvent m'en vouloir, ou alors je leur paye un restau. À eux de voir, je suis dispo.

COMMENT LIRE CES REMERCIEMENTS

Le mot «merci», pour en éviter l'usure et rappeler que je déteste voyager, est traduit dans de nombreuses langues que je ne parle pas, et associé aléatoirement aux personnes ; à l'exception de quelques cas impossible à détecter, à moins que ces personnes, physiques, morale ou décédée selon les cas, ne s'organisent pour découvrir ces liens abscon, évidents ou capillotracté, selon les cas également. Lorsque nécessaire, les premières lettres *des* noms de famille sont inclus afin d'éviter les collisions malheureuses ; et cela est nécessaire car malgré mon insigne désintérêt pour les cérémonies religieuses, je remercie ici un nombre tout à fait ébaubissant de Baptiste et de Baptiste B.

Enfin, bien que les noms, lorsque listés, le sont dans l'ordre alphabétique, j'ai conservé pour les paragraphes — relèguant à l'usage minimal l'ordre implicite standard probablement approximable par le rapport $\frac{temps\ passé\ avec}{distance\ psychologique}$ — une organisation par groupes thématiques en quatre points dont les limites sybillines, déjà floues, sont inexorablement brouillées par l'inextricable superposition spatio-temporelle des cercles sociaux. Oui, c'est le bordel.

REMERCIEMENTS

Merci Jacques, ∀.
Notamment pour m'avoir, avec une patience et une bienveillance qui m'étonne encore, encouragé, rassuré, poussé, encadré, recadré, tout en m'ayant laissé aborder ma thèse de manière très personnelle (c'est-à-dire dans tous les sens, sauf ceux décidés en réunion), et aussi pour la quantité résolument *gargantuesque* de corrections sur ma prose, qui avant cela rappelait un dialecte lisp à l'anglais claudicant, à l'expression trébuchante, et avec un problème à la fois manifeste et toujours d'actualité sur l'usage du *s*.
Je dois confesser que, la première fois que tu m'a montré de l'ASP, j'ai absolument rien bitté, mais ça avait l'air diablement rigolo. J'ai donc accepté le stage, et presque 5 ans après, j'ai écris une thèse. Merci pour ça aussi. J'ai pas eu l'air motivé (ou éveillé) tous les jours, ni d'avoir envie de continuer après, mais j'ai appris énormément, et suis très heureux de l'avoir attaquée, et terminée ! Ça, c'est de l'*outpeutt* de qualité, j'en suis très fier et j'espère que toi aussi.

Tesekkür ederim, Mikail, pour nos longues discussions, notre politisation, les encouragements. Vivement le prochain «code Jean-Pierre», ou, car je suis sûr que tu arrivera à quitter tes mauvaises habitudes, «Jean-Jacques».
Kia ora, Maëliss, pour avoir supporté longtemps et bénévolement un nombre de thésards supérieur à la dose maximale recommandée. Vivement nos premiers sureaux et châtaigners. Bravo à vous, les Marchand, et *Blagodaram* pour nos projets terminés, en cours, et celui à venir.

Mauruuru Alice, Baptiste et Thomas, pour m'avoir appris la jongle, le monocycle, la cuisine, le subjonctif, le jeu de plateau, les relations interpersonnelles... et surtout pour m'avoir aidé à prendre du recul et à dédramatiser — bien que j'y attache encore aujourd'hui une émotion particulière — mon rôle de thésard, et l'importance d'avoir une cuillère différente pour chaque pot de confiture.

Milesker Charlotte. Pour, entre autres choses, m'avoir fait connaître cuba, libre. Vivement le prochain match de mouidditch.

Tack Mathieu, pour m'avoir littéralement plongé dans l'informatique en installant linux sur mon PC, et en me présentant vim comme la seule interface dont

j'aurais jamais besoin. Ce soir là je suis repartis, perplexe et interdit, en me demandant si j'avais pas fait une connerie. Mais j'ai réussi. Et t'as raison, c'est l'éclate totale. Mais j'ai jamais réussi à reproduire le bug qui générait 24 Go de logs quand on lançait une interface graphique sur ma debian stable.

Yekeniele Mystic Glue et tous les autres amis de licence, que je n'ai plus vu depuis un certains temps, incluant Colas, Élina, Erwan, Jarog, Laurette et Samuel, mais avec qui j'en ai fait, des trucs que c'est pas marqué dans le manuel «être adulte et responsable».

Grandmercé Pierre, pour m'avoir vacciné contre les auto tamponneuses, fait découvrir la puissance de la méthode La RACHE avec le projet d'ADT, et surtout pour avoir discuté avec moi d'un nombre incalculable d'information révoltantes, fraîches et intéressantes, mais dont tout les autres se fichaient. Mis à part un ulcère, ils ne savent pas ce qu'ils ont loupé.

Köszönöm, Marie, pour la bonne humeur, le défoulement à la hache, les histoires toujours étonnantes de déboires divers dans des milieux insoupçonnés, comme par exemple les interfaces graphiques en matlab de la RATP.

Bayarlalaa David, pour le sirop, le nougat, le naheulband, et les blagues en carton. Vivement qu'à nouveau nous terraformions Mars, et fassions la chenille en armure lourde dans les couloirs sombres d'un vaisseau spatial, en carton également.

Ki'e do Sacha, les projets à la fois géniaux et foireux, pour nos longues (mais pas encore assez) discussions, et ton sens aiguë et communicatif de l'émerveillement. J'ai appris grâce à toi un nombre incalculable de choses. Je n'ai avec toi jamais été simple ni sensible, mais sache que c'est toi qui, en m'emmenant sur ton nuage, m'empêche de dormir sur mes lauriers. Et je ne te remercierais jamais assez pour m'avoir emmené dans un chalet suisse pour un weekend rando-fondue-LHC, mon plus beau et féérique souvenir de voyage à l'étranger.

Ngiyabonga kakhulu au groupe des allumés du master : Charles, David, Jérôme, Pierre, Pierre, Mattieu, Meziane, Sacha, Yoan et les autres, que j'ai oubliés, parce que non contents d'être allumés, on était nombreux. Ce furent deux prodigieuses années parfaitement inexplicables.

Stuutiyi, Baptiste B-B., même si t'es loin et qu'on ne se voit que deux fois l'an pour faire exploser des fusées ou jouer au foot à 4 roues, ce fût chaque fois excellent (comme disent les Sri Lankais). Maintenant que t'as un téléscope, on devrait se voir... plus facilement.

Dziakuju, Baptiste B. On est passé de tirer sur des bandits pour libérer Dillon, à manger de la chèvre, pagayer et couper du bois dans la montagne, en passant par un festival qui s'est terminé à $80 km.h^{-1}$ dans un fossé. Autrement dit, on a grandit ensembles, et ça fait beaucoup de souvenirs.

Arigato Gabriel, d'abord pour m'avoir fait découvrir le jeu de rôle à un moment où j'en avais grand besoin. Ensuite, pour m'avoir emmené, avec Dominique — que je re-*hvala* tout particulièrement aussi pour cela et pour le reste — et les autres, aux meilleures vacances de ma vie dans un lieu isolé, froid et pluvieux, et montré ce qu'était le lâcher prise complet, illimité et créatif. Gzor reviendra.

N'fa n'fa Symbiose, pour l'environnement calme, sain et bienveillant. Plus encore, la possibilité de bosser avec des gens à la fois de bonne compagnie et de domaines très différents, ou encore les rendez-vous réguliers pour apprendre des trucs qui n'ont rien à voir avec ce qu'on fait, mais qui participent à créer un environnement riche et intelligent.

Goeiedag Olivier, pour m'avoir encouragé à enseigner (j'adore ça !), et surtout pour m'avoir introduit auprès de celui qui deviendrait mon maître de stage, et très rapidement, mon directeur de thèse.

Spas aux biologistes, comme on ne les appelle jamais mais que là c'est pratique, notamment Denis, Fabrice, Nathalie et Stéphanie, sans qui ma thèse serait restée bien éloignée de toute considération du réel, et avec en sus beaucoup moins d'images colorées.

Mercés hèra Clémence, pour la bonne humeur constante, les idées en pagailles, et les mots en paires dont on ne sait plus quoi faire. Promis on publiera predator ; avec un nom comme ça on ne peut décemment le laisser en liberté. Joie et futur.

Kiitos l'équipe Midi les Doctorants, composée notamment de Anne, Arnaud, Grégoire, Lolita, Maël, Marine, Méline, Nicolas, Hugo et Wesley, pour ces pauses repas musicales et éclectiques nous ayant permis de découvrir les innovations d'un futur dingue et fantasque, la surdité lucalienne, les atomes vus de profil (*sic*),

l'étude de l'étalage du fromage sur girafe de 7.3m (à 2119.86 euros les matières premières, c'est donné), et bien d'autres aspects de ce qui constitue la perception du monde d'un doctorant au ventre garnis et au cerveau inextinguible.

Matondi Méline, Mikail, Wesley, Chloé, Pauline et Cédric pour l'organisation de Science en Cour[t]s, où j'ai eu l'inoubliable occasion de faire deux films et deux one man shows. C'est clairement l'expérience la plus étrange et exaltante que j'ai pu vivre. Au passage, je re-*shukriyaa* tout spécialement Hugo pour la super réussite de la fable du doctorant.

Gratias ago, Jérémy, pour avoir été le meilleur des co-bureaux. Je ne compte pas le nombre de trucs et de bidules que j'ai appris en biologie, notamment en matière de papillons, de cercles mimétiques et de reconstruction de l'histoire géologico-biologique. Et je pense que nous pouvons être particulièrement fiers des 12 cuillères, majestueuses et rutilantes, qui ont longtemps trôné en notre royaume pourtant déchu.

Tashakor Marie pour la gestion administrative, exécutée dans la bonne humeur comme si ça n'était rien, alors que c'est pour moi une massive épine dans un pied douillet, le lourd parpaing de la bureaucratie sur la tartelette aux fraises de mon ingénuité.

Misaotra Sébastien, pour avoir incarné un autre chemin, très différent du miens, qui pourtant arrivait au même endroit, donnant lieu à de nombreuses conversations des plus fascinantes. Je sais pas si on se reverra, puisque visiblement ils se re-séparent désormais.

Mèsi Camille, pour toutes les idées qu'un jour, sans doute, nous mettrons en œuvre. Notamment le site de rencontre généraliste décentralisé. C'est le futur. Avec ça, se formerons les couples et s'organiserons les révolutions.

Spacibo à l'Inria/IRISA/Université, pour m'avoir montré ce que j'avais toujours eu du mal à accepter : qu'aujourd'hui l'informatique, au-delà d'être un objet de recherche, est avant tout un moyen de reproduction des phénomènes d'asservissement, et non comme je le croyais un moyen de libération.

Paldies aux Bourneuf et aux Beaujean pour m'avoir supporté, géré, parlé, encouragé. Si seulement j'avais passé avec eux autant de temps qu'à écrire cette thèse !

Marci Papa, pour les discussions autour de l'info, de l'enseignement, le reste, le rappel à la réalité quand l'esprit part un peu trop loin. Et pour m'avoir supporté aussi, plus encore et plus longtemps que les autres. Mais maintenant, tu pourra dire que t'as élevé un docteur. Et il est très fier de son père.

Grazie Nina, pour la bonne humeur, ton aide récurrente, pour continuer la tradition des soirées crêpes, et surtout pour trouver terribles les choses simples, et simples les choses terribles. Pour avoir géré le pot aussi. T'imagine pas à quel point ça me stresse ce genre de chose ; plus que la soutenance elle-même, certainement.

Danke Schön Susie, pour avoir cassé les gonades de tout le monde en faisant exactement l'inverse de ce qu'on attendait de toi, et en ayant réussi avec une exécution parfaite un plan sur lequel personne n'aurait parié une cacahuète. Tu es un peu mon modèle de vie : quand je serais grand, je serais comme toi.

Nouari Maman. La dernière chose qu'on ait faite ensemble, d'après mes souvenirs, c'est une inspection du travail de chasseurs de mammouths. Ça date, les mammouths.

Mercé aux tontons, tatas, aux neveux, nièces, et aux grand-parents, notamment mais pas seulement Agnès, Sylvie, Mourad, François, Hervé, Mamie Jacqueline, Mamie Huguette, Didier, et tous ceux que je ne nomme pas ici, et qui ont eu à un moment ou à un autre la *chance* d'avoir à me gérer.

Merkzi aux Adonon, notamment Joris, Ruben et Sophie, parce qu'être en désaccord ne dois pas être motif de dissensions, mais de réflexions et de discussions ; le second me pose encore problème sur les sujets les plus brûlants.

Dziękuję Carl Sagan, pour m'avoir inspiré à comprendre le monde autour de moi, et pour m'avoir introduit à ce qu'aujourd'hui je comprend comme être une philosophie humaniste.

Et pour finir, *Ubuntu*, notamment avec les très, très nombreux humains qui m'ont permis d'avoir un ordinateur au service de mes doigts, avec une interface graphique des années 70 et la technologie des 20 prochaines années.

AND NOW, FOR SOMETHING TOTALLY DIFFERENT

L'anglais de lucas est imbitable ? L'exposé est terriblement long ? Vous avez déjà lu toutes les citations en début de chapitre ? Impossible de dormir sans qu'un membre du jury ne se retourne en fronçant les sourcils ?

J'ai la solution à vos problèmes !

`pip install biseau-gui`, ou allez sur le bug tracker de Biseau : il y a une tonne de boulot à faire, sur le core [1] comme sur la GUI [2]. Pour une introduction au logiciel, allez voir le chapitre 5, ou chargez le script d'exemple dans la GUI.

Ou, si la théorie des graphes et le développement logiciel vous excitent plutôt moyen-bof-pâté, je vous propose un sudoku, ainsi que son *encoding* en ASP :

2	1						7	6
9					2	4		
4					7			
		4		3			1	
			8		9			
	8			4		6		
			5					8
		5	9					4
1	6						9	2

```
s(1,1,2).  s(1,2,1).
s(1,8,7).  s(1,9,6).
s(2,1,9).  s(2,6,2).  s(2,7,4).
s(3,1,4).  s(3,6,7).
s(4,3,4).  s(4,5,3).  s(4,8,1).
s(5,4,8).  s(5,6,9).
s(6,2,8).  s(6,5,4).  s(6,7,6).
s(7,4,5).  s(7,9,8).
s(8,3,5).  s(8,4,9).  s(8,9,4).
s(9,1,1).  s(9,2,6).
s(9,8,9).  s(9,9,2).
```

Travail à faire : trouver la solution au sudoku. Implémenter le code ASP qui trouve la solution. Du joueur, du programmeur, ou du solveur, qui est allé le plus vite ?

Le sudoku lui-même est censé être complexe à résoudre pour un humain (Merci à Maël, Grégoire et Jacques pour l'avoir testé). Dans l'encoding ASP, $s(X, Y, V)$ indique que la case au coordonnées (X, Y) se trouve la valeur V. Il est tout à fait possible de produire un programme qui, à partir de ces atomes, produit ceux qui manquent.

1. `https://gitlab.inria.fr/lbourneu/biseau/issues/`
2. `https://gitlab.inria.fr/lbourneu/biseau-gui/issues/`

Table of Contents

Résumé de la thèse — French summary of the thesis **3**
Publications 18

1 Graph compression for visualization and Answer Set Programming **19**
1.1 Introduction 19
1.2 Power Graph Analysis 27
1.3 Formal Concept Analysis (FCA) 30
1.4 Answer Set Programming 34
1.5 An introduction to ASP for people working with FCA 37
1.6 Thesis contributions and plan 53

2 PowerGraph compression with formal concepts **55**
2.1 Graph contexts to encode undirected graphs 56
2.2 Graph motifs as formal concepts 58
2.3 Compression with formal concepts 61
2.4 Search spaces of Power Graph Analysis 65
2.5 Examples of powergraph compression 69
2.6 Power Graph Analysis and *PowerGrASP* heuristics 73
2.7 Optimizations using graph and motifs properties 78
2.8 Limits of greedy approaches: the cycle motif 83
2.9 Conclusion 84

3 Triplet concepts: an extension of FCA looking for overlapping bicliques **89**
3.1 Intuition and examples 90
3.2 Triplet concepts from a graph perspective 95
3.3 Triplet maximality 100

3.4 Triplet canonical form . . . 101
3.5 Triplet concept definition . . . 105
3.6 Triplet admissibility . . . 106
3.7 Triplet concept ordering . . . 109
3.8 Triplets as a combination of formal concepts . . . 113
3.9 Triplet concept enumeration . . . 115
3.10 Maximal Clique Enumeration with triplet concepts . . . 124
3.11 Conclusion on triplet concepts . . . 130

4 Applications of Power Graph Analysis 133
4.1 Introduction to the data . . . 133
4.2 Application to a transcriptomic study of the pea aphid, *Acyrthosiphon pisum* . . . 136
4.3 Application to human extra-cellular matrix . . . 138
4.4 Benchmarks of the *PowerGrASP* optimizations . . . 140
4.5 Conclusion on the applications . . . 141

5 Biseau: an ASP environment for high-level specification and visualization in graph theory 153
5.1 Software contribution . . . 154
5.2 Biseau, a flexible environment for quick prototyping of concept structure search . . . 155
5.3 Graph drawing With Biseau . . . 158
5.4 Build and Visualize Concept Lattices With Biseau . . . 160
5.5 Pulling Constraints On The Model . . . 163
5.6 Conclusion . . . 167

6 Conclusion and perspectives 169
6.1 Conclusion . . . 169
6.2 Perspectives . . . 170

Bibliography 183

Résumé de la thèse

L'homme de science le sait bien, lui, que seule la science a pu, au fil des siècles, lui apporter l'horloge pointeuse et le parcmètre automatique sans lesquels il n'est pas de bonheur terrestre possible.

Pierre Desproges
Vivons heureux en attendant la mort
1983

0.1 Introduction

Les graphes sont une des structures mathématiques les plus utilisées pour la représentation de problèmes théoriques, la représentation de connaissances, l'analyse de réseaux, la théorie des jeux ou la recherche opérationnelle. Ils sont aussi utilisés pour représenter des données, par exemple les interactions de composés chimiques, les partitions spatiales ou les diagrammes de Feynman. Si les graphes ont l'avantage de pouvoir être représentés graphiquement, déterminer une représentation lisible est néanmoins une tâche complexe dès lors que le graphe comporte plus d'une dizaine de nœuds. Les graphes en bioinformatique en comportent généralement plusieurs milliers, parfois plusieurs millions. L'effet «boule de poil», où un graphe est si grand que sa représentation graphique est une tâche noire informe sur l'écran, peut être évité par plusieurs méthodes, incluant l'agencement intelligent des nœuds et des arcs, ou le résumé automatique. Une autre approche à ce problème de lisibilité est la compression de graphe, consistant en la réduction d'un graphe en des agglomérats de nœuds et d'arcs, afin de diminuer le nombre d'éléments dans la représentation graphique, et donc augmenter sa lisibilité. Dans le cadre de cette thèse, nous nous intéressons à une méthode de compression de graphe sans perte (le graphe compressé permet de retrouver le graphe d'entrée), hiérarchique (les agglomérats, ou clusters, sont imbriqués), et créant des clusters

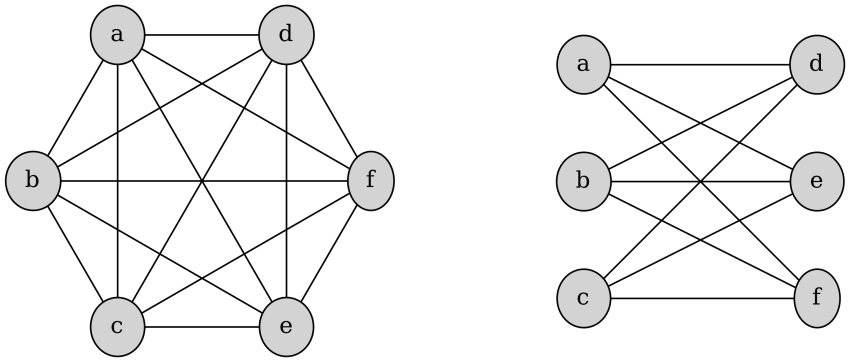


FIGURE 1 – Deux exemples de graphes, qui sont également des motifs généraux. *Gauche :* Une clique de 6 nœuds, couvrant 15 arcs. *Droite :* Une biclique de 3 × 3 nœuds, couvrant 9 arcs.

représentant uniquement des motifs génériques (les cliques et les bicliques, cf Figure 1).

Cette méthode, l'Analyse PowerGraph, nous la formaliserons en utilisant l'Analyse de Concepts Formels (ACF), un champs des mathématiques que nous présenterons rapidement en section 0.1.2. Nous implémenterons notre approche avec la Programmation par Ensembles Réponses (ASP), une forme de programmation purement déclarative conçue pour la résolution de problèmes combinatoires, et présenté en section 0.1.3.

0.1.1 Analyse PowerGraph

D'abord développée dans le domaine de la bioinformatique par Royer *et al.*, pour une application sur divers graphes d'interaction de molécules biologiques [94], l'Analyse PowerGraph est une méthode de clustering et de visualisation qui consiste en la détermination d'un powergraphe à partir d'un graphe d'entrée.

Un powergraphe est un graphe dont les nœuds (powernœuds) sont des agrégats de nœuds, reliés entre eux des arcs (powerarcs), eux-même des agrégats d'arcs. Ce type de graphe a été spécifiquement conçu pour représenter des cliques et des bicliques dans le graphe. Un exemple de powergraphe est donné en Figure 2.

Powergraphes

Soit $G = (V, E)$ un graphe. Un powergraphe est défini comme un graphe $PG = (PN, PE)$ où les nœuds PN sont des sous-ensembles de V et les arcs PE sont des sous-ensembles de E. De plus, un powergraphe doit respecter les conditions suivantes :

condition de sous-graphe Toute paire de powernœuds connectés par un powerarc représente une biclique dans G. Cas particulier : une clique de G est représentée par un powernœud avec un powerarc bouclant sur lui-même.

condition de hiérarchie des powernœuds Deux powernœuds sont soit disjoints, soit inclus l'un dans l'autre. Autrement dit, ils forment une hiérarchie.

condition de décomposition des powerarcs les powerarcs forment une partition de l'ensemble d'arcs E.

Le problème théorique est de trouver un powergraph avec un minimum de powerarcs. La solution optimale n'est pas nécessairement unique, et le problème a été démontré comme appartenant à la classe de complexité NP-complet [37]. Entre autres résultats, nous montrerons que la taille de l'espace de recherche grandit factoriellement avec la taille du graphe d'entrée, et que les solutions optimales ne sont pas toujours atteignables avec les heuristiques de compression existantes.

0.1.2 Analyse de Concepts Formels

L'Analyse de Concepts Formels (ACF) est un cadre d'analyse de données développé par R. Wille et B. Ganter dans les années 70. Il s'agit aujourd'hui d'un domaine aux fondements solides [45], dont les principaux objets sont les contextes formels (relation binaire entre deux ensembles d'éléments), les concepts formels (décrivant des relations maximales entre deux sous-ensembles d'un contexte formel) et le treillis de concept (graphe où les concepts formels sont organisés selon un ordre partiel et des connections de Galois).

l'ACF a été utilisée pour la formalisation d'un large panel de domaines, notamment la découverte de connaissances, la classification, et la théorie des graphes [101, 25]. La dualité graphe/contexte formel a été explorée dans de nombreux travaux. Concernant les motifs de graphes, on trouve par exemple une correspondance un-

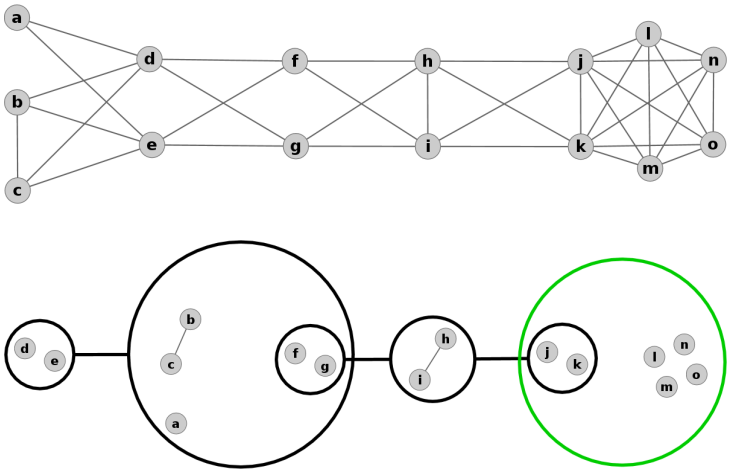


FIGURE 2 – *En haut*, un graphe de 15 nœuds étiquetés de a à o, et de 35 arcs. Un placement adapté des nœuds permet de comprendre la structure du graphe, ce qui serait plus difficile pour de plus gros graphes.
En bas, on trouve sa version compressée, le powergraphe, telle que produit par l'Analyse PowerGraph. Les powerarcs sont montrés comme des lignes noires et épaisses qui relient des powernœuds, les cercles noirs et épais. Les powernœuds dessinés en vert représente des cliques. Certains arcs, (h, i) par exemple, n'ont pas pu être compressés. Le concept $(\{h, i\}, \{f, g, j, k\})$, bien qu'il soit une biclique maximale, ne peut pas être associé à un powerarc particulier sans casser la condition de hiérarchie des powernœuds. À la place, deux powerarcs sont générés, correspondant aux bicliques $(\{h, i\}, \{f, g\})$ et $(\{h, i\}, \{j, k\})$. De la même manière, le sous-graphe des nœuds $\{d, e, f, g, h, i, j, k\}$ ne peut pas être couvert par les powerarcs, $(\{f, g\}, \{d, e, h, i\})$ et $(\{h, i\}, \{f, g, j, k\})$ car cela briserait la condition de décomposition des powerarcs.

à-un entre les concepts formels dérivés du contexte formel décrivant un graphe et des bicliques maximales qui le composent [73, 53].

Cette thèse se propose d'exploiter le lien entre théorie des graphes et l'ACF pour la formalisation d'un problème de théorie des graphes et de classification.

0.1.3 Programmation par Ensembles Réponses

La Programmation par Ensembles Réponses, ou *Answer Set Programming* (ASP), est une forme de programmation purement déclarative et logique conçue pour la résolution de problèmes combinatoires [76]. Il est utilisé en représentation des connaissances, résolution de problèmes, raisonnement automatique ou encore en recherche opérationnelle et en optimisation.

Le processus de calcul d'ASP implique 2 étapes, le *grounding* et le *solving*. À partir d'un ensemble de règles écrites en ASP, le *grounder* génère un programme propositionnel. Le solveur produit, à partir de ces propositions, des modèles minimaux stables (ensembles-réponses). Évidemment, pour un problème spécifique, un programme dédié sera généralement plus efficace que son équivalent, plus compact, en ASP. Cependant, ASP est utile pour la conception de prototypes : c'est une alternative aux langages procéduraux standard qui permet de produire rapidement des implémentations raisonnablement efficaces de résolution de problèmes complexes.

ASP a été utilisé en ACF pour implémenter des langage de requête expressifs sur des contextes formels [58, 95]. Il a aussi été utilisé pour la génération procédurale [97, 103], et dans plusieurs tâches en intelligence artificielle, notamment la robotique et la planification [40].

Notre introduction à ASP (section 1.4 dans la thèse) propose quant à elle une explication du langage ainsi qu'une implémentation en ASP de plusieurs tâche typiques de l'ACF, notamment l'énumération des concepts formels via la reproduction d'un algorithme de référence et son encodage par recherche de point fixe.

0.1.4 Contributions de cette thèse

Tout d'abord, nous formulons et étudions l'Analyse PowerGraph à la lumière de l'ACF. Ceci nous permet d'en proposer une généralisation à l'aide d'un unique motif plus général qui regroupe en un seul espace de recherche les cliques et les bicliques. Cette extension des concepts formels, nommée concepts triplets, est détaillée sur de nombreux aspects, y compris leur relation aux concepts formels et leur énumération. En troisième point, nous formulons le problème de modélisation en ACF dans un contexte logique, l'ASP. Dans notre cas, le travail de compression de graphe nécessite des calculs lourds et du code spécifique, puisque c'est tout l'espace des concepts qui doit être exploré. En quatrième point, nous avons testé notre code sur des réseaux biologiques réels, et explorés les données pour y trouver les concepts triplets que nous avons définis. Finalement, nous présentons Biseau, un environnement ASP à usage général pour la théorie des graphes, que nous appliquons à l'ACF dans le but d'en illustrer les capacités.

0.2 Compression de powergraphe avec l'ACF

Tout d'abord, nous formulons et étudions la compression powergraphe dans le cadre de l'ACF. Notre objectif est de mieux comprendre l'espace de recherche que doivent explorer les programmes qui implémentent cette compression. Nous partion de la notion de contexte de graphe, un type spécial de contexte formel représentant des graphes simples non orientés, et qui nous servent de base pour encoder la recherche de motif dans les graphes et la compression de ces motifs en groupes de nœuds et d'arcs. Cette approche nous permet de proposer une formalisation générale de la compression de graphe, soulignant les principales sources de difficultés du problème. Nous décrivons le problème de la compression comme un problème d'optimisation, où les solutions optimales sont les séquences de motifs à compresser qui minimisent le nombre d'arcs nécessaires pour décrire le graphe. L'espace de recherche est, dans sa forme la plus simple, l'ensemble des combinaisons possibles des concepts formels dérivés du graphe. La taille de cet ensemble forçant à n'en explorer qu'une partie, nous formalisons également l'heuristique gloutonne proposée par Royer *et al.* consistant à ordonner les concepts par leur couverture

en arcs : il s'agit de compresser en priorité les motifs les plus grands. Cette réduction significative de l'espace de recherche, si elle ne garantit pas l'optimalité de la solution, rend le problème approximable pour des graphes réalistes.

Cette vue générale du problème initial nous donne la capacité de formuler des variantes ainsi que des approches pour sa résolution, et nous permet de découvrir des limites à l'approche gloutonne de l'Analyse PowerGraph : nous détaillons notamment un motif de graphe particulier, le *cycle de concepts*, qui prévient toute heuristique gloutonne — cherchant uniquement des motifs maximaux ou basée sur les concepts formels — d'atteindre une compression optimale, c'est-à-dire minimisant le nombre d'arcs.

Nous proposons également des optimisations par réduction du treillis de concept, ou de l'espace de recherche que nous avons implémenté dans un outil, *PowerGrASP*, Nous terminons par une proposition théorique amorçant la conception d'une stratégie *diviser pour régner* pour résoudre le problème de recherche de motif.

Parmi les pistes identifiées pour la réduction de l'espace de recherche, nous nous sommes particulièrement intéressés aux limites induites par l'usage des concepts formels et de leur treillis, pour la représentation de l'espace de recherche. L'une d'entre elle est la séparation des cliques et des bicliques, cherchées et traitées parallèlement dans deux espaces différents, nécessitant donc la création et la maintenance de deux ensembles de concepts formels pour réaliser la compression. Notre réponse à ce problème est la notion de *concept triplet*, un nouvel objet mathématique définissant un espace de recherche intégrant à la fois les cliques et les bicliques.

0.3 Les concepts triplet : une extension de l'ACF pour la recherche de biclique chevauchantes

Nous proposons dans cette section une extension des concepts formels, les concepts triplets, conçu spécifiquement pour les contextes de graphes. Les concepts triplets représentent un motif de graphe plus général que les cliques et les bicliques,

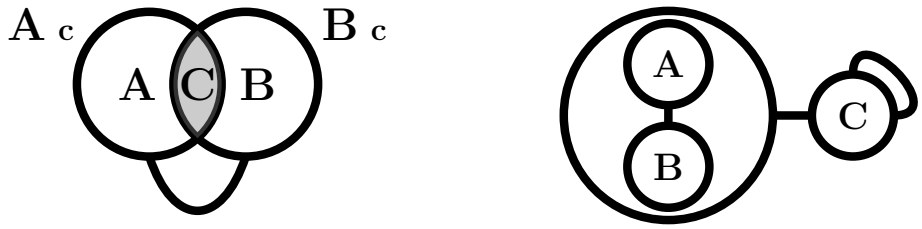


FIGURE 3 – *Gauche :* Représentation à arc unique d'un concept triplet (A, B, C) comme deux ensembles de nœuds chevauchants liés l'un à l'autre, A_c et B_c. $A = A_c \backslash B_c$ et $B = B_c \backslash A_c$. $C = A_c \cap B_c$ est l'ensemble de nœuds liés à touts les autres nœuds du motif, aussi appelés nœuds *ubiquitaires*. A, B et C sont des ensembles disjoints.
Droite : Une compression powergraphe du concept triplet (A, B, C), où C est une clique (d'où le powerarc réflexif) et (A, B) une biclique.

qui en sont des cas particuliers : les bicliques chevauchantes.

Nous introduisons d'abord le terme de *triplet* pour référencer un type de concept formel dérivé d'un contexte de graphe réflexif. Du point de vue des graphes, un triplet est une biclique chevauchante, c'est-à-dire une biclique où les deux ensembles de nœuds peuvent être en intersections non vide. Les triplets sont écrits en utilisant la notation (A, B, C).

Lorsqu'aucun autre triplet ne couvre strictement plus d'arcs qu'un triplet T, on dit que T est arc-maximal, ou tout simplement maximal. Les triplets maximaux sont donc un sous-ensemble des triplets, et décrivent des bicliques chevauchantes maximales dans le graphe. Les triplets peuvent également être écrits dans une forme canonique, afin d'assurer l'unicité de notation d'un triplet donné. La notation d'un triplet est canonique lorsqu'elle obéit à deux conditions : il n'y a aucun nœud ubiquitaire dans A ou B, et $min(A) = min(A \cup B)$.

Les concepts triplets sont simplement les triplets qui sont à la fois maximaux et écrits de manière canonique. Les concepts triplets peuvent être utilisés pour décrire une variation de l'Analyse PowerGraph, où plutôt que considérer les cliques et les bicliques, le processus de compression cherche et traite les bicliques chevauchantes. Nous étudions le rendu graphique du cas général et des cas spéciaux (notamment les bicliques, cliques, et stars), montrant ainsi que les triplets peuvent facilement être incorporés dans l'Analyse PowerGraph sans changements fondamentaux dans

sa définition, c'est-à-dire sans briser les trois conditions détaillées dans la section 0.2.

Nous définissons les triplets *admissibles* comme un sous-ensemble des triplets qui respectent un certains nombre de conditions globales. Ainsi, avec quelques tests, il est possible de filtrer les triplets qui ne peuvent, de par leur non maximalité, être des concepts triplets. De fait, il limite le nombre de triplets à comparer deux à deux pour déterminer quels sont les triplet maximaux, et donc diminue grandement le temps nécessaire à calculer les concepts triplets. Les conditions globales reposent sur l'observation qu'un nœud extérieur au triplet, mais qui est connecté à touts les nœuds de A et C, pourrait être ajouté à B, ainsi qu'un nœud dans A ou B ne peut pas être lié à touts les autres nœuds. Sinon il serait ubiquitaire, et donc devrait appartenir à C. Il est intéressant de noter que nous n'avons pas pu démontrer que les triplets admissibles sont maximaux, bien que dans toutes nos recherches, nous n'avons jamais trouvé un triplet admissible qui ne soit pas maximal.

Nous avons montré également que les concepts triplets sont, à l'instar des concepts formels, partiellement ordonnés, mais qu'avec la définition actuelle de la forme canonique, ils ne forment pas un treillis. Obtenir une définition améliorée de la forme canonique, débouchant sur la constitution d'un treillis des concepts triplets, est un problème ouvert.

Comme nous présentons la modélisation du problème du point de vue logique, nous abordons l'implémentation en ASP de la recherche des concepts triplets. Nous détaillons et comparons l'efficacité de trois méthodes différentes pour la génération de concepts triplets avec leurs encodages respectifs en ASP. Chaque méthode se base sur une représentation différente du graphe d'entrée : l'ensemble des relations dans le contexte du graphe, l'ensemble des concepts formels qui en sont dérivés, ou l'ensemble de concepts formels dérivés du contexte réflexif du graphe. Nous avons comparé ces méthodes sur des jeux de données aléatoires, montrant que si la recherche de concepts triplets a une complexité exponentielle dans tous les cas, les premières méthodes restent nettement plus efficaces.

Enfin, nous avons exploré le lien entre concepts triplets et énumération des cliques maximales, et montré que les concepts triplets forment une représentation compressée des cliques du graphe.

Notre proposition de représentation explicite des bicliques chevauchantes par le biais des concepts triplets semble être une puissante généralisation du processus de compression. Avec les concepts triplets, les deux motifs de l'Analyse PowerGraph (biclique et clique) sont unifiés en un seul. Un programme de compression reposant sur les concepts triplets pourrait encoder l'ensemble de l'espace de recherche avec ce seul motif.

Perspectives

Nous proposons de nombreuses pistes pour le développement des concepts triplets et leur intégration comme une extension de l'Analyse PowerGraph. Par exemple, la représentation des efficace bicliques chevauchantes suppose de changer la définition d'un powergraph. Notamment, la condition sur la hiérarchie des powernœuds doit être étendue pour permettre l'usage de la représentation réduisant un triplet à un seul powerarc.

Nous terminons ici en listant quelques problèmes ouverts que notre approche a soulevés.

Comment trouver les concepts triplets *nécessaires*, c'est-à-dire ceux dont toute compression optimale requiert la compression ? Peut-on prédire le score d'une compression qui résulterait d'une liste de concepts (triplets ou formels), sans réaliser la compression ? Peut-on trouver une structure en treillis pour organiser les concepts triplets ? Un triplet admissible est-il nécessairement maximal ? Autrement dit, la maximalité est-elle équivalente à l'admissibilité ?

0.4 Applications de l'Analyse PowerGraph

Les données

Nous avons travaillé sur deux jeux de données biologiques. Le premier, *rna*, vient d'une étude en transcriptomique sur le puceron du pois, *A. pisum*, en collaboration avec Denis Tagu (INRA Le Rheu) [109]. Il s'agit d'un réseau bipartite d'interaction de 15 mi-ARN avec 1810 ARN messagers. Le second réseau, *mdb*, est étudié dans le cadre d'une collaboration avec Nathalie Théret (Inserm Rennes), et est extrait de la base de donnée MatrixDB décrivant les interactions de protéines extra-cellulaires [69]. Nous avons obtenus différents résultats sur ces réseaux, en

devant parfois nous limiter à leurs versions réduites. En effet, ces réseaux étant de grande taille, notamment *mdb*, certaines opérations n'ont pu être réalisées en un temps raisonnable sur les réseaux complets.

Ce que l'on cherche dans les données

D'abord, nous compressons les graphes, et comparons les deux implémentations de l'Analyse PowerGraph que nous connaissons : celle de Royer *et al.*, *Oog*, et la nôtre, *PowerGrASP*. Nous montrons ainsi que les deux implémentations produisent des résultats équivalents, validant notre approche de la formalisation.

Nous cherchons également dans les réseaux à y déterminer la place et les particularités des concepts triplets/bicliques chevauchantes. Comme décrit dans la section précédente, ce motif de graphe permet d'unifier les cliques et les bicliques. La question est donc de savoir si l'utilisation de ce motif change fondamentalement le processus de compression, ou si au contraire cette généralisation est transparente du point de vue de la compression finale.

Ce que l'on trouve dans les données

La première observation est qu'il n'y a pas de dégradation de la compression powergraphe, lorsque l'on compare notre implémentation *PowerGrASP*, reposant sur notre formalisation du problème et la recherche de concepts formels, et *Oog*, l'implémentation de référence de Royer *et al.*. Cela valide notre approche de la formalisation, bien que notre implémentation ne soit pas aussi efficace sur des graphes de grande taille.

En fait, les deux implémentations utilisant la même stratégie, la réduction en arcs ne diffère que par de petites variations inhérente au processus non déterministe qu'est le choix des motifs à compresser. *Oog* utilisant un algorithme dédié, il est aussi plus efficace.

Concernant les concepts triplets, un résultat important, quoiqu'attendu, est qu'une implémentation de Analyse PowerGraph reposant sur les concepts triplets plutôt que les cliques et les bicliques atteindrait les même résultats, à moins que des bicliques chevauchantes ne se soient présentes dans les données. En effet, une biclique chevauchante peut grouper ensemble des composés qui ne l'étaient pourtant pas dans les compression powergraphe classique.

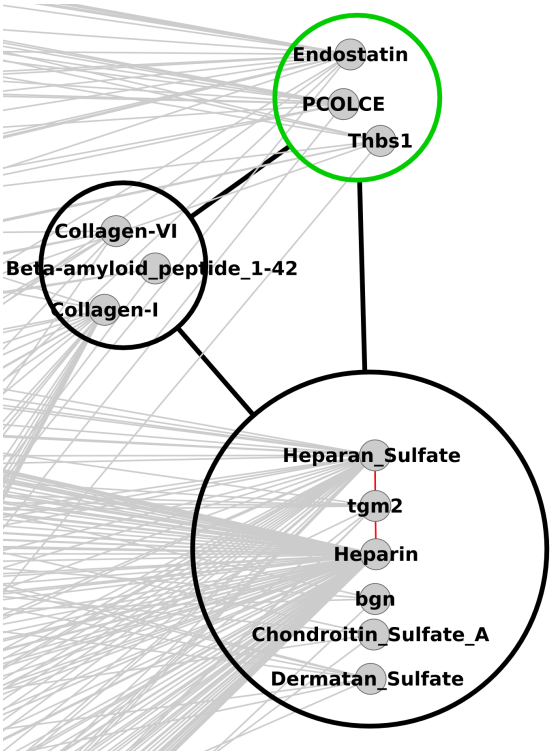


FIGURE 4 – Le plus grand concept triplet dans la version réduite du réseau *mdb*, avec 3 éléments dans l'ensemble C (en vert). En rouge, on trouve deux arcs internes dans le cluster de l'héparine qui ne sont pas couverts par le motif lui-même. Les arcs liants les nœuds du triplet aux nœuds à l'extérieur du triplet sont montrés en gris clair, soulignant l'enchevêtrement de ce motif dans le reste du réseau.

Le plus grand concept triplet, dans le réseau *mdb* réduit, ayant au moins 3 éléments dans l'ensemble C est montré en Figure 4. Le groupement réalisé semble rassembler des composés ayant un rôle dans l'angiogénèse, processus biologique ayant trait à la création de vaisseaux sanguins.

Optimisations

Nous avons également effectué une comparaison des performances de *Power-GrASP*, selon l'usage des optimisations présentées dans le chapitre sur la formalisation de l'Analyse PowerGraph (cf section 0.2). Il apparaît que dans leur ensemble ces optimisations diminuent sensiblement le temps de compression, suggérant une réduction significative de l'espace de recherche.

0.5 Biseau : un environnement ASP pour la visualisation et la spécification de haut-niveau en théorie des graphes

Cette thèse est intimement liée à la notion de programmation logique, présente tout au long de la recherche. Cette approche nous a permis de développer des concepts complexes de manière très compacte. Un des programmes majeurs développé dans le cadre de cette thèse, est l'environnement de développement Biseau. Partant de l'idée de créer un lien entre Programmation par Ensembles Réponses et Analyse de Concepts Formels, nous sommes arrivés à la définition d'un environnement dédié à l'analyse de graphes en ASP, dont l'ACF a été la première application.

Biseau est un compilateur prenant en entrée un encodage ASP décrivant un graphe, et produisant une représentation graphique dudit graphe. Pour cela, Biseau utilise le langage de description DOT permettant une description déclarative de graphes de tout types. En pratique, Biseau implémente un compilateur ASP-vers-DOT, et utilise le programme graphviz pour le rendu graphique, par exemple, en ACF, pour afficher le treillis des concepts. L'usage d'ASP pour générer du DOT permet de décrire un graphe en intension.

Contrairement à d'autres outils comme LatViz et FCAbundles, Biseau ne cherche pas à être efficace sur une tâche précise : il cherche au contraire à être le plus souple possible, afin de permettre à l'utilisateur de créer ou d'adapter facilement une approche, spécifique à son problème ou ses données. Là où, par exemple, LatViz permet d'explorer efficacement des treillis complexes et de grande taille, Biseau permet de définir le treillis lui-même et la manière de l'afficher.

Nous avons montré quelques possibilités offertes par Biseau pour le prototypage et l'exploration de relations mathématiques. Nous reproduisons ainsi les principaux résultats de l'Analyse de Concepts Formels (treillis, AOC poset), ainsi que quelques extensions, notamment les treillis iceberg, l'ACF à 3 dimensions et les *pattern structures* appliquées au nombres entiers.

Avec cette présentation du logiciel, nous espérons proposer à la communauté ACF un guide pour l'usage d'ASP, et encourager l'usage de Biseau ou plus généra-

lement du prototypage via des langage déclaratifs pour l'exploration et la définition de relations mathématiques.

Biseau est actuellement développé avec deux objectifs majeurs : permettre aux utilisateurs de partager simplement les unités minimales de code qu'ils produisent, implémentant potentiellement des idées nouvelles en cours de formalisation, et apporter une interface pratique et puissante pour la composition de code ASP permettant de facilement implémenter une première version d'un logiciel implémentant un nouvel objet mathématique.

0.6 Conclusion

Cette thèse propose un cadre de travail, des résultats théoriques et des implémentations pour une compression de graphe sans perte et concervant de bonnes propriétés de visualisations. Elle développe une méthode particulière, l'Analyse PowerGraph, qui a déjà fait ses preuves dans de nombreuses applications, notamment en bioinformatique. Nous listons ici nos principales contributions.

Analyse PowerGraph et Analyse de Concepts Formels

Nous avons présenté une formalisation de l'espace de recherche de l'Analyse PowerGraph, en tant que problème d'optimisation, par le biais de l'Analyse de Concepts Formels. Nous avons décrit deux versions de cet espace, dont la taille exponentielle au regard de la taille du graphe considéré mène à au caractère NP-complet de la recherche. Nous avons ensuite formalisé l'heuristique de recherche proposée par Royer *et al.*, et proposé notre propre implémentation, *PowerGrASP*, ainsi que des optimisations qui sont applicables dans toute implémentation de la compression powergraphe. Nous avons également montré les limites de l'approche gloutonne, en démontrant qu'elle n'était pas nécessairement capable de trouver une solution optimale. Par cette étude, nous avons montré de nombreuses pistes pour la conception d'extensions de la compression powergraphe.

Triplet Concepts

Une contribution majeure de cette thèse est l'unification des deux motifs de graphes utilisés par l'Analyse PowerGraph : les cliques et les bicliques. Les concepts

triplets sont la formalisation du point de vue de l'ACF du motif de biclique chevauchante, où deux ensembles de nœuds formant une biclique sont en chevauchement. Puisque les cliques et bicliques sont des cas particuliers de la biclique chevauchante, cette contribution théorique unifie les recherches de ces motifs normalement opérées séparément en Analyse PowerGraph. En ACF, les concepts triplets sont un sous-ensemble des concepts formels dérivés du contexte formel décrivant le graphe à compresser. Nous avons également détaillé et comparé différentes méthodes pour générer ces concepts triplets, à partir de plusieurs représentations des données d'entrée, et montré que les concepts triplets représentaient les cliques (maximales) du graphe de manière compacte.

Applications

Nous validons notre approche reposant sur les concepts formels, implémentée le logiciel *PowerGrASP*, en reproduisant les résultats de l'Analyse PowerGraph. Nous montrons également le gain de performances des différentes réductions de l'espace de recherche que nous avons proposées. Finalement, nous avons étudié la présence de concepts triplets dans les données, montrant que le motif était utilisable dans le cadre de la compression powergraphe, reproduisant les même résultats, tant que des bicliques chevauchantes sont absentes dans les données. La recherche de concepts triplets stricts permet aussi de découvrir des structures différentes de celles trouvées par l'approche bicliques+cliques.

Contribution logicielle

Outre *PowerGrASP*, discuté tout au long de la présentation de notre formalisation, nous avons proposé Biseau, un environnement à usage général pour la théorie des graphes que nous avons utilisé pour la reproduction des principaux objets mathématiques de l'ACF, notamment le treillis de concept et ses structures dérivées. Avec cet exemple particulier, nous espérons avoir montré que Biseau est adapté à un large champs d'applications comme outil d'exploration pour les travaux reposant sur la théorie des graphes.

Publications

FCA in a Logical Programming Setting for Visualization-Oriented Graph Compression

Publié à l'*International Conference of Formal Concept Analysis* 2017 (ICFCA'17) [20], cet article présente la formalisation de l'Analyse PowerGraph résumée dans la section 0.2 et approfondie dans le chapitre 2.

Concept Lattices as a Search Space for Graph Compression

Publié à l'*International Conference of Formal Concept Analysis* 2019 (ICFCA'19) [19]. Cet article présente des résultats, résumés entre autre dans la section 0.3 et approfondit dans le chapitre 3, ainsi que les cycles de concepts et les réductions de treillis résumés dans la section 0.2 et détaillés dans le chapitre 2.

Entre la publication de cet article et la rédaction de la thèse, de nombreux développements ont été explorés. Par conséquent, un nouvel article plus complet et tourné spécifiquement vers les concepts triplets est en cours de rédaction.

An Answer Set Programming Environment for High-Level Specification and Visualization of FCA

Publié au workshop *Formal Concept Analysis for Artificial Intelligence* (FCA4AI), lors de l'*International Joint Conference on Artificial Intelligence* et de l'*European Conference on Artificial Intelligence*, en 2018 [17]. Cet article présente Biseau, résumé dans la section 0.5 et approfondit dans le chapitre 5.

Biseau : An Answer Set Programming Environment for High-Level Specification and Graph Visualization applied to FCA

Publié au workshop *Applications and Tools of Formal Concept Analysis* lors de l'International Conference of Formal Concept Analysis 2019 (ICFCA'19) [18]. Cet article reprend les principaux éléments de l'article précédent, mais réoriente les résultats et la discussion afin de s'adapter au sujet du workshop : unifier les efforts de développement de la communauté dans le but de simplifier les échanges et l'implémentation de nouveaux concepts et relations mathématiques.

Chapter 1

Graph compression for visualization and Answer Set Programming

---

> The whole idea of what happens when you read a book, I find absolutely stunning. Here's some product of a tree, little black squiggles on it, you open it up, and inside your head is the voice of someone speaking, who may have been dead 3000 years, and there he is talking directly to you, what a magical thing that is.
>
> Carl Sagan
> May 20th, 1977

1.1 Introduction

A graph is both a simple representation of relations between elements and a complex mathematical object. Graphs come in a wide variety of uses and forms. Typical use is the encoding of theoretical problems, such as knowledge base representation, language modelling, network analysis, flux analysis, cartography, community detection, game theory, path finding, planning and more generally operations research. A graph can also represent data: in biology for instance, produced data are often represented in form of networks whether they are real interaction networks of living organisms, such as metabolic, regulation, connection or signaling networks, as well as abstract networks of related data such as genome assembly networks. In other fields, one can cite distributed computing, space partitioning (triangu-

lation, k-d tree), Web graph, the Resource Description Framework (RDF), social networks, meshes (geography, geology, simulation), Feynman diagrams (particle physics), molecule graphs (chemistry), circuit design and electrical network analysis (electrical engineering), traffic network (urban planning), and more generally network sciences.

Beyond being an abstract data type, a graph has the strong advantage of having a graphical representation. But this representation does not scale well. While a graph is only a collection of vertices with links between them, as soon as it contains more than a dozen objects it becomes uninterpretable without a well-designed layout of nodes and the highlight of subsequent structures. Real-data graphs come in very different size: they easily exceed thousands or millions of elements. Examples include, in increasing order of size: interaction network of specific molecules in specific tissues, co-authorship networks in academic literature, the Wikipedia edition dataset, or a whole human genome assembly. The larger the graph, the more robust the visualization technique must be to keep the overall structure of the graph readable. But at some point, even the best layout techniques do not prevent the *fur-ball* effect, where ultimately the graph is just graphically rendered as a big black smudge. To overcome this effect, imposed by computational, screen or human limits, to apprehend millions of objects at a quick glance, many layout, summarization, and compression approaches have been studied.

The computation of a layout for the visual rendering of a graph consists into finding positions of nodes and edges on the plane or a 3D space, in order to achieve a pleasant, readable and easy to interpret representation of the underlying data [54]. Most layout systems are force-directed, where the graph is assimilated to a physical system where nodes and edges are attracted and repelled by different forces. The task is then modeled as an energy minimization problem, thus relies mainly on the optimization of various metrics, such as the minimization of edge crossings, the distance between neighbors, or edge length variation. Such an approach is however insufficient for large graphs, where the density of underlying structures prevents any layout to achieve a readable rendering of the graph. It is necessary to preprocess the graph in such cases in order to summarize its content.

The automatic summarization research field consists into the extraction of meaningful information from various types of data. Often applied to text and

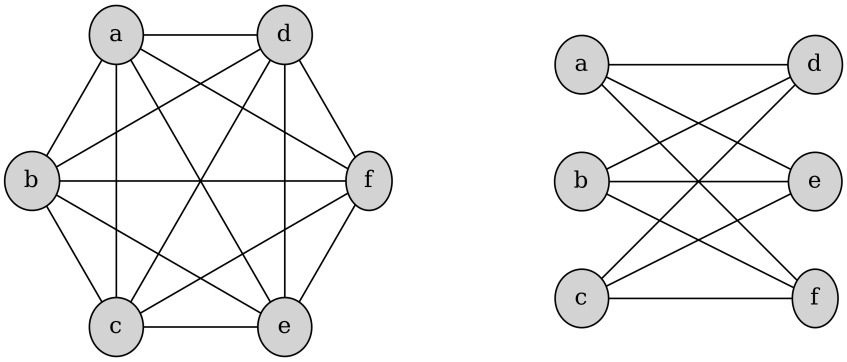


Figure 1.1 – Two examples of graphs, that happen to be perfectly formed motifs. *Left:* A clique of 6 nodes, covering 15 edges. *Right:* A biclique of 3 × 3 nodes, covering 9 edges. A star is a biclique of 1 × n nodes.

videos, it is also applied on graphs. In [66] for instance, typical graphs motifs such as cliques, bicliques (cf Figure 1.1), stars and chains are detected in the graph, and a minimal amount of them is selected to account for the whole graph structure, and is presented in a final summary made of a graphical representation highlighting the retained motifs. And since the graph motifs have a meaning in the underlying data, the final report is interpretable by humans. Such a representation of the graph may also achieve a lossless compression of the graph by describing it only in terms of structures of elements, effectively performing a compression of the data needed to store the graph.

Graph compression is the central subject of this thesis and we will particularly work on the clique and biclique motifs.

1.1.1 Graph compression

Graph compression is a broad term describing a wide array of techniques and in a large set of research domains, including biology, linguistics, knowledge representation, graph databases, graph layout, graph summarization or decomposition [14].

One of the objectives of graph compression is to represent the graph in a compact way, in order to speed up costly treatments such as Maximal Clique Enumeration (MCE) [60]. Graph compression also encompasses the storing and request of dynamic graphs more efficiently, for instance by encoding the graph

as bitmaps compressed with standard methods but still practical for querying. Another way to achieve compact representations of graphs in memory is to reduce the amount of elements needed to describe the graph, by using typical graphs motifs just as in the graph summarization protocol described previously, or by detecting repeated patterns that are specific to the input graph. For instance, authors in [100] build a Vertex Replacement Grammar, whose rules describe motifs found by hierarchical clustering in the graph. Its expansion results in the generation of graphs sharing common properties with the input graph. In that latter case, the goal of compression is the extraction of the main features of the graph for a general study of its structure.

Lossless graph compression and hierarchical clustering

Another application of graph compression is readability, where the compression process looks for possible node or edge aggregations, i.e. connections between clusters of nodes instead of connections between individual nodes. The clusters may be determined by various hierarchical clustering methods based for instance on the nodes' annotations or their place in the network. The resulting clusters are used to generate a simplified rendering of the input graph, where clusters are rendered as special objects in the graph representation, in order to minimize the number of edges needed to described the whole graph. Not surprisingly, it has been shown that graph readability increases with edge reduction [36]. When the compressed graph is exactly equivalent to the input graph, the compression is said lossless. A hierarchical organization of nodes and edges, combined with a lossless clustering enables a vast range of justified and abstractions of input graph with multiple applications ranging from graphical rendering to functional analysis and algorithmic aspect of graphs. Among early works in this direction, Agarwal *et al.* [2] have shown that visibility graphs, a type of graph commonly used in computational geometry, can be represented compactly as a union of cliques and bicliques. The decomposition (partition of the edges) or the covering (multiple use of edges) of graphs into subgraphs belonging to a particular family have been the subject of many studies. Bounds on the size complexity of such coverings have been early established for the important particular case of complete bipartite subgraphs [29] but many interesting open combinatorial problems remain in this area [61]. From

an algorithmic perspective, the generation of all maximal bicliques of a graph is related to and may be considered as an important subtask of the covering problem. Apart from algorithms developed in Formal Concept Analysis, applied mathematics have also worked on classes of graphs for which it is possible to find the set of all maximal bicliques in time polynomial in the size of the graph. For instance, it is possible to find linear time algorithms for the case of graphs of bounded arboricity [39] or for domino-free graphs [6]. For general graphs, the best one can hope is to get total polynomial algorithms, i.e., polynomial with respect to the size of the union (input + output). This has been proposed in [5], with an algorithm derived from the consensus method. The compression of graphs through a hierarchical clustering of nodes is a widely explored field, with many methods for the determination of clusters. For example, authors in [59] propose a genetic algorithm where each individual encodes a succession of node merges into clusters. The fitness function minimizes, knowing a graph and a target compression ratio, the number of edges falsely lost or added by the merges.

In this thesis we were specifically interested in hierarchical methods based on clique and biclique analysis in graphs. In this domain, two methods that achieve a lossless compression have been designed with different goals quickly introduce here, namely Modular Decomposition and Power Graph Analysis.

Modular Decomposition

Modular Decomposition is an efficient deterministic approach for graph compression. It has been used for instance to compute graphical representations [87], enumerate the maximal cliques [12], or find functional modules in protein-protein interaction networks [42]. It has been thoroughly studied in term of search space and algorithmics [57]. The modular decomposition of a graph is computed by merging nodes forming a strong module in the graph, i.e. a group of nodes having the exact same neighbors outside of the module, and if the module is either a subset, a superset, or disjoint to other modules. The process is repeated, since formed modules may in turn share the same neighborhood with another node or module, iteratively building a hierarchy of modules, typically represented by a tree of modules. For instance, each set of a biclique (A, B) would first be grouped as such, since all nodes in it share the same external neighbors (i.e. the nodes of the other set).

The two resulting modules having the same external neighbors (that is none), they are merged, constituting the final module of the graph. In case of a clique, only one module is created, annotated to recall that the nodes in it are all linked together. Since there is only one set of strong modules for a graph, and because modules are either adjacent (all nodes of a module are linked to all nodes of another) or non-adjacent (no edge between the nodes of the modules), the process is deterministic. It thus has the great advantage of yielding, in quasi-linear time, a unique decomposition for a given graph. However, that deterministic behavior comes at a cost, that of not being expressive enough to achieve high compression rate: when two motifs (bicliques for instance) overlap, many unrelated strong modules are formed, and grouped together by modular decomposition, compressing poorly both motifs. Because of the requirement on exact neighborhood, modular decomposition cannot render well a motif and poorly another. It has to decompose fully both of them. In order to obtain a clearer view of at least a part of the motifs, one has to introduce the notion of *choice*, to favor motifs instead of others. This is the approach of the next compression method.

Power Graph Analysis

Our own work started from Power Graph Analysis, another method of graph decomposition into hierarchical modules motivated by common neighborhood [90]. In some specific cases, for instance biological networks, data show a preponderance of particular formal structures such as cliques and bicliques, that can for instance represent respectively protein complexes and domain-induced interactions in protein-protein interaction networks [23], or transcription-factor — micro RNA functional modules in regulatory networks [86]. Studying a compression process based on these specific motifs is critical in this case.

Power Graph Analysis was first applied in bioinformatics [94], later refined in [93, 90] and applied to other domains. As it is one of the main subject of that thesis, it is presented extensively in its own section 1.2.

Power Graph Analysis shares some similarities with Modular Decomposition, but differs in an important manner: nodes are grouped together when sharing *similar* neighborhood, a relaxation of constraint compared to Modular Decomposition. The powergraph of a graph — the compressed version of the graph — is no

longer necessarily unique. For instance, in the case of two overlapping motifs, one may first compress (possibly fully) one of them, the other, or the intersection of the two, then to compress the remaining edges, which also may need some choices. The introduction of these choices makes Power Graph Analysis a significantly harder problem: the determination of an optimal powergraph — presenting the minimal number of edges to describe the graph — is an NP-complete problem [37]. The approach show great results in biology, successfully associating motifs with functional modules in various types of networks [94, 93, 32, 109]. Power Graph Analysis has also been explored with variations, for instance applied on directed graphs [37], or with some relaxation of constraints on the cluster hierarchy [106] or on the edge compression [37]. In this thesis, we are interested by undirected simple graphs.

1.1.2 Finding cliques and bicliques in graphs

In graph summarization and compression, two concurrent approaches are studied regarding the nature of graph motifs. In one case, they are specific to the graph, first researched and then used, for instance for memory compact representation or efficient querying. In the other case, the considered graph motifs are general objects, such as clique and biclique motifs, which by their sole structure unravel interpretable structures in the data. In Power Graph Analysis and in this thesis, we consider the second case, and the cliques and bicliques as discussed in the following.

A clique is a set of nodes that are all connected to each other. In a simple graph, a clique constitutes the most dense subgraph, as shown on the left side of Figure 1.1. As such, cliques reveal important structures in the data, for instance protein complexes in interaction networks, or communities in social networks [94, 112, 78]. When a clique cannot be extended with another node, the clique is said *maximal*. The Maximal Clique Enumeration (MCE) in a graph is one of the Karp's 21 NP-complete problems. It however does not preclude their study, as an efficient algorithm for MCE was designed and iteratively refined to handle increasingly larger graphs [22], and recently a distributed implementation was devised in order to solve the MCE for graphs so large that a sequential approach is unpractical [21].

The other typical graph motif is biclique, composed of two disjoint sets of interacting nodes as shown on the right side of Figure 1.1. As for cliques, the enumeration of bicliques in a graph is an NP-complete problem [88]. However, many families of graphs, such as domino-free graphs, enables the enumeration of maximal bicliques in polynomial times [85], although many optimizations enable to obtain a lower bound, taking advantage of graph properties. Biclique is a graph motif unraveled by modular decomposition [42], and typically searched in bipartite graphs such as drug-disease or transcriptional networks [32, 86]. The star motif is a special case of biclique, where one node, the hub, is linked to a set of other nodes. It is often found in social networks where group leaders and influencers are hub nodes [3]. In some applications, such as Modular Decomposition, the two sets of a biclique are required to be independent. It means that the induced subgraph needs to be bipartite, and that a dense graph cannot generally be decomposed in bicliques. In contrast, we will consider as in Power Graph Analysis a more flexible definition allowing nodes in each set of a biclique to share some connections.

1.1.3 The thesis proposal

The aim of this thesis is to propose a formalization of the Power Graph Analysis search space as a graph compression process looking for motifs with readability constraints.

Our approach is two-fold. First, the theoretical aspect of the problem is addressed in the light of Formal Concept Analysis (FCA) [45], a well-defined mathematical framework presented in section 1.3, from which major results in classification and graph theory are issued, including the search for graph motifs. Second, the practical implementation of our results are performed with Answer Set Programming (ASP), a form of purely declarative programming oriented towards the resolution of combinatorial problems, enabling a high-level specification of mathematical properties [76]. ASP is introduced in section 1.4.

The purpose of this thesis is illustrated in Figure 1.2, where three equivalent representation of a graph are presented: the input graph, its compressed versions, and its concept lattice. This thesis aims at studying the relations between these

representations, and shows they are all equivalent. By definition, the compressed graph is exactly equivalent to the uncompressed graph, and we formalize the compression as an optimization problem. The concept lattice is another representation of the uncompressed graph, and we show its role to encode the compression search space.

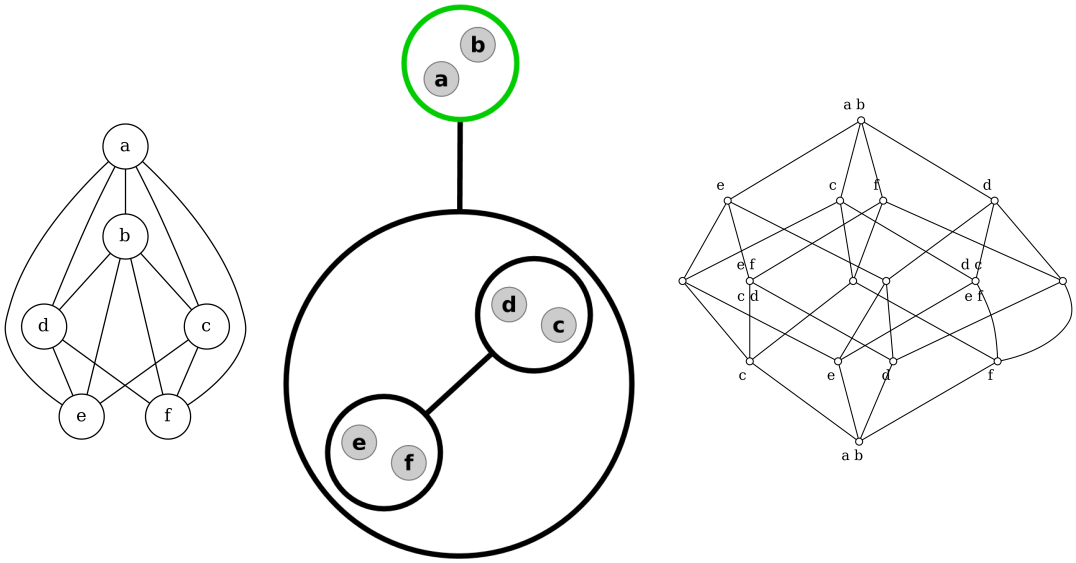


Figure 1.2 – Three representation of the same graph. *Left:* the graph itself. *Center:* its powergraph. *Right:* its concept lattice representation.

Part of the material for this thesis was published in earlier articles [20, 19, 17]. We present in this document a fully revised version of them, including corrections, clarifications and additional content, together with a new FCA contribution on triplet, which is the subject of chapter 3.

1.2 Power Graph Analysis

This introduction is adapted from our paper on the formalization of Power Graph Analysis in FCA presented at ICFCA 2017 [20].

A nice visualization of compressed graphs introduces additional constraints on the choice of subgraphs interesting for compression, which add a complexity level

in the covering or decomposition problem. We have already mentioned that it is useful to allow both cliques and bicliques for more compact representations.

Introduced by Royer *et al.*, Power Graph Analysis is a clustering and visualization method [94] that starts from this requirement and has been specifically designed to show these subgraphs typically arising in bioinformatics. Indeed, they have been associated to important structures in biological networks, particularly for protein interactions [3, 78, 42]. Bicliques also show interactions induced by specific protein domains in case of protein-protein interactions, or for multi-target drugs in case of drug/target/disease networks [32]. Furthermore, this approach has been used as an alternative approach to compare two biological networks by measuring the compression ratio of the compressed union of the graphs [82]. Due to the genericity of the subgraph motifs, Power Graph Analysis has been used for applications in other research fields like reliable community detection in social networks [106].

1.2.1 Power Graphs

Given a graph $G = (V, E)$, a powergraph is defined as a special graph $PG = (PV, PE)$ where the nodes PV are subsets of V and the edges PE are subsets of E. Furthermore, a powergraph must fulfill the three following conditions:

subgraph condition Any pair of powernodes connected by a poweredge represents a biclique in graph G. As a special case, with a slight abuse of notation, a clique in G is represented by a single powernode and a poweredge looping on this node.

powernode hierarchy condition Any two powernodes are either disjoint, or one is included in the other. From the classification point of view, the sets of vertices clustered in powernodes form a hierarchy.

poweredge decomposition condition Poweredges form a partition of the set of edges.

An example of graph compression is shown in Figure 1.3.

The issue is to exhibit a powergraph with a minimal number of poweredges. The solution is not necessarily unique. It has been shown to be a NP-complete

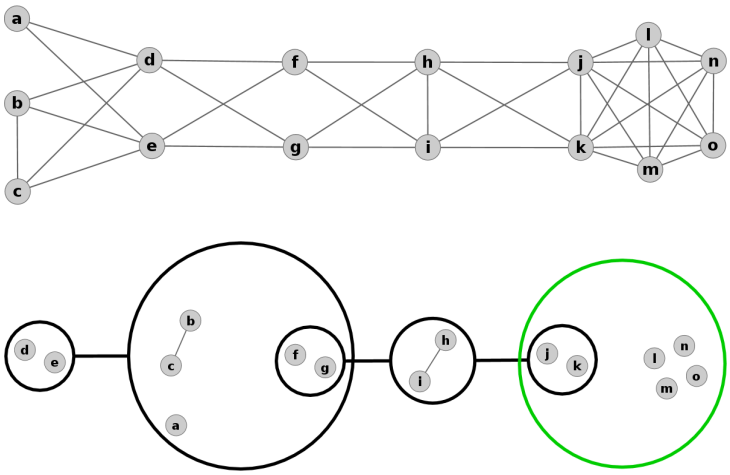


Figure 1.3 – A graph with 15 nodes labelled from a to o and 35 edges. A smart layout of this graph allows one to understand its underlying structure, something that becomes hard with a growing number of edges. The bottom compressed version of this graph has been produced by Power Graph Analysis and rendered with Cytoscape [99], through a plug-in developed by Royer *et al.* Poweredges are shown as thick and black lines linking powernodes (thick circles, black for bicliques and green/grey for cliques). Some edges like (h, i) remain uncompressed. Concept $(\{h, i\}, \{f, g, j, k\})$, despite being a maximal biclique, can't be associated to a single poweredge without breaking the powernode hierarchy condition. Instead, two poweredges are generated, corresponding to bicliques $(\{h, i\}, \{f, g\})$ and $(\{h, i\}, \{j, k\})$. The same way, the subgraph on the subset of vertices $\{d, e, f, g, h, i, j, k\}$ can't be covered by poweredges $(\{f, g\}, \{d, e, h, i\})$ and $(\{h, i\}, \{f, g, j, k\})$ since it would break the poweredge decomposition condition.

problem [37]. An algorithm and a software are described in [94]. It implements a two-phase approach, first processing the possible powernodes by a hierarchical clustering of the nodes using the Jaccard index on the sets of neighbors, and then building the poweredges than can be drawn between any pair of powernodes following a greedy incremental approach, choosing at each step a maximal subgraph in the number of covered edges. The algorithm is very fast but remains heuristic and only computes an approximation of the minimal powergraphs. It also has been implemented with slight variations, for instance to work on directed graphs, or to enable overlapping powernodes for visualization of relations between non-disjoints sets [3], or to enable an edge to be used multiple times for a faster search for near-to-optimal compression [37].

1.3 Formal Concept Analysis (FCA)

This thesis addresses a formalization of Power Graph Analysis with Formal Concept Analysis, a mathematical framework based on lattice theory, first defined and developed by R. Wille and B. Ganter in the early 80s. FCA benefits from solid foundations [45], and of many extensions such as n-ary concept analysis for multidimensional datasets [72], fuzzy concepts and concepts in possibility theory to handle vague and uncertain data [46, 35], use of pattern structures to handle intervals, orders or graphs as attribute value [44], or relations between concepts issued from other contexts with Relational Concept Analysis (RCA) [92].

It has been used for the formalization of a wide array of domains, such as knowledge discovery, classification, and graph theory [101, 25], leading to numerous applications in numerous fields such as software engineering [67], ontology design [30], knowledge processing [89], or network analysis. This last domain is of particular interest to us: for instance in bioinformatics, where FCA's ternary concepts were used to mine patterns in gene expression [63], and in social networks, where metrics to monitor communities and viral content are expressed in the framework of FCA [11].

FCA and graph theory

The duality graph/formal context was explored in many works. For instance,

to represent formal contexts by a graph where nodes correspond to the formal concepts [65].

Regarding graph motifs, there is a one-to-one mapping between the formal concepts issued from the formal context representation of a graph and its maximal bicliques, as described in [73], motivating the design of algorithms based on the concept lattice for the enumeration of these motifs [53, 1]. Fuzzy contexts are contexts yielding formal concepts despite missing relations, an approach shown to be similar to the search of quasi-bicliques [46], enabling for instance a more robust detection of communities in noisy data.

1.3.1 An introduction to FCA most fundamental concepts

Formal Concept Analysis revolves around the notion of *formal concept*, a maximal set of objects and attributes so that objects all have the attributes, and all attributes are held by the objects. Those formal concepts are issued from a *formal context*, typically a binary table such as Table 1.1, where each row is an object, and each column an attribute.

The formal context (O, A, R) describes a binary relation $R \subseteq O \times A$ over a set of objects O and a set of attributes A. The derivation operator associates a set of objects (resp. attributes) to a set of attributes (resp. objects), such that:

Definition 1.1. *Derivation operator on R*
Given a set of objects X (resp. attributes Y), the set X' (resp. Y') is made of all attributes (resp. objects) related to objects in X (resp. attributes in Y):

$$X' = \{y \in Y \mid \forall x \in X,\ r(x, y)\} \qquad Y' = \{x \in X \mid \forall y \in Y,\ r(x, y)\} \tag{1.1}$$

As for derivation, the derivation operator can be combined multiple times:

$$X'' = \{x \in X \mid \forall y \in X',\ r(x, y)\} \qquad Y'' = \{y \in Y \mid \forall x \in Y',\ r(x, y)\} \tag{1.2}$$

For instance, in Table 1.1, we have $\{bat\}' = \{fly, skeleton, wings, viviparous\}$ and $\{fly, skeleton, viviparous\}'' = \{wings\}$.

Formal concepts as fixed point pairs of subsets over $O \times A$:

	breathes in water (a)	fly (b)	beak (c)	hands (d)	skeleton (e)	wings (f)	lives in water (g)	viviparous (h)	produces light (i)
bat (1)		×			×	×		×	
eagle (2)		×	×		×	×			
monkey (3)				×	×			×	
parrot fish (4)	×		×		×		×		
penguin (5)			×		×	×	×		
shark (6)	×				×		×		
lantern fish (7)	×				×		×		×

Table 1.1 – Formal context *Vertebrate*, reproduced from [25].

Definition 1.2. *Formal Concept* In a formal context (O, A, R), a formal concept is a pair (X, Y), such as:

$$X = \{x \in O \mid (x, y) \in R \ \ \forall y \in Y\} \tag{1.3}$$

$$Y = \{y \in A \mid (x, y) \in R \ \ \forall x \in X\} \tag{1.4}$$

Where $X \subseteq O$ is the *extent* of the formal concept, and $Y \subseteq A$ is its *intent*.

Formal concepts can also be defined using the derivation operator:

Definition 1.3. *Formal Concept (with the derivation operator)* In a formal context (O, A, R), a formal concept is a pair (X, Y), $X \subseteq O$, $Y \subseteq A$, such that $X' = Y$ and $Y' = X$.

Intuitively, a formal concept is a maximal association of objects and attributes, for instance $\{1, 2\} \times \{b, e, f\}$ in Table 1.1. The number of formal concepts in a formal context can be exponential with respect to context size. Note that, in the absence of an object holding all attributes or an attributes held by all objects, both $(all\ objects, \emptyset)$ and $(\emptyset, all\ attributes)$ are valid concepts. Many extensions of FCA extends the formal concept definition, in order to handle integers, intervals, probability, graphs,... instead of a binary value in the formal context. This thesis only relies on binary attributes.

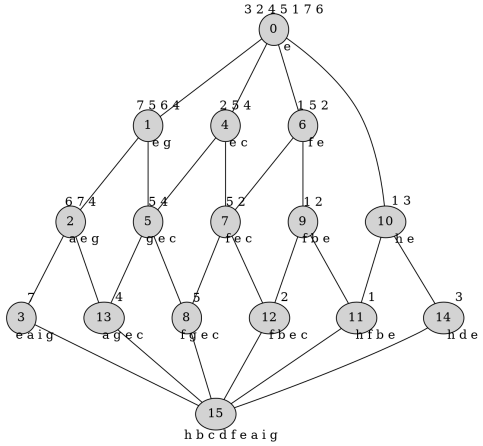


Figure 1.4 – Visualization of the concept lattice of context in Table 1.1 with extent and intent shown for each concept.

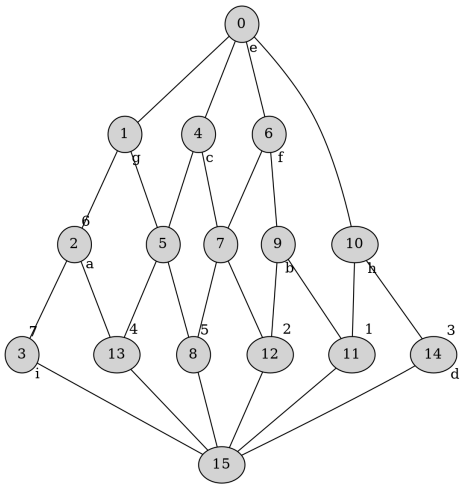


Figure 1.5 – Visualization of the concept lattice of context in Table 1.1 with reduced labelling.

Concept lattice

The set of formal concepts obtained from a formal context is partially orderable, following this inclusion relation:

Definition 1.4. *Order on Formal Concepts* Let $C_1 = (O_1, A_1)$ and $C_2 = (O_2, A_2)$ be formal concepts. $C_1 \leq C_2 \Leftrightarrow O_1 \subseteq O_2$ and $A_1 \supseteq A_2$

As any partially orderable collection, a line diagram can be drawn to visualize the order, as in Figure 1.4, where each node is a formal concept issued from Table 1.1, and two nodes/concepts are linked if directly next to each other in the order. Since the derivation operators defines a Galois connection between the sets of objects and attributes, and because each pair of formal concepts have a greatest common subconcept (*meet*) and a least common subconcept (*join*), the resulting line diagram is a Galois lattice. Because it is constituted of formal concepts, we also call it the concept lattice.

Object and attribute concepts

Object concepts and attribute concepts are two subsets of the formal concepts. The object (resp. attribute) concept of an object o (resp. attribute a) is the concept $(\{o\}'', \{o\}')$ (resp. $(\{a\}', \{a\}'')$). This concept is unique for each object (resp. attribute), and is the smallest (resp. largest) concept with o in its extent (resp. a in its intent).

The most straightforward application of object and attribute concepts are the reduced labelling of concept lattice, as shown in Figure 1.5, where only the object concepts and attribute concepts are labelled with the objects and attributes associated with them. Other applications are typically found in classification, where the concept lattice is reduced to the object and attribute concepts, filtering out the formal concepts that are not object or attribute concepts. This structure, called *AOC-poset*, retains important structure in the data while being bounded in size by $\mathcal{O}(|O| + |A|)$, because in the worst case each object and each attribute, as in Figure 1.5, possesses its own object or attribute formal concept. The AOC-poset was used for software engineering [55] and data classification by [84]. It was used in Relational Concept Analysis to avoid combinatorial explosion of the output [33]. There exists many efficient algorithms to compute the AOC-poset [13].

The last section of this chapter addresses an important specificity of our work, the setting in which we have implemented and tested our approach.

1.4 Answer Set Programming

1.4.1 Principles

The following presentation of ASP is adapted from [20]. For a language reference or an in-depth dive into the language, the reader is redirected to [47] and [76].

ASP is a form of purely declarative programming oriented towards the resolution of combinatorial problems [76]. It has been successfully used for knowledge representation, problem solving, automated reasoning, and search and optimization. Unlike Prolog, ASP handles cross-references of rules, enabling the writing of

code much closer to the specification.

An ASP program consists of Prolog-like rules h :- b_1, ..., b_m, *not* b_{m+1}, ..., *not* b_n, where each b_i and h are literals and *not* stands for *default negation*. Mainly, each literal is a predicate whose arguments can be constant atoms or variables over a finite domain. Constants start with a lowercase letter, variables start with an uppercase letter or an underscore (don't care variables). The rule states that the head h is proved to be true (h is in an answer set) if the body of the rule is satisfied, i.e. $b_1, \ldots, b_m$ are true and one cannot prove that $b_{m+1}, \ldots, b_n$ are true. Note that the result is independent of the ordering of rules or of the ordering of literals in their body, in contrast with Prolog. An ASP solver can compute one, several, or all the answer sets (stable models) that are solutions of the encoded problem. If the body is empty, h is a fact while an empty head specifies an integrity constraint. Together with model minimality, interpreting the program rules this way provides the stable model semantics (see [52] for details). In the head part, A *choice rule* of the form $\{p(X) : q(X)\}$ will generate $p(X)$ as the powerset of $q(X)$ for all values of X. In the body part, $\{p(_)\}$ will count the number of atom p with one parameter, and $N = \{h\}$ evaluates N to the cardinal of the set of h. In the body part, $p(X) : q(X)$ holds if for all X, if $q(X)$ holds, then $p(X)$ holds. Finally, lines starting by % are comments. In practice, several syntactical extensions to the language that are not interesting for this paper are available. In the rest of the chapter, multiple examples of ASP will be presented, which will allow the reader to familiarize himself with these few syntax elements.

For all ASP-related work, we used the Potassco system (*Potsdam Answer Set Solving Collection*), developed in Potsdam University [48], that proposes an efficient implementation of ASP. ASP processing implies two steps, grounding and solving. From a set of ASP rules, the grounder generates a propositional program, replacing variables by their possible values. The solver is in charge of producing the minimum stable models (answer sets) of the propositional program. Of course, a dedicated algorithm for a specific problem will be generally more efficient than its equivalent compact ASP encoding. However, ASP is useful for the design of prototypes. It is an attractive alternative to standard imperative languages that enables to produce quickly safe and reasonably efficient developments.

An overview of ASP applications is presented in section 1.4.2, The main lan-

guage features are then introduced from the FCA point of view in section 1.5.

1.4.2 Applications of ASP

ASP has already been applied to FCA to accomplish expressive query languages for formal contexts [58], later extended to n-adic FCA and improved with additional membership constraints, in order to handle large context exploration [95]. The perspectives of [95] included some design proposal about a graphical user interface allowing user to constrain the search for concepts and obtaining the final selected concept. We implemented such a program [1], with the ASP backend proposed by [95].

ASP has also been used for procedural content generation, i.e. the generation of large instances from comparatively small numbers of examples and constraints, for instance maze generation [79], or map generation for real-time strategy game were resources and player bases has to be placed fairly, while taking accounts of terrain features such as cliffs and roads[102]. ASP is also the representation and reasoning language of an automatic composition system, Anton, extracting melody and harmony from input music, and generating new composition from the derived set of rules [15]. Wave-Collapse Function (WCF) is a quantum theory derived algorithm taking as input a pattern, and reproducing it on larger scales with variations, only ensuring local similarity between input and output. It has many applications, including level generation for games, creation of 3D worlds, and text generation. WCF has been implemented using ASP, redefining WFC as a constraint-solving problem [62]. It was later improved by the incorporation of non-local constraints, such as dependencies or distances, in order to achieve more complete procedural content generation [97, 103].

ASP has been naturally applied to various artificial intelligence tasks, notably in robotics and planning [40]. For instance, ASP was used to implement a strategy for solving the game Angry Birds [24] using the DLVHEX system [38], to encode realistic instances of multi-agent path-finding, typically automated ware-

1. Available at `https://github.com/Aluriak/navicept`

house management, where a fleet of robots with distinct targets must maneuver in a known layout of hallway [81] and order-picking system [98, 50].

1.5 An introduction to ASP for people working with FCA

This section presents examples of ASP encodings related to various Formal Concept Analysis algorithms. It acts both as an ASP introduction, and as a primer for chapter 5, in which an environment is proposed for a more complete exploration of FCA structures (iceberg lattices, for instance).

ASP Syntax Point. An ASP encoding is made of rules ended by a dot. The simplest being the declaration of truth value of an atom, such as `rel(1,a)`. A set of terms arguments of a functional symbol like `rel` implicitly defines a domain of values for it. Atoms can be true under conditions, e.g. `a :- b` indicating that atom `a` is true if `b` is true. For instance, the encoding `b. a :- b.` has one solution with two true atoms `a` and `b`. Mathematical variables are identifiers whose first letter is upper case. Each variable must have a domain and using a variable in an expression is a shortcut for a set of expressions where each variable is replaced by all possible values in its domain. Hence, each variable is universally quantified on a finite domain. For instance, `a(X) :- b(X)` implies that `a(X)` holds for all `X` that are arguments of `b` atoms. Conjunctions are written with a semicolon, such as `a(X) :- b(X) ; X<12`, that restrict `X` to be an argument of `b/1` and less than 12. Note that some built-in operators can act on domains, such as `#max`, which return the maximum value of a domain. Example: `a(X) :- b(X) ; X<#max{V:val(V)}`. The combinations of rules and atoms constitute an ASP encoding that, once compiled in propositional formulas by a grounder, may be solved by a dedicated prover that provides one or all logical models of the set of formulas. Each model is therefore a set of atoms describing a fixed point of the encoding.

	a	b	c	d	e	f	g	h	i
1		×			×	×		×	
2		×	×		×	×			
3				×	×			×	
4	×		×		×		×		
5			×		×	×	×		
6	×				×		×		
7	×				×		×		×

(a) Formal context *Vertebrate*, reproduced from [25].

```
1 rel(1,b). rel(1,e). rel(1,f). rel(1,h).
2 rel(2,b). rel(2,c). rel(2,e). rel(2,f).
3 rel(3,d). rel(3,e). rel(3,h).
4 rel(4,a). rel(4,c). rel(4,e). rel(4,g).
5 rel(5,c). rel(5,e). rel(5,f). rel(5,g).
6 rel(6,a). rel(6,e). rel(6,g).
7 rel(7,a). rel(7,e). rel(7,g). rel(7,i).
```

(b) ASP encoding of the formal context, using `rel/2` atoms indicating when an object is in relation with an attribute.

Figure 1.6 – Running example for the ASP introduction.

1.5.1 Running example data

Figure 1.6 provides the formal context used in this section as a running example, along its ASP representation to be used with the encodings of this section, (subfigure 1.6b). For reference, that formal context is taken from [25], Table 1.1, *Vertebrate context*. Sixteen formal concepts, including supremum and infimum, are issued from it.

In the ASP representation, predicate `rel/2` (of name `rel` and arity 2) encodes related elements in the formal context. For instance, atom `rel(1,a)` implies that object `1` holds attribute `a`. Together with ASP encodings seen later, this constitutes a testable working example.

1.5.2 Formal concept mining: Next Closure algorithm

There are many algorithms for mining formal concepts from a formal context, one of them being *Next Closure*, first detailed in [45] and efficiently revisited in [16]. We chose that algorithm both for its historical importance and its simplicity.

An implementation of *Next Closure* in ASP is proposed in Code 1.1, and commented right away. Note that ASP being a purely declarative language, the order in which rules are written/fed to the solver does not have any impact on the output. In this respect it differs substantially from the algorithm or from other logic programming approaches like Prolog that use a procedural solving approach based on a search tree. ASP uses instead a heavily customizable but non-predictible

heuristic system, that enables in that regard much more powerful expression of mathematical relations, and as a main feature cannot be stuck in infinite recursions.

For reference and help to the reader to understand the ASP encoding, algorithm 1 shows the *Next Closure* algorithm, reproduced from [25].

Algorithm 1 The Next Closure algorithm for computing the set of concepts from a formal context, reproduced from Figure 2.3 of [25].

Require: Context (G, M, I)
Ensure: Computation of formal concepts
1: $C \leftarrow \{(M', M)\}$
2: $currSubset \leftarrow \{max\{g \in G\}\}$
3: $nextObj \leftarrow max\{g \in G\}$
4: **while** $currSubset \neq G$ **do**
5: **if** there is no $g \in currSubset'' \setminus currSubset$ such that $g < nextObj$
6: **then**
7: $C \leftarrow C \cup \{currSubset'', currSubset'\}$
8: $nextObj \leftarrow max\{g \in G \setminus currSubset''\}$
9: $currSubset \leftarrow currSubset''$
10: **else**
11: $nextObj \leftarrow max\{g \in G \setminus currSubset$ such that $g < max(currSubset)\}$
12: $currSubset \leftarrow currSubset \cup \{nextObj\}$
13: $currSubset \leftarrow \{g \in currSubset$ such that $nextObj \geq g\}$
14: **end while**

Comments in ASP are marked with `%`, as shown in lines 1-5. Line 8 is a solver (clingo) internal feature, enabling us to work in an incremental mode. In this mode, the program is made of three parts: the initial part, the guess part (lines 20-40), and the check part (lines 43 and 44). The guess and check parts depend of a constant k starting at 1 and incremented by 1 each time the check part fails. As it is not the subject of this introduction to ASP, it is intentionally glanced over. See [49] for a more in-depth tutorial of the incremental capabilities of the clingo solver.

The initialization step of the algorithm. Line 8 is a rule extracting from the input `rel/2` atoms the objects. The rule `object(G) :- rel(G,_)` reads *atom object(G) is true if G is the first element of a relation atom.* Two types of variables

```
% Implementation of Next Closure algorithm,
%  using the incremental mode of the clingo solver.
% INPUT: rel(X,Y): object X is in relation with attribute Y.
% OUTPUT: ext(N,X), int(N,Y): concept N has X in its extent
%                             and Y in its intent.
#include <incmode>.

object(G) :- rel(G,_).
attribute(M) :- rel(_,M).

% Initially, produce the concept whose intent is the whole set of attributes.
int(0,M) :- attribute(M).
ext(0,G) :- object(G) ; rel(G,M): int(0,M).

% Initialize the first and next step.
current(0,Max) :- Max = #max{O: object(O)}.
next(0,Max) :- Max = #max{O: object(O)}.


#program step(k).
% Produce the extent base.
current(k,X) :- not empty(k-1) ; derive2(k-1,X) ; next(k,T) ; T>=X.
current(k,X) :-     empty(k-1) ; current(k-1,X) ; next(k,T) ; T>=X.
current(k,X) :- next(k,X).  % add the next object

% Change of state.
next(k,T) :- not empty(k-1) ; T=#max{O: object(O), not derive2(k-1,O)}.
next(k,T) :-     empty(k-1) ; MaxCurrent = #max{P: current(k-1,P)} ;
             T = #max{O: object(O), not current(k-1,O), O<MaxCurrent}.

% Derivation operator.
derived(k,Y) :- attribute(Y) ; rel(X,Y): current(k,X).
derive2(k,X) :-    object(X) ; rel(X,Y): derived(k,Y).

% Newly derived objects must be greater than the next object.
empty(k) :- next(k,T) ; derive2(k,X) ; not current(k,X) ; X<T.

% Yield the concept at step k.
ext(k,O) :- not empty(k) ; derive2(k,O).
int(k,A) :- not empty(k) ; derived(k,A).

#program check(k).
:- query(k), not ext(k,O) ; object(O).

% Outputs.
#show ext/2.  #show int/2.
```

Encoding 1.1 – ASP implementation of Next Closure Algorithm

are used here: G is a variable in the mathematical sense, denoting here the first argument of the relation, and the argument of object. The underscore is a don't care variable denoting the second argument of `rel/2`. Each occurrence of G must have the same value but each occurrence of the don't care variable may have any value. Line 9 does the same for attributes. Note that these lines are only here for readability: next rules will make use of it, but there is no difference in using `rel(G,_)` instead of `object(G)`.

Line 12 defines the intent of the 0-th concept as the set of all attributes. This is a direct application of the very first line in Algorithm 1. The next line takes care of the extent of the concept. `ext(0,G) :- object(G) ; rel(G,M): int(0,M)` is a rule with a universal quantification reading *G belong to the extent of the 0-th concept if G is an object, and G is in relation with M for all M in the intent of the first concept*. Note the use of `:` (colon) to indicate the *for all* quantifier, whereas `;` (semicolon) stands for the *and* operator.

Together with the first concept, the initial values of the current subset and the next object to consider have to be computed (lines 2 and 3 of Algorithm 1, and lines 16 and 17 in the program). *currSubset* is a set of values, used as a primer to generate the intent, and in turn the extent and the concept ; whereas *nextObj* is a single object, used to populate *currSubset* at each step. Both will be modified during the algorithm execution. In ASP, no such things as collections exists, only atoms: at a given step k, there will be one and only one $next/2$ atom (corresponding to the *nextObj* value), whereas $current/2$ atoms may be many, one for each object present in the *currSubset* collection.

The function `#max` is acting on a set, forming a so-called aggregated value that can be compared to other values. Here, the function retains the maximal element of the set of objects O argument of `object/1`. It has to be equal to variable Max. Therefore, in line 16, rule `current(0,Max) :- Max = #max{O:object(O)}` reads as *Max is in the current set at step 0 if it equals the maximal element O found among the set of object(O) atoms.*

Variable *nextObj* is called *next* in the ASP encoding for brievity.

About Programs. Line 20, `#program step(k).` indicates that the following lines, up until the next program line (line 43), belong to the `step(k)` program.

This program, in the incremental setup, contains the new atoms specific to the k-th step. In that case, k is a parameter starting at 1, and going to whatever is the last step, as decided by the `check(k)` program we will explain later. In the absence of program indication, all atoms belong to program `base`. Both the initialization program (lines 8-17) and the input `rel/2` atoms belong to `base`, and are grounded at start, only one time. It can be understood as the step 0.

The position where a `#program` metaprogramming directive is written is important, since it acts on the *following* lines, up until the next program directive.

The division of encoding in programs enables the solver to ground and solve atoms of a specific program, one step at a time and with a specific value for each argument, instead of all of them in one go. The incremental setup is making advantage of this feature to divide the program into initialization, iteration, and stop condition.

The evolution of *currSubset*. Lines 22-24 are updating the value of *current* based on the previous iteration. A rule such as

`"current(k,X):- not empty(k-1); derive2(k-1,X); next(k,T); T>=X "`

reads *X belongs to current at step k if (1) previous step wasn't "empty", (2) X belongs to the extent of previous concept, (3) T is the next object, and (4) T is greater or equal to X*. The definition of `empty/1`, `next/2` and `derive2` will be given later. All these conditions are the translation of lines 9 and 13 of Algorithm 1, i.e. the transformation of *currSubset* when a concept has been yielded. Together with line 24, it implements the sequence of lines 9, 12 and 13 of Algorithm 1 defining *currSubset* at the next step.

Line 23 implements the exact same treatment, but when no concept has been yielded, i.e. line 13 of Algorithm 1. The difference lies in the use of `current/2` instead of `derive2`, because when no concept has been yielded, *currSubset* is not replaced by any concept extent, as per line 9 of Algorithm 1 ; the else clause (line 10) does not modifie the *currSubset* collection. Line 24 is the equivalent of line 12 in Algorithm 1: adding the next object to the current pool.

The evolution of *nextObj*. Just as *currSubset*, *nextObj* is modified at each step. Lines 27-29 of the ASP encoding are therefore taking care of lines 8 and 11

of Algorithm 1.
"`next(k,T):- not empty(k-1); T=#max{O:object(O), not derive2(k-1,O)}`" reads *the nextObj T at step k, if no concept has been yielded at previous step, equals to the maximal object O that does not belong to the extent of the yielded concept.* As in line 22, the extent of previously yielded concept is given by `derive2`. This line implements the line 8 of Algorithm 1. Note the use of the `,` (comma) as an *and* operator of higher priority than the `for all` quantifier. In this case, the two conditions apply to the same value O. The use of a semicolon would have applied `#max` on two sets of objects.

Lines 28 and 29 is the translation of the 11-th line of Algorithm 1. It reads "*the nextObj T at step k, if no concept has been yielded at previous step, equals the maximal object O that does not belong to previous step's $currSubset$, nor is smaller than $MaxCurrent$, the maximal element of the previous step's $currSubset$*".

Derivation operator. At some point, $currSubset'$ and $currSubset''$ have to be computed. This is the role of lines 32-33. Line 32, `derived(k,Y) :- attribute(Y); rel(X,Y): current(k,X)`, reads *Y is result of the derivation at step k if (1) it is an attribute, (2) it is in relation with object X, for any X in $currSubset$.* Line 33, `derive2(k,X) :- object(X) ; rel(X,Y): derived(k,Y)`, reads *X is doubly derived at step k if (1) it is an object, (2) it is connected to all attributes Y derived from $currSubset$.*

Deciding if a formal concept is to be yield. Line 36, `empty(k) :- next(k,T); derive2(k,X) ; not current(k,X) ; X<T`, reads *step k does not correspond to a formal concept if, T being the nextObj, an object X found in the extent but not in $currSubset$ is smaller than T.* This is the negation of the condition in line 5 of Algorithm 1. The presence of `empty/1` atom is the main indication for the next step to decide the value of $currSubset$ and $nextObj$.

Yielding the formal concept. Lines 39 and 40 are defining respectively the extent and the intent of the yielded concept. The two rules are quite simple: they only yield $ext/2$ and $int/2$ atoms if `empty/1` is absent, based on the derivation defined in lines 32-33.

Stopping the iterative process and showing the results. Line 43, similarly to line 20, declares that following rules belong to program `check(k)`, k being the step number. While the `step(k)` program role was to define the rules applied at each step to compute the formal concepts, this program's role is basically to stop the process when needed. In our case, Algorithm 1 has a very precise exit condition in line 4: *currSubset* contains all the objects, meaning that the infimum has been yielded during the previous loop.

This condition is implemented in line 44, as an integrity *constraint*, i.e. a rule that invalidates the model whenever it holds. More precisely, `:- query(k); not ext(k,O); object(O)` reads *the model is invalid if, at step k, an object O does not belong to the extent of the current concept.* The condition `query(k)` is used to ensure that the condition is only tested at step k `query(k)`, by convention of clingo's incremental mode, only holds at the current step. In default incremental mode as used here, clingo will ground and solve each step, and stop as soon as a valid model is found. With this constraint, all models found during the concept search are invalidated, up until the last. At this point, the solver stops the process, and outputs all the atoms found as true during the search for the satisfiable model.

In our case, those atoms include, among others, `ext/2` and `int/2` defined in lines 39 and 40. In order to keep the output simple, we specify to the solver to only provide us with these atoms, with the metaprogramming directive `#show`, as used in lines 48.

Results obtained with data in Figure 1.6. Together with the `rel/2` atoms, this encoding will yield the `ext/2` and `int/2` atoms describing the formal concepts, including for instance the 20-th step production `ext(20,5) ext(20,2) int(20,c) int(20,f) int(20,e)`, which can be written for humans as $\{2, 5\} \times \{c, e, f\}$. The complete output of the program is:

```
int(0,b) int(0,e) int(0,f) int(0,h) int(0,c) int(0,d) int(0,a) int(0,g) int(0,i)
ext(1,7) int(1,i) int(1,g) int(1,a) int(1,e) ext(3,5) int(3,g) int(3,c) int(3,f)
int(3,e) ext(6,4) int(6,g) int(6,a) int(6,c) int(6,e) ext(9,5) ext(9,4) int(9,g)
int(9,c) int(9,e) ext(12,3) int(12,d) int(12,h) int(12,e) ext(17,2) int(17,c)
int(17,f) int(17,e) int(17,b) ext(20,5) ext(20,2) int(20,c) int(20,f) int(20,e)
ext(23,5) ext(23,4) ext(23,2) int(23,c) int(23,e) ext(27,1) int(27,h) int(27,f)
```

```
int(27,e) int(27,b) ext(32,3) ext(32,1) int(32,h) int(32,e) ext(37,2) ext(37,1)
int(37,f) int(37,e) int(37,b) ext(40,5) ext(40,2) ext(40,1) int(40,f) int(40,e)
ext(44,7) ext(44,6) ext(44,5) ext(44,4) ext(44,3) ext(44,2) ext(44,1) int(44,e)
ext(8,7) ext(8,6) ext(8,4) int(8,g) int(8,a) int(8,e) ext(11,7) ext(11,6)
ext(11,5) ext(11,4) int(11,g) int(11,e).
```

Its standard representation is:
$\emptyset \times \{a,b,c,d,e,f,g,h,i\}$, $\{7\} \times \{a,e,g,i\}$, $\{5\} \times \{c,e,f,g\}$, $\{4\} \times \{a,c,e,g\}$, $\{4,6,7\} \times \{a,e,g\}$, $\{4,5\} \times \{c,e,g\}$, $\{4,5,6,7\} \times \{e,g\}$, $\{3\} \times \{d,e,h\}$, $\{2\} \times \{b,c,e,f\}$, $\{2,5\} \times \{c,e,f\}$, $\{2,4,5\} \times \{c,e\}$, $\{1\} \times \{b,e,f,h\}$, $\{1,3\} \times \{e,h\}$, $\{1,2\} \times \{b,e,f\}$, $\{1,2,5\} \times \{e,f\}$, $\{1,2,3,4,5,6,7\} \times \{e\}$.

Note the exact reproduction of Next Closure behavior, notably the order of generation mirroring the lexicographical order used by Next Closure.

Conclusion over the ASP code study. At this stage, the reader has been (1) learning the most important features of ASP thanks to a small ASP encoding, and (2) likely convinced that ASP is not better suited to implement algorithms, than imperative languages. This question is addressed in the next section, where the declarative power of ASP is used to specify the concept mining task, instead of trying to mimic an exact algorithm behavior.

1.5.3 Formal concept mining: declarative algorithm

The previous section described the ASP implementation of formal concept mining, in the specific case of Next Closure algorithm. However, the expressivity of ASP enables us to write the formal concept mining in a simpler declarative manner.

In the reproduced Next-Closure approach, the complexity came from mimicking the procedural approach, for which ASP is not suited for. Whereas, in declarative programming, one should instead describe the specification of the problem and its solutions (*what* instead of *how* to compute). This is exactly what is done in the following code, which is a direct translation of the equations of definition 1.2:

```
1 ext(O) :- rel(O,_) ; rel(O,A): int(A).
2 int(A) :- rel(_,A) ; rel(O,A): ext(O).
```

First line stands for *O belong to the extent if O is an object, and O is in relation with attribute A for all A in the intent*. The second line provides the symmetric treatment for the intent: *A belong to the intent if A is an attribute, and A is in relation with object O for all O in the extent*.

These two lines are enough for the solver to find all the formal concepts, since a fixed point of this program is a maximal set of objects and a maximal set of attributes in relation, i.e. a formal concept. Each fixed point is a solution model found by the solver. For instance, this program together with ASP encoding of Figure 1.6 leads to 16 models, each containing atoms describing a formal concept. Because of the non-predictability of the system exact behavior, one shouldn't expect the solver to yield formal concepts in the same order as in the Next Closure algorithm, nor in any particular order.

Note that ASP allows to stay very close to the original specification: in other logic programming approaches such as Prolog, this writing is not possible. A Prolog program would end up in an infinite loop, and one should devise a practical algorithm involving recursion and cuts to obtain the same results as ASP.

1.5.4 Comparing the implementations

Although it is possible to write a very compact code in ASP, one may wonder if it remains computationally tractable. We have compared the two ASP implementations of formal concept search through their running time, according to context size and density. The results are shown in figures 1.7, 1.8, 1.9 and 1.10.

The implementations are referred to as Next Closure, the ASP implementation explained in section 1.5.2, and Two-Liner, explained in section 1.5.3.

Figure 1.7 provides the time needed by each method to complete the full formal concept search, as a function of the context size. The number of concepts in the context is given for reference. Two different context densities are tested: 10% and 40%. Two-Liner clearly scales better than Next Closure: for given context sizes (8 to 21 objects), time grows exponentially for the latter, while it seems to remain constant for Two-Liner. Next Closure running time appears to be very dependent of the amount of concepts.

A closer look at Two-Liner behaviour shows that the same dependency on the

number of concepts exists, although at a much lower level (shown in Figure 1.8)

Another way to look at these data is to compute the average number of concepts found per second. The results as a function of context size are shown in Figure 1.9. As expected, Two-Liner outputs concepts quicker than Next Closure. However, as shown in Figure 1.10, when a context of density 0.4 reaches a size of 30, the performance of Two-Liner degrades. Peak performance appears later and smoother for lower density.

Compared to state of the art algorithms yielding millions of concept per second on vast contexts, the ASP approach is costly. For a real comparison, consider the benchmark proposed in Table 3 of [7], where the *internet ads* dataset (3279 objects and 1565 attributes) [34] is treated in less than a second by all different implementations of the Close-By-One algorithm. By running the fourth version of In-Close on our dataset, we got the full concept set in 0.1s. Our implementation of Next Closure in Python took 1 hour to yield around 25% of the concepts, and our implementation of Next Closure in ASP appears at least 700 times slower. However, using Two-Liner, it took about 73 seconds to get the full concept set. This indicates that a straightforward ASP encoding, while less efficient than carefully designed programs, remains competitive compared to early approaches in FCA.

Note that the enumeration of models needs two steps: the expansion of the encoding to a set of formulas (grounding), and the search for fixed points (solving). For the *internet ads* dataset, the grounding of Two-Liner takes more than 75% of the total time and 160 Mo of formulas. The remaining 20 seconds are used by the solver to enumerate the formal concepts from the formulas. This indicates that, even with parallelization of the solving part, Two-Liner does not scale, since it is first limited by the exponentially growing amount of data. For the Next Closure implementation in ASP, the incremental mode requires each step to be grounded then solved individually, and the management costs of this mode largely exceeds the gain of quicker grounding steps.

The real interest of ASP is in a research context to easily design and tests new ideas in FCA. In other words, ASP is well-suited to encode directly mathematical relations, and takes into account known properties on this formal setting.

From one side, this allows to check it on non trivial examples. On the other side, the associated algorithmic complexity may be difficult to estimate since it

```
1 { ext(O): rel(O,_) }.
2 int(A) :- rel(_,A) ; rel(O,A): ext(O).
3 #show int/1.
```

Encoding 1.2 – Mining intents with ASP

depends on two complex programs, the grounder and the solver.

1.5.5 Optimizations of encodings

It is also possible to experiment optimizations to handle larger data. In the case of Two-Liner, the grounding is the costliest step. For instance, let us define the ASP encoding 1.2 that enumerates the intents.

The first line is enumerating all subsets of the set of objects using a cardinality expression. The second line is computing the intent associated with the set of chosen objects. This enumeration of intents is however not unique, since subsets of objects are not necessarily maximal extent. It is however possible to hint the solver that only distinct solutions must be returned. Using the command-line option `-project=show`, the clingo solver collapses together all models showing the same atoms, as defined in `int/1` atoms in the third line.

Solving that program with the *internet ads* dataset takes 48 seconds, while the grounding only takes around 24 seconds and yield 82 Mo of formulas. Although it only computes the intent sets, it is easy to ground the `int/1` with the extent generation (first of the Two-Liner) to get the full concepts, or use a dedicated program. In the end, the concepts are obtained using less memory, shifting the time/memory tradeoff compared to the Two-Liner implementation.

1.5.6 A second example: a first approach to Relational Concept Analysis

Relational Concept Analysis is an extension of Formal Concept Analysis which treats multi-relational datasets, i.e. with different contexts and relations between the objects of these contexts [92].

The input is a *Relational Context Family* (K, R), where K is a set of formal

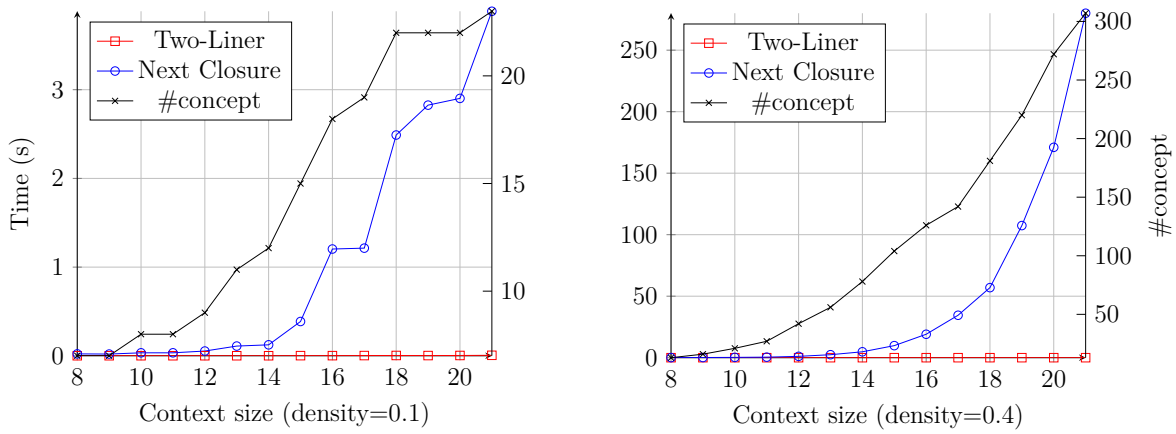


Figure 1.7 – Comparison of the ASP Next Closure and Two-Liner implementations for the concept search. Measure is overall time needed by the solver to enumerate all concepts from random formal contexts of different size. The black lines show the number of formal concept in the contexts. *Left:* the context has a density of 0.1. *Right:* the context has a density of 0.4.

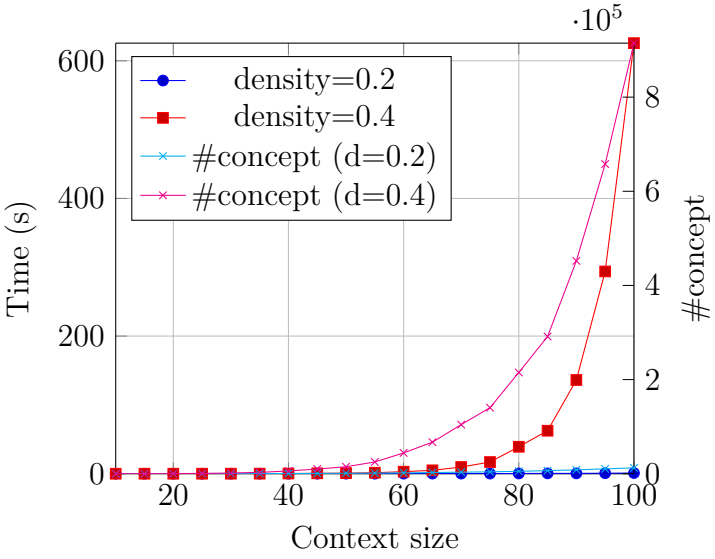


Figure 1.8 – Efficiency of the ASP Two-Liner implementation for the concept search, depending on context size and density. Measure is overall time needed by the solver to enumerate all concepts.

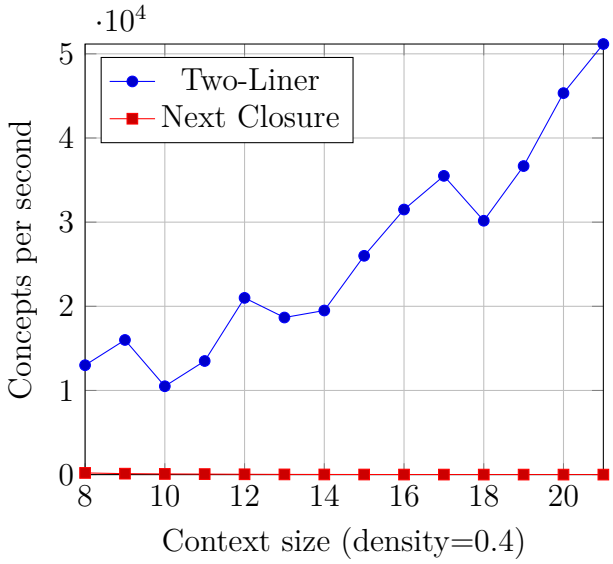


Figure 1.9 – Comparison of the rate of concepts per second, measured for the two ASP implementations of concept search accross different context sizes with a density of 0.4. Next Closure appears to get a very low output compared to Two-Liner. The efficiency of Two-Liner improving with context size is an artifact due to small context sizes, as shown in Figure 1.10.

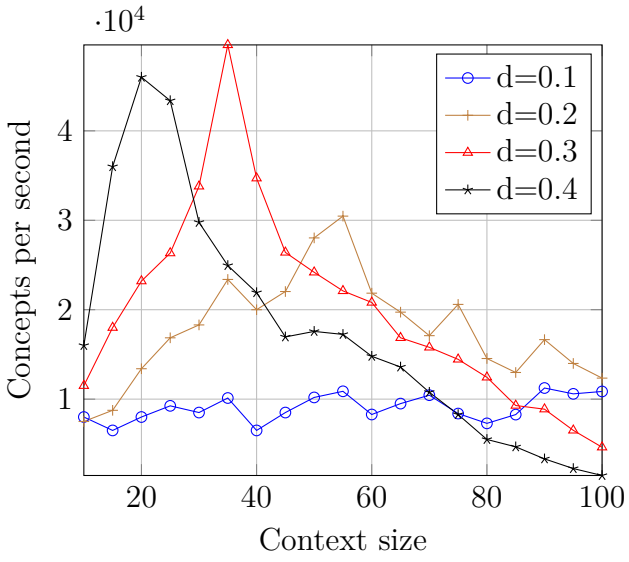


Figure 1.10 – Comparison of the average number of concepts yielded by second using the Two-Liner method, depending on context size, compared across different context densities.

```
% Define constants min and max used in lasts operators.
#const min=3.
#const max=42.
% Universal scaling:
rso(X,C):- object(X); concept(C); ext(C,V):rel(X,V).
% Strict Universal scaling:
rso(X,C):- object(X); concept(C); ext(C,V):rel(X,V) ; rel(X,_).
% Existential scaling:
rso(X,C):- object(X); concept(C); rel(X,V) ; ext(C,V).
% Cardinality restriction (min) scaling:
rso(X,C):- object(X); concept(C); min{rel(X,_)}.
% Cardinality restriction (max) scaling:
rso(X,C):- object(X); concept(C); {rel(X,_)}max.
% Cardinality restriction (min & max) scaling:
rso(X,C):- object(X); concept(C); min{rel(X,_)}max.
% Qualified Cardinality restriction (min) scaling:
rso(X,C):- object(X); concept(C); ext(C,V):rel(X,V); min{rel(X,_)}.
% Qualified Cardinality restriction (max) scaling:
rso(X,C):- object(X); concept(C); ext(C,V):rel(X,V); {rel(X,_)}max.
% Qualified Cardinality restriction (min & max) scaling
rso(X,C):- object(X); concept(C); ext(C,V):rel(X,V); min{rel(X,_)}max.
```

Encoding 1.3 – ASP encoding of various Scaling Operators in the context of RCA

contexts, each one describing a binary relation between a set of objects and a set of attributes, and R is a set of *relation*s, i.e. a binary relation between the objects of a context K_i and the objects of a context K_j. The relational concepts are defined and processed iteratively. Formal contexts are pair-wise distinct regarding their objects, i.e. to a set of objects corresponds one and only one context at each step of the process.

At each step of the process, each formal context $K_i \in K$ is *scaled*, i.e. is augmented with additional attributes indicating relations of various kinds with other formal contexts. The kind of relations depends of the scaling operator used.

The final output is, for each formal context $K_i \in K$, a relational concept lattice L_i composed of the initial concept lattice, expanded with additional concepts and attributes. It preserves the initial lattice as a sub-order [92].

Scaling operators. For this introduction, we will present in encoding 1.3 the ASP implementation of the most common scaling operators, presented in Table 7 of [92] along their equivalent constraint on the incidence matrix of the relational

extension context I^+. Each of them, and combinations thereof, are reproduced on odd lines, except lines 1-3 that introduces *min* and *max* as constants in the ASP program, with their default values. Some operators relying on cardinality will make use of them.

Predicate `rso/2` is used to encode the Result of the Scaling Operator, applied on a context $K = (G, M, I)$ and a relation $rel/2$, where G is the domain of r, G_r the range of r, and $K_r = (G_r, M_r, I_r)$ is another context from which the concepts C_r are derived. Thus, predicate `object/1` encodes the set of objects G on which relations are added. Predicate `concept/1` encodes the identifiers of concepts C_r, i.e. the new attributes in the relational scaling of K. The remaining conditions in each rule implement the scaling operator constraints on I^+. For instance the universal scaling operator, where the object image $r(X)$ must be completely included in the extent of C in order for X to get the relational attribute $\forall\ r : C$. This is implemented with the condition `ext(C,V): rel(X,V)` reading "*V must belong to the extent of C for all V in relation with X*". Note that the condition holds even if $r(X)$ is empty, hence linking X to C despite the absence of any common object. To avoid that behavior, the universal strict operator was introduced in [92], requiring that $r(X) \neq \emptyset$. In ASP, it is implemented by ensuring the existence of an atom `rel(X,_)`, encoding a relation between X and anything. The complete implementation of the universal strict operator is found in line 7.

The remaining operators (lines 11-21) are introducing conditions on the cardinality of $r(X)$. In ASP, this is implemented by using a cardinality expression `N { A } M`, which ensures that the number of atoms matched by `A` is bounded by `N` and `M` (included). If omitted, the default lower and upper bounds are 0 and infinity. Hence, `n { rel(X,_) }` reads "*X is linked to at least n attributes*", thus implementing the cardinality restriction (min) scaling. The other operators are setting an upper bound, and in case of the qualified cardinality restriction (min & max) scaling, ensuring both that $r(X) \subseteq Ext(C)$ and $n \leq |r(X)| \leq m$.

Fixed-point and incremental algorithms. As shown in the previous sections where Next Closure algorithm and definition-derived implementations were proposed and compared, ASP is more suited to fixed point formulations rather than explicit descriptions of an enumeration method. To our knowledge, the com-

plete relational extension of a RCF has only been defined in an iterative way. No elegant encoding like the Two-Liner implementation can be devised. Moreover, the incremental mode of the clingo solver is not powerful enough for the computation of the complete relational extension of an RCF: the concept search is a task to be solved for each range of a relation, at each step, and this would require an external call to a program to handle it properly.

1.5.7 Continuing the implementations for FCA

This section addressed the ASP encoding of formal and relational concept enumeration. Those two examples show the strengths and limits of ASP for this task.

Many other objects of the Formal Concept Analysis framework can be encoded in Answer Set Programming, and this will be further discussed in chapter 5, which makes use of FCA to introduce an ASP-based drawing engine that incidentally happens to be a tour of some of FCA most popular objects.

1.6 Thesis contributions and plan

Contributions

First, we formulate and study Power Graph Analysis in the framework of Formal Concept Analysis. The goal is to be able to benefit from the advances in this domain to get a better view of the structure of the search space, suggesting variants and solving approaches, and conversely, to offer a playground for new studies in FCA.

Second, we propose an FCA-based generalisation of Power Graph Analysis relying on a single, more general graph motif. This extension to formal concepts, called triplet concepts, allows to manage together cliques and bicliques.

Third, we state the Formal Concept Analysis modeling problem in a logical setting, Answer Set Programming (ASP). The first proposition of ASP program for FCA seems due to CV Damásio and published only in [58], where the focus is on developing expressive query languages for formal contexts. A more recent study extends the search for n-adic FCA [95] and is focused on the issue of filtering large concept spaces by checking additional membership queries. In our case, the

particular task of graph compression needs heavier calculations and a specific code since the whole concept space needs to be explored.

Fourth, we have tested our code on several real biological networks, and explored the datasets in the light of the triplet concepts.

Finally, we present Biseau, a general-purpose ASP environment for graph theory, that we apply on FCA in order to illustrate the capabilities of the software.

Plan

Chapter 2 introduces the link between Power Graph Analysis and Formal Concept Analysis, and formulates the limits of this approach, one of them being the concurrent treatments of cliques and bicliques. Chapter 3 exposes a new theoretical concept enabling to handle the two motifs as a special case of a more general motif, namely the overlapping biclique. Chapter 4 is a tour of biological data applications specifically realized in that thesis. Chapter 5 presents various ASP encodings of FCA objects and extensions as an excuse to demonstrate the capabilities of Biseau, an ASP framework targeting the specification of graphs structures. Finally, chapter 6 concludes, and offers some perspectives to conclude this work.

Chapter 2

PowerGraph compression with formal concepts

> Mathematics, in an earlier view, is the science of space and quantity; in a later view, it is the science of pattern and deductive structure.
>
> Philip J. Davis, Reuben Hersh
> The mathematical experience (1981)

The previous chapter presented Formal Concept Analysis and Power Graph Analysis. This chapter aims at formalizing the search space of Power Graph Analysis using Formal Concept Analysis.

Royer et al. introduced in 2008 Power Graph Analysis [94], an efficient software using classes of nodes with similar properties and classes of edges linking node classes to achieve a lossless graph compression. The contributions of this chapter are twofold. First, we formulate and study this issue in the framework of Formal Concept Analysis. Our aim is to better understand the search space explored by this program. This leads to a generalized view of the initial problem offering new variants and solving approaches. Second, we state the FCA modeling problem in a logical setting, Answer Set programming, which allows highly modular developments and provides a great flexibility for explaining various specification of concept search spaces.

This chapter is organized as follows. First the *graph contexts* objects are defined in section 2.1, and section 2.2 uses them to encode Power Graph motif search in

the graph. FCA-based Power Graph Analysis is then presented in section 2.3, and a representation of its search space and associated heuristics in section 2.4. Two examples of powergraph compression and search space exploration are detailed in section 2.5. Then, using concept lattices, section 2.6 shows how to simplify the search space.

Section 2.7 details some optimizations of our Power Graph Analysis implementation regarding the compression of specific motifs and overall memory and time management. Then, section 2.8 addresses the limits of greedy approaches for compression, showing that some kind of complex motifs cannot be efficiently compressed as a Power Graph using only maximal motifs.

The chapter finishes in section 2.9 with the introduction of the need for unification of Power Graph motifs in the context of graph context-based compression, thus motivating the work on *triplet concepts* presented in chapter 3.

2.1 Graph contexts to encode undirected graphs

A *graph context* is a formal context (X, Y, I) where the set of objects X and attributes Y are subsets of vertices in an undirected graph $G = (V, E)$ and the binary relation $I \subseteq X \times Y$ represents its edges E. Usually, formal contexts are defined on disjointed sets X and Y. In graph contexts objects and attributes represent the same kind of elements and can intersect. For an undirected graph, the graph context is symmetric.

G. Chiaselotti and T. Gentile [27] have proposed to take $X = Y = V$ for a graph context. This corresponds to a standard representation of a graph by its (symmetrical) adjacency matrix. Note however that for a same graph, there may exist several interesting smaller representations trying to minimize the intersection of X and Y. For instance, a bipartite graph can be defined by a bipartition of disjointed sets X and Y. This can be important from a practical perspective since a number of treatments depend on the size of the matrix.

The authors have shown that for simple undirected graphs (no loop, no multiple edges), the standard application of FCA leads to a coincident derivation operator for object and attributes: the derivative of a subset of vertices X in terms of graph is the neighborhood intersection of all its elements. It is equivalently the set of

	1	2	3	4
1		×	×	×
2	×		×	×
3	×	×		
4	×	×		

	1	2	3	4
1	×	×	×	×
2	×	×	×	×
3	×	×	×	
4	×	×		×

Figure 2.1 – *Left:* A small graph context on the set $S = \{a, b, c, d\}$. *Right:* its reflexive version.

vertices whose neighborhood includes X. Since graphs are simple, the derivative of a subset of vertices X has no intersection with X. Thus concepts are pairs of disjointed sets. The concept lattice is by construction self-dual. See example 2.1 for an illustration on Figure 2.1.

Example 2.1. We consider the small graph context in Figure 2.1. With the classical definition proposed by Chiaselotti et al. and apart from the top and bottom concepts, concepts are $C_1 = \{1\} \times \{2, 3, 4\}$, $C_2 = \{2\} \times \{1, 3, 4\}$, and $C_3 = \{1, 2\} \times \{3, 4\}$, duplicated with their dual concepts $C'_1 = \{2, 3, 4\} \times \{1\}$, $C'_2 = \{1, 3, 4\} \times \{2\}$, and $C'_3 = \{3, 4\} \times \{1, 2\}$. C_1 and C_2 are linked to C_3 in the lattice an dually C'_3 to C'_1 and C'_2.

For graphs with loops or even for simple graphs, if we not do not worry about fictitious reflexive edges, it is interesting to accept concepts that consider pairs of possibly intersecting sets. We study in this thesis the case of simple graphs. Therefore, the reflexive edges, represented by the diagonal in the graph context, do not carry any information on the input graph.

Let us define the *reflexive graph context*, a special kind of graph context, where the diagonal is filled, as illustrated in Figure 2.1 (right). This object, as shown in example 2.2, does not yield the same formal concepts as its non-reflexive counterpart.

Example 2.2. Let us again consider the small graph context in Figure 2.1, already presented in example 2.1, this time in its reflexive version, where the diagonal is set to 1. Applying standard FCA results, symmetry aside, in the following concepts: $C_4 = \{1,2,3\} \times \{1,2,3\}$, $C_5 = \{1,2,4\} \times \{1,2,4\}$ and $C_6 = \{1,2\} \times \{1,2,3,4\}$. It appears that C_4 and C_5 are not maximal and C_6 is the sole maximal concept.
As a foretaste of developments that will appear in the next chapter, one can note that since reflexive links are fictitious, equivalent concepts could be represented by the following equations that avoid duplicated edges: $C_4 = \{1,2\} \times \{2,3\}$, $C_5 = \{1,2\} \times \{2,4\}$ and $C_6 = \{1,2\} \times \{2,3,4\}$.

In this chapter, both non-reflexive and reflexive graph contexts will be used in section 2.2 to encode motif search in the graph. This will ultimately lead to an encoding of Power Graph Analysis as a set of concepts, as shown in section 2.3 and following sections of this chapter.

2.2 Graph motifs as formal concepts

In Power Graph Analysis, the two main motifs used for graph compression are cliques and bicliques. This section shows how these motifs relate to graph contexts, and more generally FCA. Next sections elaborate on these notions to perform the powergraph compression.

2.2.1 Bicliques as formal concepts

In a non-reflexive graph context, there is a one-to-one mapping between maximal bicliques and formal concepts. On the other hand, in a reflexive graph context, the number of concepts describing a biclique (A, B) increases linearly with its size: due to the diagonal, an object-concept $\{a\} \times \{a'\}$ exists for each element x in $A \cup B$. Counting the biclique (A, B) itself and the fact that each concept (X, Y) has a symmetric (Y, X), the number N of formal concepts for the biclique context is therefore $N = 2 \times (|A| + |B| + 1)$.

	a	b	c	d
a			×	×
b			×	×
c	×	×		
d	×	×		

	a	b	c	d
a	×		×	×
b		×	×	×
c	×	×	×	
d	×	×		×

	a	b	c	d
a		×	×	×
b	×		×	×
c	×	×		×
d	×	×	×	

Figure 2.2 – *Left:* A small graph context on the set $S = \{a, b, c, d\}$ describing a biclique $\{a, b\} \times \{c, d\}$. *Middle:* its reflexive variation, which cannot be compressed efficiently in Power Graph. *Right:* a clique in a non-reflexive graph context, issuing 16 formal concepts.

Example 2.3. The context on the left of Figure 2.2 describes a single biclique and consequently contains only one formal concept, $(\{a, b\}, \{c, d\})$. If the relation becomes reflexive (middle of Figure 2.2), it issues 4 more formal concepts: $(\{a\}, \{a, c, d\})$, $(\{b\}, \{b, c, d\})$, $(\{c\}, \{a, b, c\})$, $(\{d\}, \{a, b, d\})$ and $(\{b, d\}, \{b, d\})$. Note that, symmetry put aside, there is exactly one new formal concept per node, since each of them can be considered as the hub of a star.

2.2.2 Cliques as formal concepts

Cliques can only be generated from concepts in the *reflexive* graph context, since a clique requires overlapping extent and intent. For any formal concept (X, Y), $X \cap Y$ describes a clique or the empty set. In principle it could be possible to reconstruct a clique from the formal concepts they yield in the non-reflexive graph context, but no concept includes the whole clique and a clique can be composed of an exponential number of concepts. Indeed, the maximal number of concepts of a context is achieved by a fully filled context, except one element per line (all different). In such a case, all subsets of elements can appear in the object or attribute part of a concept and the number of concepts is 2^n. This corresponds to the clique case, which is a complete context. In a reflexive graph context however, all these formal concepts will collapse. For reference, another worst-case is presented on the center context of Figure 2.2: each object is unique, but this time a clique cannot express the whole dataset.

From this analysis, it follows that using a reflexive or non-reflexive graph context impacts heavily the search for bicliques and cliques. Bicliques are easily searched in a non-reflexive graph context, but produce additional formal concepts in a reflexive graph context. Cliques, on the other hand, generate an exponential number of concepts in the non-reflexive graph context, but are represented compactly in the reflexive version of the same context.

One way to avoid an explosion of concepts when looking for both bicliques and cliques is to consider both the reflexive and non-reflexive graph context, and the formal concepts issued from them. The formal concepts found in the reflexive graph context are of interest when extent and intent overlaps, i.e. describe a clique. Then, the search in the non-reflexive graph context may ignore any formal concept covered by previously found cliques: this suggests that bipartite graphs, and more generally triangle-free graphs, do not need reflexive structures.

2.2.3 Other graph motifs

Stars

A star is a special case of biclique, where one set is a singleton. It is a common motif in graphs, typically associating a particular node with a large quantity of other elements. In protein-protein interaction networks for instance, a so-called hub protein interacts with many others and would create a large star motif. This effect will appear clearly in applications described in chapter 4. In social networks, group leaders are often found as hub node in stars [3].

From a concept point of view, a maximal star motif centered on node s is expressed by the object concept $\{s, a_1, a_2, ..., a_n\} \times \{b_1, b_2, ..., b_n\}$ (see section 1.3.1 for a definition), with b_i being the nodes in relation with s and a_i being nodes having the exact same neighbors as s. If the context is not clarified, a_i nodes may exists and there is another maximal star for each of them. In such a case, the biclique described by the concept covers more edges than the distinct stars. The object and attribute concepts therefore indicate the existence of maximal stars, although some of them may be grouped together in a more general biclique motif.

Quasi-motifs

Quasi-cliques and quasi-bicliques are both defined in literature as cliques and bicliques that are not complete but remain dense. The rationale behind these structures is that maximal motifs are sensible to missing data and noise, motivating the need for a repair process to detect falsely missing or supplementary edges [110], or to a relaxed search of quasi-motifs, where some edges are missing. The exact definition of these quasi-motifs is however complex, because one wants to avoid an almost empty quasi-motif. For quasi-bicliques, [74] shows the interest of simple constraints to define balanced quasi-bicliques, i.e. bicliques with missing edges but without poorly connected nodes. Authors introduce μ-tolerance maximal quasi-bicliques, using parameter μ as a threshold defining both the minimum number of edges present and the maximum number of edges absent between the sets of the biclique. One of the main property of μ-tolerance maximal quasi-bicliques is that they generally allow to collapse many maximal bicliques together.

From a concept point of view, the exploration of quasi-bicliques has already been formalized using fuzzy contexts [46].

This thesis does not address the use of quasi-motifs in Power Graph Analysis. It would however be an interesting extension to replace searched motifs by quasi-motifs counterpart, and to highlight missing edges in order to keep a lossless compression, while obtaining an higher edge compression.

2.3 Compression with formal concepts

This section introduces definitions needed for the characterization of the Power Graph Analysis search space. We recall that its input data structure is an undirected graph where self-loops are ignored. In this method each compression step will consist into (1) the discovery of a graph motif (biclique or clique), and (2) the corresponding refinement of the powergraph. We address here the second item.

Powernodes are the result of a hierarchical aggregation of nodes and other powernodes. Poweredges aggregate the edges of a biclique between the nodes of two powernodes. When a poweredge links a powernode to itself, it describes a clique.

We will use the word *motif* to describe concepts, i.e. cliques or bicliques, and the infix parenthesized notation for trees (e.g. $T = (a(b(c, d), e))$ for a tree of root a having children b and e, and internal node b having children c and d).

2.3.1 Powergraph

A powergraph is a compressed graph that can be represented as two graphs: the poweredges, the hierarchy of powernodes, and fulfill three conditions with respect to the initial graph it represents.

Definition 2.1. *Powergraph* A powergraph $\mathcal{G} = (PN, PE, H)$ is a pair of graphs on a set of nodes PN. (PN, PE) is the poweredge graph and (PN, H) is a tree called hierarchy graph. Elements of PN are called powernodes, and elements of PE are called poweredges. $\mathcal{G}$ is a powergraph of a graph $G = (V, E)$ if and only if:

1. *powernode hierarchy condition:* there is a one-to-one mapping between terminal nodes of H and nodes in V, and PN is a hierarchy of subsets of V that is described by H. If p is an element of PN, the subset of V is denoted $nodes(p)$
2. *poweredge decomposition condition:*PE is a partition of E. For any e in PE, $edges(E)$ denotes the corresponding subset of E
3. *subgraph condition:* a poweredge between a powernode P_1 and a powernode P_2 exists whenever all elements of P_1 are connected to all elements of P_2.

Any graph $G = (V, E)$ can be represented as a powergraph $\mathcal{P} = (V, E, H)$ by considering a trivial flat hierarchy H, made of the set of singletons for each element of V. In Power Graph Analysis, a singleton powernode and the node itself will be generally confused. Powergraph compression is a process starting from a graph and producing a powergraph with special properties. The whole problem of Power Graph Analysis is, for a graph G, to decide a powernode hierarchy and a set of poweredges that fulfill the constraints defined in definition 2.1 (subgraph, powernode hierarchy, and poweredge decomposition conditions) while minimizing the size of PE.

2.3.2 Powernode hierarchy

Let us now define some operators on the powernode hierarchy, in order to simplify future definitions.

Definition 2.2. *Powergraph hierarchy operators* Let $\mathcal{P} = (PN, PE, H)$ be a powergraph of a graph $\mathcal{G} = (V, E)$. H being a tree, one can define:

— $\uparrow n$ is the parent powernode of n in H.

— $\mathcal{N}(n)$ is the neighborhood of powernode n in poweredge graph (PN, PE).

— $nodes(p)$ is the mapping of p to the subset of V it represents, or conversely given a subset S of V, $\mathcal{PN}(S)$ is the powernode representing S or $\emptyset$ if it does not exist.

Definition 2.3. *Mergeable powernodes* Let $\mathcal{P} = (PN, PE, H)$ be a powergraph and $p, q \in PN$, $p \neq q$. p and q are mergeable if $\uparrow p = \uparrow q$ and if $\mathcal{N}(p) \cap \mathcal{N}(q) \neq \emptyset$.

Definition 2.4. *Maximal powernode* Let $\mathcal{P} = (PN, PE, H)$ be a powergraph and and $p \in PN$. p is maximal if it cannot be merged with another powernode.

The existence of mergeable powernodes in a powergraph indicates that the compression ratio can be improved: it is possible to place two powernodes p and q in a new parent powernode if they have the same parent. The process is only possible if this new parent powernode can be the endpoint of a poweredge that would replace one or more poweredges linking p and q.

2.3.3 Compression of a powergraph

As described in section 1.2, the construction of a fully compressed powergraph is iterative. At each step is performed a *motif compression*: a biclique (or clique) is chosen, and its associated edges are replaced by poweredges and associated nodes grouped under (possibly existing) powernodes. The hierarchy of the graph is also updated to reflect the apparition of the new powernodes. In order to have a safe compression, the modifications of powernodes, poweredges and powernode hierarchy must fulfill the Power Graph conditions as per definition 2.1.

The process finishes whenever no remaining compressible motifs are found. The compressed graph resulting from a motif compression is defined in definition 2.6.

Definition 2.5. *Compressible Motif* Let $\mathcal{P} = (PN, PE, H)$ be a powergraph, $M = (A, B)$ a biclique, and $\mathcal{P}' = (PN', PE', H')$ the powergraph resulting from the compression of M in $\mathcal{P}$, such that $A \times B \subseteq PE$. If $PE \neq PE'$, then M is compressible in $\mathcal{P}$. M is perfectly compressible if PE' has exactly one more poweredge than PE ($|PE'| = |PE| + 1$).

Note that a compressible motif may leave the hierarchy unchanged because its sole effect is to add poweredges between existing powernodes. The previous definition only retain motifs that are useful in the sense that they allow the number of poweredges to decrease.

Definition 2.6. *Motif compression* Let $\mathcal{P} = (PN, PE, H)$ be a powergraph and $M = (A, B)$ a compressible motif in $\mathcal{P}$. $\mathcal{P}' = (PN', PE', H')$ is the result of the compression of M in $\mathcal{P}$, if: $PN' - PN$ is the set of new powernodes grouping the nodes $A \cup B$, $PE' - PE$ is the set of new poweredges covering $A \times B$, and H' is the new powernode hierarchy that includes new powernodes $PN' \setminus PN$ as roots or intermediate nodes.

An implementation of Power Graph Analysis has to ensure that, knowing a motif to compress and a powergraph, a valid powergraph is produced.

Description of Motif Compression (Algorithm 2)

The goal of this algorithm is to determine the new powernodes and poweredges that are produced by the compression of a motif in a graph. Functions in upper case are quickly described here.

Function *siblings_partition()* produces a partition of the set of nodes, grouping siblings in the hierarchy H. Function *simplified()* takes as input a partition, and replaces each part by its powernode if it exists. This function is called repeatedly along with *siblings_partition*, so that the powernodes newly added by *simplified()* are grouped with their siblings, thus handling the recursive structure of the powernode hierarchy. Function *simplifiable()* fails when no siblings can be replaced by a powernode.

The overall output of the algorithm, PN^+, is made of sets of powernodes to group in new powernodes, and PE^+ the set of new poweredges described as pair

of powernodes. The modification of the powernode hierarchy H consists in the introduction of new powernodes as new roots or intermediate nodes in the trees.

Algorithm 2 Motif Compression: determination of the powernodes and poweredges introduced by the compression of a motif M on a graph $\mathcal{P}$.

Require: Powergraph $\mathcal{P} = (PN, PE, H)$, motif $M = (A, B)$
Ensure: Compute the new powernodes PN^+ and poweredges PE^+ to add to $\mathcal{P}$

```
PN_A^+ ← siblings_partition(A)
PN_B^+ ← siblings_partition(B)
while simplifiable(PN_A^+) do
    PN_A^+ ← siblings_partition(⋃simplified(PN_A^+))
end while
while simplifiable(PN_B^+) do
    PN_B^+ ← siblings_partition(⋃simplified(PN_B^+))
end while
PN^+ ← PN_A^+ ∪ PN_B^+
PE^+ ← PN_A^+ × PN_B^+
```

Next section will address the choice part of Power Graph Analysis, since current definitions only provides vocabulary and constraints on a powergraph, but little insight on which motif should be compressed in order to obtain a readable graph compression of the input graph.

2.4 Search spaces of Power Graph Analysis

Previous section defined how a powergraph is (iteratively) compressed using motifs such as cliques and bicliques, until achieving a fully compressed powergraph. This section covers the combinatorial aspect of this task, recalls the most important optimization metric, and proposes two possible search spaces.

2.4.1 Edge reduction optimization

In its initial design, Power Graph Analysis was introduced with two metrics representing the overall quality of compression.

The first is *edge reduction*, that measures the compression ratio with respect to edge number. The second measure is *conversion rate*, measuring the mean reduction of edges per powernode introduced. The higher the conversion rate, the larger the motifs covered by poweredges are. Conversion rate is therefore dependent of edge reduction, average motifs size, and of the presence of stars and cliques (that can be compressed with only one powernode and one poweredge).

Definition 2.7. *Quality metrics in Power Graph Analysis* Let $\mathcal{P} = (PN, PE, H)$ be the powergraph of a graph $\mathcal{G} = (V, E)$.

$$\text{edge reduction} = \frac{|E| - |PE|}{|E|}$$

$$\text{conversion rate} = \frac{|E| - |PE|}{|PN \setminus V|}$$

Power Graph Analysis is an optimization program using edge reduction as its objective function.

Example 2.4. Considering a graph of four elements $V = \{a, b, c, d\}$ reduced to a biclique $B = (\{a, b\}, \{c, d\})$. The initial graph has 4 edges and 4 nodes. The powergraph with the biclique compression has 4 nodes, 2 powernodes and 1 poweredge. The compression thus achieve a 75% *edge reduction* and a *conversion rate* of 1.5.

Let us now define the concept-based search space for this task. A first approach to the problem is to first select concepts, then perform their compression. This approach is presented in section 2.4.2. The second one is to select poweredges from concepts, then do the compression. This approach is described in section 2.4.3.

The determination of the optimal powergraph, i.e. the powergraph with maximal edge reduction, is recognized as an NP-complete problem [37], and as such requires heuristics to explore the search space. This is the subject of section 2.6.1.

2.4.2 Selection-transformation method search space

This method consists into the compression of the concepts in a chosen order, so that all necessary powernodes and poweredges are added to the graph, as described in section 2.3.3.

Let $\mathcal{P} = (PN, PE, H)$ be a powergraph, and $C = \{c_1, ..., c_k\}$ be the available concepts to compress. Each permutation p of $|C| \geq k > 0$ concepts of C describes a compression of $\mathcal{P}$ and an edge reduction score. The search space of this method is therefore the permutation space over subsets of concepts covering the input graph. Note that appending a concept c to a permutation p will increase the edge reduction only if c is compressible in $\mathcal{P}$ after the compression of p.

Because the search space is the permutation space of formal concepts, an optimal compression cannot be searched in practice. It is therefore necessary to devise an approximation enabling to find a good level of edge compression without exploring the complete set of permutations. This is presented in section 2.6.1.

2.4.3 Iterative concept determination method search space

As described in section 2.3.3, compressing one concept after another may need the creation of more than one poweredge, since the second concept is no longer compressible as-is. Each of these poweredges form the *decomposition* of the concept. This approach considers that each poweredge of this decomposition is a separate concept. In other words, the set of concepts selected for compression is changing at each compression step. The overall compression method is shown in Algorithm 3, where most implementation details are abstracted to highlight the most important algorithmic aspects.

Definition 2.8. *Independent Concept* Let $\mathcal{G} = (PN, PE, H)$ be a powergraph, $C = \{c_1, ..., c_k\}$ be the available concepts to compress, and c_1 the next concept to compress. The set $\mathcal{I}$ of concepts independent from c_1 in $\mathcal{G}$ is defined as:

$$\mathcal{I}(c_1, C, \mathcal{G}) \quad \Leftrightarrow \quad \{c \in C \quad | \, \mathcal{N}(C) \cap \mathcal{N}(c_1) = \emptyset\}$$

Algorithm 3 Step-by-step Power Graph Compression: the compression algorithm implemented by Power Graph Analysis and *PowerGrASP*. Implementations of *best_concept()* is typically selecting the largest concept. *compress()* takes care of output data. *independent_concepts()* returns the set of concepts that share no node with the compressed concept. *decomposed_concepts()* ensures that concepts that are overlapping with c are divided into smaller concepts in order to respect the hierarchy condition of powernodes as described in section 1.2.1.

Require: Graph $\mathcal{G} = (V, E)$, concepts $\mathcal{C}_\prime = \{c_1, c_2, ..., c_n\}$
Ensure: Computation of a Power Graph compression
1: $i \leftarrow 0$
2: **while** $A \subset \mathcal{N}$ **do**
3: $c \leftarrow$ best_concept(C_{i-1})
4: $E \leftarrow E \setminus$ edges_of(c)
5: $COMPRESS(c)$
6: $I \leftarrow$ independent_concepts(C_{i-1}, c)
7: $B \leftarrow$ decomposed_concepts$(C_{i-1} \setminus I, c)$
8: $C_i \leftarrow I \cup B$
9: $i \leftarrow i + 1$
10: **end while**

Definition 2.9. *Concept Decomposition* Let $\mathcal{G} = (V, E, PN, PE, H)$ be a compressible graph, $C = \{c_1, ..., c_k\}$ be the available concepts to compress, and c_1 the next motif to compress. The decomposition of c_1 is the set of pairs of nodes describing the poweredges and powernodes constituting a valid compression of c_1 in $\mathcal{G}$.

A graph compression can be achieved by picking one concept and compressing its edges in poweredges incrementally.

In the following sections we study how to reduce the possible permutations of concepts considered during this iteration, in order to obtain a good approximation of the optimal compression. we study three features of the search space as defined in this section. As in Power Graph Analysis, the ordering of concepts may be based on their edge cover. This is presented in section 2.6.1.

Second, the treatment of concept lattice symmetry and other properties may reduce the number of useful concepts. It is detailed in section 2.6.

Finally, we show that there is no guarantee that an optimal compression can

always be represented by an ordering of concepts. Section 2.8 gives an example of such a case, and discusses how it may be handled.

Before introducing more features of the compression search space, we propose an illustration of the complete compression process on simple graphs.

2.5 Examples of powergraph compression

The powergraph compression is illustrated in two examples, *Two-bicliques* and *Zorro*.

2.5.1 Two-bicliques example

Let us consider the graph defined by the formal context in Figure 2.4. It can be compressed by choosing first concept number 11 $\{d, i, j, k\} \times \{e, f, g, h\}$ in the concept lattice, then concept number 5 $\{d, e\} \times \{a, b, c, g\}$ (see Figure 2.3). This compression order is noted $(11, 5)$. Since these concepts overlap (they share nodes g, e and d, and edge $d \times g$), meeting the decomposition and hierarchy conditions require to split the second concept. It will be split in this case into three poweredges: $\{a, b, c\} \times \{d\}$, $\{a, b, c\} \times \{e\}$ and $\{g\} \times \{e\}$.

While choosing this concept order leads us to an optimal compression of the graph, other concept orders may lead to different compressions with different number of poweredges: for instance, order $(1, 2, 3, 7)$ will give a 5 poweredges compression, with the first created powernode being $\{a, b, c, d, i, j, k\}$, as shown in Figure 2.6.

The chosen concepts and their ordering (i.e. the order in which they are compressed) determine the final compressed graph. In this example, the second concept to be compressed was compressed in three steps: $\{a, b, c, g\} \times \{d, e\}$ was compressed as $\{d\} \times \{a, b, c\}$, $\{e\} \times \{a, b, c\}$ and $\{e\} \times \{g\}$. The standard heuristic is to always compress first the concept, or in case of overlap the poweredge, covering the highest number of edges. This will however not necessarily yield the best compression.

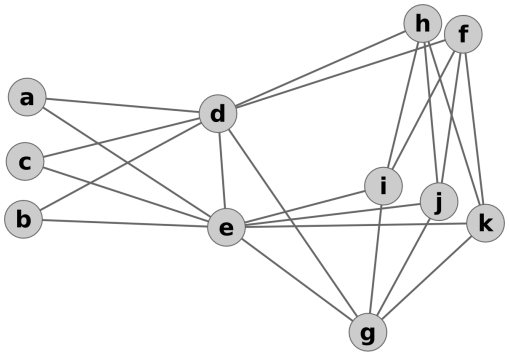


Figure 2.3 – The graph associated to the formal context in Figure 2.4 (left).

	a	b	c	d	e	f	g	h	i	j	k
a				×	×						
b				×	×						
c				×	×						
d	×	×	×		×	×	×	×			
e	×	×	×	×			×		×	×	×
f				×					×	×	×
g				×	×				×	×	×
h				×					×	×	×
i					×	×	×	×			
j					×	×	×	×			
k					×	×	×	×			

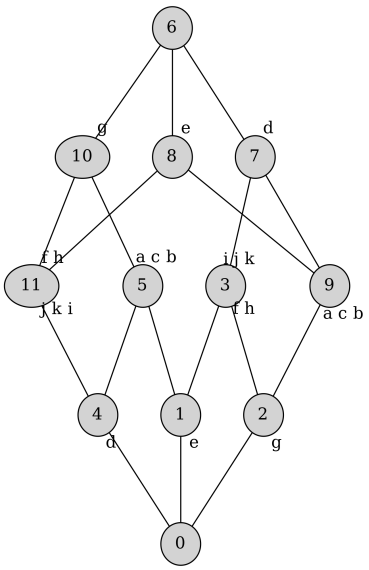


Figure 2.4 – Formal context describing the graph shown in Figure 2.3 and its associated concept lattice.

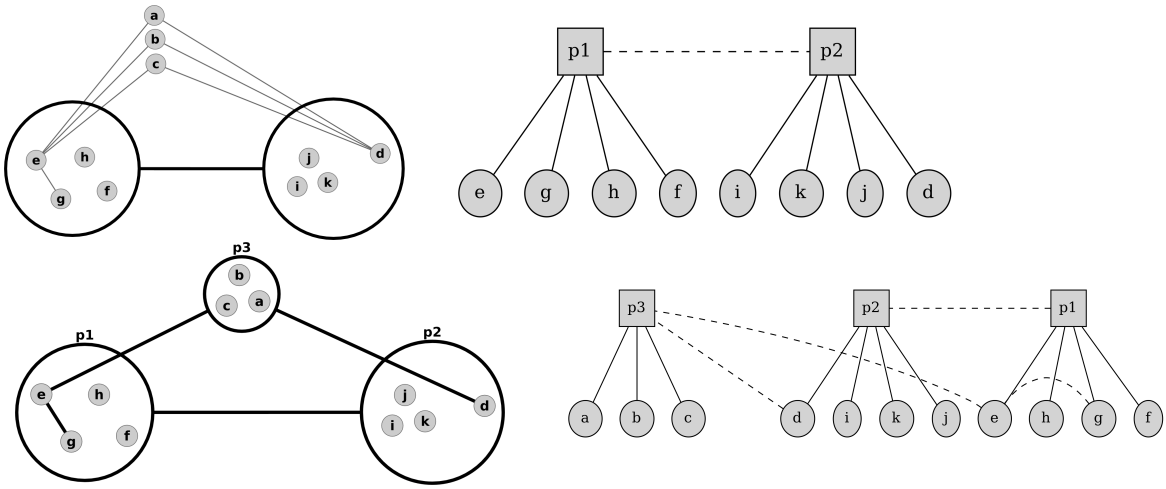


Figure 2.5 – *From top to bottom:* demonstration of a two-step powergraph compression on the graph shown in Figure 2.3, defined by the formal context in Figure 2.4 (left).
Left: the Power Graph compression, where the concept/biclique $\{d, i, j, k\} \times \{e, f, g, h\}$ (number 3 or 11 in Figure 2.4) was compressed first (top), followed by concept/biclique $\{a, b, c, g\} \times \{d, e\}$ (number 5 or 9 in Figure 2.4), that is split in 3 poweredges (bottom). *Right:* the representation of the powernode hierarchy at each step of the computation, with nodes as circles, powernodes as rectangles, poweredges as dashed edges and hierarchy relation in plain edges.

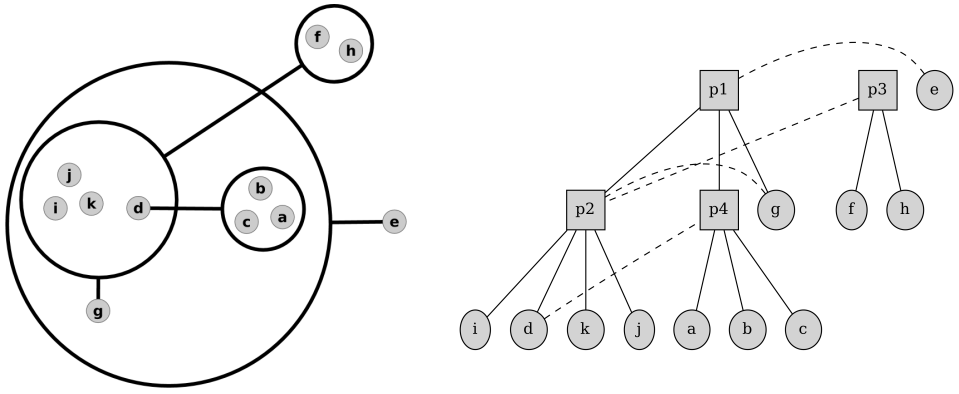


Figure 2.6 – A non-optimal compression of the graph shown in Figure 2.3, defined by the formal context on the left of Figure 2.4. *Left:* the Power Graph compression, where the concept/star $\{e\} \times \{a, b, c, d, g, i, j, k\}$ (number 1 or 8 in Figure 2.4) was compressed first, followed by concept/star $\{g\} \times \{d, e, i, j, k\}$ (number 2 or 10 in Figure 2.4), followed by concept/biclique $\{d, i, j, k\} \times \{e, f, g, h\}$ (number 3 or 11), and finally concept/star $\{d\} \times \{a, b, c, e, f, g, h\}$ (number 4 or 7). *Right:* the representation of the powernode hierarchy in the powergraph, with nodes as circles, powernodes as rectangles, poweredges as dashed edges and hierarchy relation in plain edges.

2.5.2 Zorro Example

This example aims at showing that even a simple graph may have multiple optimal compressions, ut the reaching of it may require subtle heuristic choices.

The graph is called Zorro, and shown in Figure 2.7 along with one of its powergraph compression. The graph is constituted of three bicliques B_1, B_2 and B_3 covering respectively 24, 20 and 16 edges. B_2 overlaps with the two others, so that the compression of that graph is not trivial. The iterative concept determination method, taking the biclique size ordering, B_1 is compressed first, then B_3, since B_2 is split in two bicliques covering 8 and 12 edges after the compression of B_1. The final powergraph, shown on the right of Figure 2.7, uses 6 poweredges.

With the selection-transformation method, taking the concept size ordering, we obtain the order (B_1, B_2, B_3). The obtained compression is optimal, and shown on the left of Figure 2.8.

Finally, a method relying on motif overlaps to decide the concept order could have observed that B_2 would have been cut in four if the others were compressed

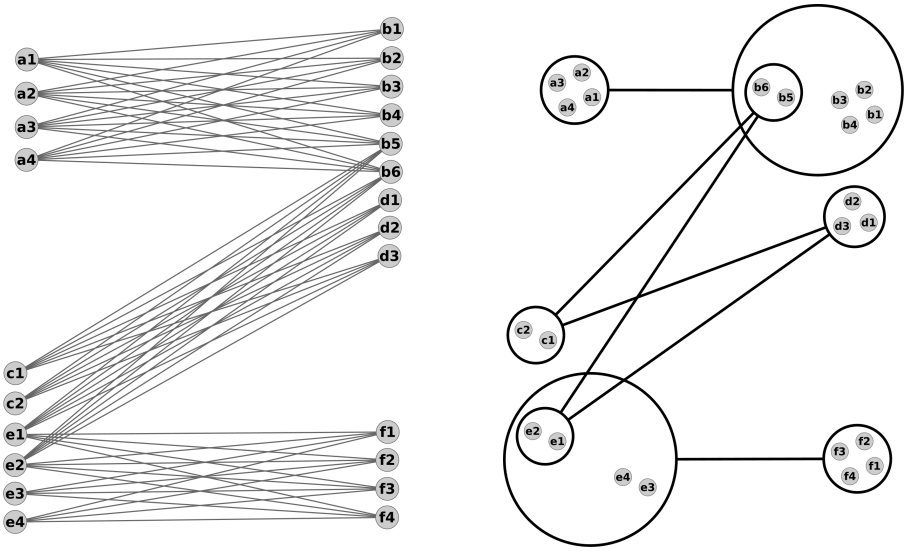


Figure 2.7 – The Zorro graph example. *Left:* the graph, constituted of 3 maximal bicliques of edge cover 24, 20 and 16. *Right:* a suboptimal Power Graph Analysis of the graph.

first, so B_2 cannot be last. This suggest to try B_2 first, resulting in a second optimal solution shown on the right Figure 2.8.

With Royer *et al.* implementation of Power Graph Analysis, we can obtain the two optimal compressions shown in Figure 2.8 by tweaking the input parameters. As we will show in the next section, our implementation *PowerGrASP*, because of the direct implementation of iterative concept determination method, only outputs the suboptimal powergraph shown on the right of Figure 2.7. This point to the interest of introducing some capacities of backtracking in *PowerGrASP* in the future.

2.6 Power Graph Analysis and *PowerGrASP* heuristics

2.6.1 Power Graph Analysis and *PowerGrASP* heuristics

The search spaces presented in section 2.4 are not practical for graphs with more than a few nodes. As a consequence, an approximation method exploring only a

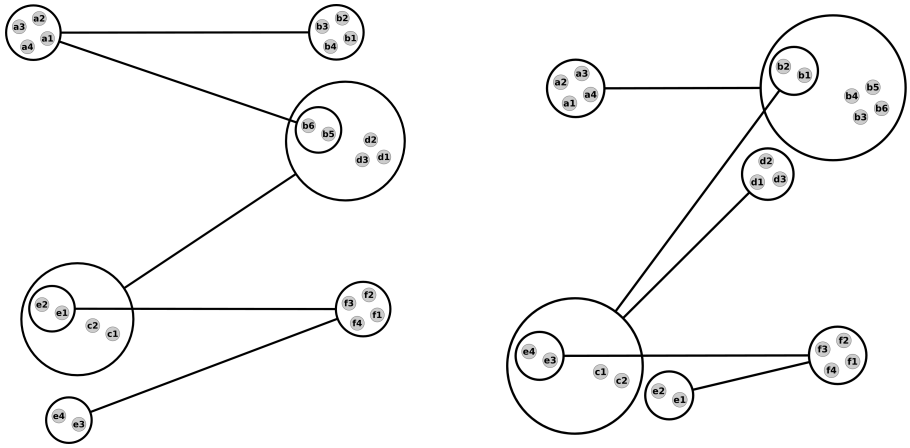


Figure 2.8 – Two optimal compressions of the Zorro graph example. *Left:* Obtained by compressing the middle biclique of score 20 first. *Right:* Obtained by compressing the upper biclique of score 24, then the middle biclique of score 20.

fraction of the search space has to be devised. Note that the determination of which concept is to be compressed next is independent of the compression method (iterative concept determination or selection-transformation method).

Oog*: Power Graph Analysis by Royer *et al.

The heuristic proposed by Royer *et al.*, fully described in [90], is based on ascending hierarchical classification. It consists in taking greedily the largest motif to compress (maximizing the number of covered edges). Then each node is placed in its own cluster, and a neighborhood similarity score is computed for each pair of nodes. Iteratively, the pair of clusters having the maximal score is grouped in a new cluster, until no pair of cluster reaches a minimal neighborhood similarity. A list of poweredges is then derived from the existing clusters. Then, in decreasing edge-cover size, motifs are compressed, and if necessary decomposed into multiple poweredges that are added to the list. It accepts as input parameter a minimum similarity threshold for cluster merge, and use the weights on edges to adjust the similarity scores.

From the FCA point of view, the Power Graph Analysis heuristic is implementing the compression with (1) a motif discovery based on similarity scores and bottom-up hierarchical clustering, and (2) the iterative concept determination method for the choice of motifs to compress.

PowerGrASP

We implemented *PowerGrASP*, an ASP-based implementation of powergraph compression, that differs from Royer *et al.* implementation in the first part of the program: instead of a motif search relying on a heuristic, *PowerGrASP* performs a full enumeration of concepts corresponding to maximal cliques and bicliques in the graph. In the second part (creating poweredges based on detected concepts), *PowerGrASP* is a direct implementation of the iterative concept determination method. Its performance, as it will be shown in later chapter, is insufficient for large graphs, but medium sized graphs remain compressible in acceptable time, achieving the same results as Royer *et al.* implementation.

We propose in this section two reductions of the search space based on the concept lattice representation or the AOC-poset.

As we will see in next sections, the main interest of *PowerGrASP* lies in its ASP implementation, that has been used to test some optimizations for time and memory. The stars and quasi-bicliques motifs, presented in section 2.2.3, were implemented without having to modify the core algorithm nor the core encodings.

2.6.2 Reducing the concept lattice symmetry

The formal context is symmetric by construction : all nodes are present both in objects and in attributes. This property is visible in the concept lattice, where a concept (A, B) has a symmetric (B, A) covering the same edges. A naive solution to handle this redundancy would be to fix an arbitrary ordering $>$ on the object/attribute sets and remove duplicates (A, B) if $A > B$. This is correct but many inclusion links between concepts may be lost in the remaining structure since the choice are independent from the lattice structure.

We propose a structure-preserving procedure, described in algorithm 4. It uses an arbitrary fixed total ordering on concepts (numbering) to choose a direct child of the supremum to keep, and consequently a direct parent of the infimum to

discard. Each direct child of the supremum is called a *root*. Their symmetrical are the direct parents of the infimum. By marking a root as *kept* (i.e. deciding to keep it in the reduced lattice), we also set its symmetrical to be *discarded* (absent in the reduced lattice). All childs of the root are also kept, and therefore all parents of the root symmetrical (which includes at least one root) are discarded. This is repeated until no more root is marked either as kept or discarded.

Because a node cannot be linked to itself in the formal context, it is not possible to have a concept marked both as kept and discarded. The resulting structure is not itself a concept lattice, it is just used to choose the list of candidate concepts for the compression. Depending of the root order, there are multiple reductions possible, but all are equivalent in term of edge cover.

The concept lattice of the example graph in Figure 2.3 cleared of redundant concepts is shown in Figure 2.9.

Concept lattice reduction 4 Top-down reduction of a given symmetric lattice by deciding nodes/concepts to keep.

Require: Symmetric Concept Lattice L, symmetries between concepts
Ensure: Compute the set of concepts to keep as *taken*

```
1: taken ← ∅
2: discarded ← ∅
3: roots ← direct_childs(supremum(L))
4: for all root ∈ roots do
5:    if root ∉ taken ∪ discarded then
6:       taken ← taken ∪ subconcepts(root)
7:       discarded ← discarded ∪ supconcepts(symmetric(root))
8:    end if
9: end for
```

2.6.3 Reducing the number of concept permutations with the AOC-poset

Necessary Concepts (Kernel)

A concept that is compressed in all the optimal compressions (regardless of symmetry) is necessary. An obvious optimization would be to discard any solution that does not contains all the necessary concepts. Determining the complete set of

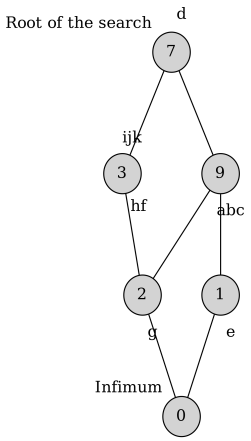


Figure 2.9 – A reduced concept lattice representation of the lattice in Figure 2.4. Concept 7 is the only root since it is the only child of the lattice supremum.

necessary concepts is an open problem, but the following definition 2.10 provides a property of a subset of them. the idea is to identify edges that are covered by only one formal concept. If those edges are to be compressed, it must be through the compression of that concept.

Definition 2.10. *Necessary Concept*
Let $C = (A, B)$ be both an object-concept and an attribute-concept of specific objects $X \subseteq A$ and specific attributes $Y \subseteq B$. C is the only concept (with its symmetric, removed by reduction) to cover edges $(x, y) \quad \forall x \in X, y \in Y$. Therefore the only way to compress them is to use C. If X and Y are singletons, a single edge is concerned, thus there are no compressible edges specific to that concept. If (X, Y) covers 2 or more edges, there are at least two edges to compress that can only be compressed with C. C is therefore a necessary concept.

Example 2.5. In the graph example of Figure 2.4, concept 11 and its symmetric 3 are the only ones to cover edges $\{f, h\} \times \{i, j, k\}$. They are necessary and need to be compressed at some point.

2.7 Optimizations using graph and motifs properties

PowerGrASP implementation of powergraph compression enabled us to test numerous optimizations aiming at reducing the time needed to perform the overall compression. This section detail the optimizations that are not specific to our implementation, and may be back-ported to other implementations, not necessarily using ASP nor specifically looking for a powergraph.

We end with a proposition for a divide-and-conquer strategy to mine specifically the largest formal concept in a formal context.

Note on bounds

PowerGrASP, in order to limit the search space of concept search, computes a lower and upper bound on the edge cover of the motif to compress for the ASP solver. The bounds are computed in a simple manner ; *PowerGrASP* could certainly benefit from recent research in branch-and-bound algorithms.

2.7.1 Star search

This optimization consists into the consideration of a third graph motif: stars.

The star is a special case of biclique, and as such doesn't need to be searched. However, bounds for a star search are easier to compute than bounds for bicliques: the maximal and minimal bound for the maximal stars search in a graph are equal to the greatest node degree. Stars are also preponderance in many datasets.

By delegating the star search to a specific process, the biclique search space is reduced, since both sets of the biclique must contain at least 2 nodes.

From a concept point of view, stars can be enumerated from the AOC poset, as described in section 2.2.3, and the constraint on the biclique search space is simply to only enumerate concepts with at least 2 objects and 2 attributes.

2.7.2 Graph filtering

Some edges of the input graph cannot be part of the largest concept, and can be pruned before looking motifs. For this optimization to provide an interesting *edge loss*, a good lowerbound on the motif edge cover has to be defined. The higher the lowerbound, the higher number of edges may be filtered out. From a concept point of view, this is equivalent to removing relations from the graph context that are not involved in a specific type of motif.

Proposition 2.1. *Edge filtering for largest star, biclique and clique*
Let $G = (V, E)$ be a graph, and l the lowerbound for the edge cover of a motif. An edge (x, y), such that:

- *$degree(x) < l$ and $degree(y) < l$, cannot belong to a star with an edge cover of l or more*
- *$degree(x) \times degree(y) < l$, cannot belong to a biclique with an edge cover of l or more*
- *$clustering_coefficient(x) == 0$, does not belong to a clique.*

Proof. Either x or y must be the hub of the star and have a degree at least l. This is the same logic for bicliques. A node with a clustering coefficient of 0 cannot belong to a clique, and so are its adjacent edges. □

This optimization is implemented in *PowerGrASP* as follow: each motif to be searched is associated to a copy of the initial graph, and edges are pruned from this graph, definitively when covered by a motif previously compressed, or for the current step with graph filtering, as described in proposition 2.1. For instance, the graph on which stars are searched is pruned of all edges that cannot belong to a star. When the star search (cf section 2.7.1) is used, there are three motifs: star, non-star biclique, and clique. Each may use its associated graph filtering method. Each motif search is therefore working on its own graph.

This memory/time tradeoff can be shifted by using only one graph for all motifs, and removing an edge only if it matches the three conditions of proposition 2.1.

Removing edges from a graph may have important implications on connectivity. An obvious effect of graph filtering is to split connected components into smaller connected components. In the context of graph compression, this is a great way to decrease importantly the size of the search space: because of the exponential growth of the complexity, two connected components of n edges are easier to compress than a single one with $2n$ edges. This particular connected component detection is not implemented in *PowerGrASP*.

2.7.3 Stable search in graph context to reduce the search space

If the number of nodes of a graph is reduced by a factor k, its graph context is reduced by a factor k^2 and the search space of concepts is reduced by a factor 2^{k^2} . This emphasizes the well known importance of reduction strategies as a preprocessing step of a concept search. Simple techniques include the application of standard clarification and reduction procedures and the decomposition of the graph into connected components. Concerning the FCA reduction procedures and since we are interested in largest concepts, the computation of the edge cover size requires all reduced nodes to have an associated weight counting the number of nodes they represent. Concerning the decomposition of the graph, we propose a generalization of the connected component property, the *bipartite property*, aiming at further splitting the graph context. It be applied recursively, in order to work on increasingly smaller contexts.

We first introduce the *dot operator* on sets of objects or attributes, which is a relaxed variant of the derivation operator of FCA where the universal quantification is replaced by an existential one:

Definition 2.11. *Dot operator*
Given a set of objects X (resp. attributes Y), the set $\dot{X}$ (resp. $\dot{Y}$) is made of all attributes (resp. objects) related to *at least one* attribute in X (resp. Y):

$$\dot{X} = \{y \in Y \mid \exists x \ \in \ X, \ r(x, y)\} \qquad \dot{Y} = \{x \in X \mid \exists y \ \in \ Y, \ r(x, y)\} \tag{2.1}$$

As for derivation, the dot operator can be combined multiple times:

$$\ddot{X} = \{x \in X \mid \exists y \in \dot{X},\ r(x,y)\} \qquad \ddot{Y} = \{y \in Y \mid \exists x \in \dot{Y},\ r(x,y)\} \tag{2.2}$$

Search for a *divide-and-conquer*-like strategy

A stable in a graph is a set of unconnected nodes, i.e. a clique in the complementary graph. In a graph context, it translates as an empty rectangle, and can be used to reduce the sets of nodes to consider as candidates for the composition of the largest concept. Proposition 2.2 expresses the relation between a stable and the search space for the largest concept in the FCA framework.

Proposition 2.2. *Given a set of objects O and a set of attributes A, let $P = \{O_1, O_2\}$ be a partition of O and $Q = \{A_1, A_2\}$ be a partition of A. Let $LC(O, A)$ denotes a largest concept of the formal context $\mathcal{C}(O, A)$, i.e. a concept corresponding to a submatrix of largest size. Then, the following property holds:*

$$LC(O, A) = \max\left(LC(O_1, A_1), LC(\dot{A}_2 \cup \ddot{O}_2, \dot{O}_2 \cup \ddot{A}_2)\right) \tag{2.3}$$

Moreover, this equation may be refined if no relation holds over $O_1 \times A_1$:

$$LC(O, A) = \max\left(LC(\dot{A}_2, A_2), LC(O_2, \dot{O}_2)\right) \tag{2.4}$$

Proof. Let $P = \{O_1, O_2\}$ be a partition of a set of objects O and $Q = \{A_1, A_2\}$ be a partition of a set of attributes A. If the largest concept $LC(O, A)$ is in the context $\mathcal{C}(O_1, A_1)$, then by definition, it will take the right value $LC(O_1, A_1)$. Else, there exists an element of the largest concept either in O_2 or in A_2. If the element is in O_2, then the attributes of $LC(O, A)$ have to be included in $\dot{O}_2$. The same way, the objects of $LC(O, A)$ have to be included in the objects sharing at least one relation with attributes of $\dot{O}_2$, that is, $\ddot{O}_2$. With a symmetric argument, if there is an element of A_2 in the largest concept, then attributes of $LC(O, A)$ have to be included in $\dot{A}_2$ and objects in $\ddot{A}_2$. Altogether $LC(O, A)$ must be a concept of the context $\mathcal{C}(\dot{A}_2 \cup \ddot{O}_2, \dot{O}_2 \cup \ddot{A}_2)$.

If no relation holds over $O_1 \times A_1$, then every concept has either all its elements in O_2 or all its elements in A_2. In the first case they are in the formal context

		A_1			A_2			
		H	I	J	K	L	M	N
O_1	a					×	×	
	b							
	c				×	×	×	×
O_2	d			×		×	×	×
	e	×		×	×			
	f	×		×	×		×	×
	g			×	×		×	×

Table 2.1 – A partitioned context with no relation over $\mathcal{C}(O_1, A_1)$. The five possible positions of the largest concept $\mathcal{L}$ are shown. $\mathcal{L}$ could be in $\mathcal{C}(O_2, A_1)$ (e.g. $(\{e, f\}, \{H\})$), $\mathcal{C}(O_2, A_2)$ (e.g. $(\{f, g\}, \{M, N\})$), $\mathcal{C}(O_1, A_2)$ (e.g. $(\{a\}, \{L, M\})$), $\mathcal{C}(O_2, A_1 \cup A_2)$ (with an element in A_1 and an element in A_2, e.g. $(\{e, f, g\}, \{J, K\})$) or $\mathcal{C}(O_1 \cup O_2, A_2)$ (e.g. $(\{c, d\}, \{L, M, N\})$).

$\mathcal{C}(O_2, \dot{O}_2)$, and in the second case they are in the formal context $\mathcal{C}(\dot{A}_2, A_2)$. The largest concept is the largest of the largest concepts of the two contexts. Table 2.1 gives details on the way the search can be split in this case. □

From the point of view of graph modeling, the fact that no relation holds over $O_1 \times A_1$ means that it is stable set of the graph (note however that O_1 and A_1 may overlap). If moreover no relation holds over $O_2 \times A_2$, it corresponds to the existence of at least two connected components in the graph. From a graph perspective, if $O_1 = A_1$, O_1 is a stable set, if $O_1 \cap A_1 = \emptyset$, O_1 is a bistable (two sets of unrelated nodes), but the more general motif represented by $(0_1, A_1)$ is the overlapping bistable, i.e. a biclique in the complementary graph whose sets overlap. Moreover, we get $O_2 = A_2$, so $LC(\dot{A}_2, A_2) = LC(O_2, \dot{O}_2)$ and it is sufficient to consider the graph context $\mathcal{C}(O, \dot{O}_2)$. Since the aim is to speedup the search for largest concepts, the most straightforward implementation of an optimization based on proposition 2.2 is a search in bounded time of a stable (not necessarily maximal).

Implementation of Proposition 2.2 remains to be seen. Later chapter 4 will (1) show that notable stables are common in the datasets, suggesting that this kind

of optimization may have interesting applications.

2.7.4 Parallel compression of independent concepts

When its first concept is compressed, a powergraph ceases to present a simple structure, precluding the application of previously described optimizations such as graph filtering, that would need to be developed specifically to be applicable to a powergraph. Another limit of the base *PowerGrASP* implementation is that at each step, the largest concept to compress is searched.

In order to avoid such costly approach, one may enumerate all maximal concepts M, and compress them all, thus compressing multiple concepts with only one search. The problem here is that if some maximal concepts are overlapping, one has to choose the concepts to compress. In *PowerGrASP*, this is implemented with a greedy approach. First, the solver enumerates the optimal solutions. Second, one concept chosen randomly is compressed, and the remaining ones are compressed one after the other if they do not overlap with a previously compressed concept.

A better approach would be to find the maximal subset $S \subseteq M$ of maximal concepts so that no concepts of S is overlapping with another concept of S.

From a concept point of view, this is equivalent to the search for a set of edge-maximal concepts which meet and join are the supremum and infimum.

2.8 Limits of greedy approaches: the cycle motif

One specific class of concepts layout is resisting to the previously defined concept approach. We call them cycles of concepts, an example being displayed in Figure 2.10 for a 4-cycle. A cycle of concepts is a series of concepts that form a circular chain by inclusion of the intent of one concept in the extent of the next one. Any cycle from 3 to any number of concepts will in fact never be optimally compressed by the procedure we have used so far iterating on the choice of concepts.

The previous section has shown the interest of clique motifs in addition to bicliques to compress graphs and we highlight here a new cycle pattern that un-

derlines the richness of this pattern recognition approach.

As already sketched, the peculiar status of this cycling motif is due to a special organization of concepts, that the concept lattice helps to unravel. In a concept cycle motif, all involved formal concepts of the form $\{(A_1, B_1), ..., (A_n, B_n)\}$ are ordered so that $A_k \subset B_{k+1} \forall k \in 1..n-1$ and $A_n \subset B_1$. In fact, the basic building block for a cycle is a pair of overlapping concepts $A_1 \times (A_2 \cup B_1)$ and $A_2 \times (A_1 \cup B_2)$, where all sets are disjoint. The two concepts could be represented by a quadruplet (B_1, A_1, A_2, B_2), where all contiguous elements form a biclique. The biclique (A_1, A_2), also not maximal, is a consequence of the fact that a set may appear either as an extent or an intent.

It appears that cycle contexts often lead to a concept lattice that is made, apart from the top and bottom concept, of a graph cycle (see Figure 2.11, left) or two symmetric graph cycles (see Figure 2.11, right). However, the 4-cycle (Figure 2.10) has a special shape where the cycle has been interrupted by intermediary concepts.

It is thus possible to find globally optimal bicliques organizations that are not based solely on the use of maximal bicliques but also use bicliques associated with overlapping concepts. An enumeration of the corresponding concepts can lead to a systematic detection of the cyclic pattern, and the incorporation of more general versions of motif-concepts.

Further work on the search space might also point to other specific motifs helping to better compress the graph through meaningful recurrent patterns.

2.9 Conclusion

2.9.1 Contributions

A formalization of Power Graph Analysis and its search space

A general graph compression search space formalization in the framework of FCA was formulated, highlighting the main sources of difficulty of the problem.

Once the concepts covering the graph have been generated, the compression process can be expressed as the choice of an ordering of a subset of all concepts. We have shown that object and attribute-concepts are useful to focus on partic-

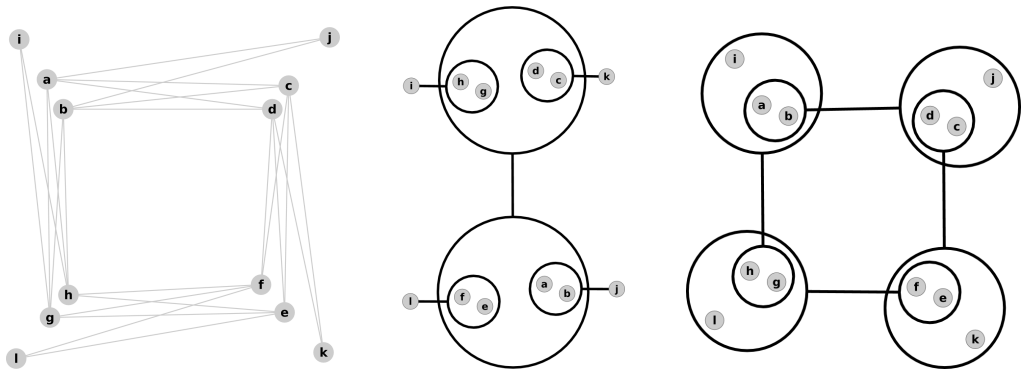


Figure 2.10 – A cycle of 4 concepts, non-compressed (left), compressed by a greedy method (middle), and optimally compressed (right). The greedy approach compresses first the largest concept (here $\{a, b, e, f\} \times \{c, d, g, h\}$), then 4 small bicliques, ending with 5 poweredges. The optimal compression is reached by using and splitting the four concepts $\{a, b\} \times \{c, d, e, f, j\}$, $\{c, d\} \times \{a, b, e, f, k\}$, $\{e, f\} \times \{c, d, g, h, l\}$, $\{g, h\} \times \{a, b, e, f, i\}$, leading to only 4 poweredges.

ular subsets and the selection of subsets remain an interesting track for further researches.

The standard heuristic for the generation of graph compression is fast but only computes an approximation of the minimal Power Graph [20, 94]. Once the (triplet) concepts have been generated, the results of the Power Graph Analysis can be reproduced by ordering the concepts by decreasing surface. This heuristic avoids to explore the space of all permutations, explaining its efficiency, despite that the approach based on a permutation over the concepts is not feasible for graphs having more than a dozen concepts.

Other approaches to graph compression ought to improve the speed or the optimality by allowing to reuse edges among multiple poweredges [37], or the overlapping of powernodes to handle simply non-disjoint sets [3]. Such approaches correspond to a relaxation of some of the constraints on Power Graphs. This can be encoded in the concept lattice formalization as a relaxation of the compression of concepts operating on the same nodes or edges. The search space for that matter is not different, despite the constraint relaxations.

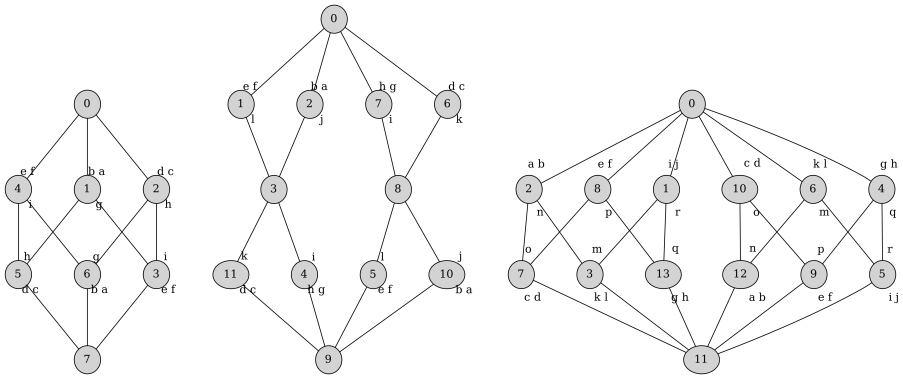


Figure 2.11 – The lattices of the 3-cycle (left), 4-cycle (middle) and 6-cycle (right) motifs. The latter presents two symmetric cycles, which is the consequence of a bipartite graph (an odd cycle). On the other hand, the 3-cycle is not bipartite, so the lattice symmetry is not perfectly separated.

Optimizations

Three types of optimization were proposed. First, the optimizations related to the search space, directly understandable as reduction of the concept lattice or the AOC poset. However, those optimizations does not mitigates the factorial complexity of the search space. Second, the optimizations allowing our implementation of Power Graph Analysis, *PowerGrASP*, to reduce its search space, such as the *graph filtering* that discard relations of the formal context that does not belong to the largest formal concept. And third, a proposition to a divide-and-conquer strategy to discard parts of the graph context that cannot contain the largest concept, thus limiting the search space of an implementation of Power Graph Analysis.

Limits of the Concept-Only Methods

We unraveled a particular graph motif that prevents heuristics of Power Graph Analysis to achieve optimal compression when this motif is present in the graph. We therefore shown that a compression method seeking only for concept-based compression in the graph context cannot reach the optimal compression, because of at least one graph motif: concept-cycles. To handle this motif, which uses non maximal bicliques, some specific pattern detection on the lattice has to be designed. For a more global point of view, applying this approach of pattern recognition in

the concept lattice could also be the basis for any motif that would convey a specific meaning in the data.

2.9.2 Towards an unification of cliques and bicliques

First, cliques and bicliques require respectively reflexive and non-reflexive graph contexts to be encoded as a single concept. Else, they are represented as an intertwining of bicliques/concepts, instead of a single coherent representation that is used by Power Graph Analysis: a single powernode with a reflexive poweredge.

The concept lattice-based model of the search space has some limits. First, the bicliques and cliques are handled as different objects. Because the latter needs the reflexive edges, two independent motif searches must be performed, on two different concept lattices. Moreover, formal concepts such that $X \neq Y$ and $X \cap Y \neq \emptyset$ convey information, but are not identified as compressible motifs. An extension of the formal concepts is needed to overcome these two limits: the *triplet concepts*, subject of chapter 3.

See chapter 6 for perspectives on this work.

Chapter 3

Triplet concepts: an extension of FCA looking for overlapping bicliques

---

> There are only two hard things in Computer Science: cache invalidation, naming things, and off-by-one error.
>
> Phil Karlton

The previous chapter showed the limits of handling separately the clique and biclique motifs in the context of graph compression. Each motif needs its own graph context and thus two concept lattices are built during compression, as two independent tasks. However, the interaction between the choices of both motifs seems crucial for a good compression level. A new formal framework is necessary for defining an efficient integrated search space.

For this purpose, this chapter proposes an extension of formal concepts, *triplet concepts*, which is tailored for symmetric formal contexts that are internal binary relations on any given set. Triplet concepts represent a more general graph motif, overlapping bicliques, from which cliques and bicliques are special cases.

Section 3.1 will first provide an intuition on the notion of triplet, and section 3.2 will place it in the context of graph theory. As for formal concepts maximality is a desirable property of triplets that needs to be properly specified in this context. This is covered by section 3.3. This property is not sufficient however since many maximal triplets can be equivalent and one has to design a unique, standard representation including all the special and degenerated cases. This is the subject of section 3.4, and the last piece needed to formally define triplet concepts, as shown in 3.5.

Later sections will introduce additional properties for practical triplet generation, such as an admissibility criterion for triplet maximality — detailed in section 3.6 — which will greatly simplify triplet concept mining, the triplet concept ordering addressed in section 3.7, and, in section 3.8, the close link between formal concepts and triplets, showing how concepts can be used to enumerate triplets.

Section 3.9 addresses algorithmic issues for triplet concepts enumeration and shows different approaches for this task. Finally, section 3.10 will explain the relation between triplet concepts enumeration and Maximal Clique Enumeration.

3.1 Intuition and examples

3.1.1 The graph context

The graph context object, and its reflexive counterpart, have been introduced in section 2.1 in order to encode graph motifs. This section redefines them for completeness, and exposes a new property enabling to deal with problems raised by the previous chapter. The definition of formal contexts, concepts and concept lattice in FCA, have already been provided in section 1.3.

Let us now consider a formal context where objects and attributes are a same set S, and the context is a binary symmetrical relation rel on S.

That kind of formal context is named *graph context*, because it is similar to an adjacency matrix: it can be used to describe a simple undirected graph, where objects and attributes are nodes of the graph, and $rel(x, y)$ holds iff node x is linked to node y.

A graph context is *reflexive* if the relation is reflexive, i.e. every item is in relation with itself. From now on, we will always consider the graph context together with its reflexive version as two representations of the same graph. This is not a real restriction, since it is always possible to mark nodes that are really with a reflexive link, and since the reflexivity assumption is only used to simplify the definitions and treatments. This assumption originates from Power Graph Analysis, where self-loops are considered non functional and added as don't care edges.

Consequently, the considered graphs are simple graphs, and the use of self-loops

is only a matter of technical data manipulation enabling one to easily describe natural kinds of motif, such as cliques. As a consequence, in this chapter, a graph context and its reflexive version are describing the same graph, but because of their differences will be used in different ways.

3.1.2 Formal concepts of the graph context

In the case of *reflexive graph context*, the extent X and the intent Y of a formal concept (X, Y) derived from a graph context may overlap. This is illustrated in example 3.1: some formal concepts issued from the reflexive graph context in Figure 3.1 have overlapping sets.

The existence of this overlap changes the interpretation of the formal concepts regarding graph structure, since it can describe not only a relation between 2 sets of nodes (as in a non-reflexive graph context), but also the internal connections between the elements in each set. In order to explicit the existence of the extent/intent overlap, we introduce the notion of *triplet* in definition 3.1. This is a first necessary definition that will gradually be refined in later sections in order to define *triplet concepts*.

As an example, triplets found in the reflexive graph context of Figure 3.1 are shown in Table 3.1 (right column), and discussed in example 3.3.

Example 3.1. Considering the graph context in Figure 3.1, and the associated formal concepts and concept lattice. With a non-reflexive relation (left in the figure), the corresponding concept lattice contains 16 formal concepts, listed in the first column of Table 3.1. All are made of two non-intersecting sets. For instance, three of them are $\{a, b, c, d, e, f\} \times \emptyset$, $\{a, b\} \times \{c, d, e, f\}$ and $\{c, d, e\} \times \{a, b, f\}$.
By making the relation reflexive (right in the figure), some of the formal concepts are made of a pair of overlapping sets, for instance $\{a, b, c, d, e, f\} \times \{e, f\}$, $\{a, b, e, f\} \times \{c, d, e, f\}$ and $\{c, d, e, f\} \times \{a, b, e, f\}$, overlapping on nodes e and f. See the second column of Table 3.1 for a complete list.
The number of concepts remains the same in this example, but as shown in example 3.2 this is not the general case.

	a	b	c	d	e	f
a			×	×	×	×
b			×	×	×	×
c	×	×			×	×
d	×	×			×	×
e	×	×	×	×		×
f	×	×	×	×	×	

	a	b	c	d	e	f
a	×		×	×	×	×
b		×	×	×	×	×
c	×	×	×		×	×
d	×	×		×	×	×
e	×	×	×	×	×	×
f	×	×	×	×	×	×

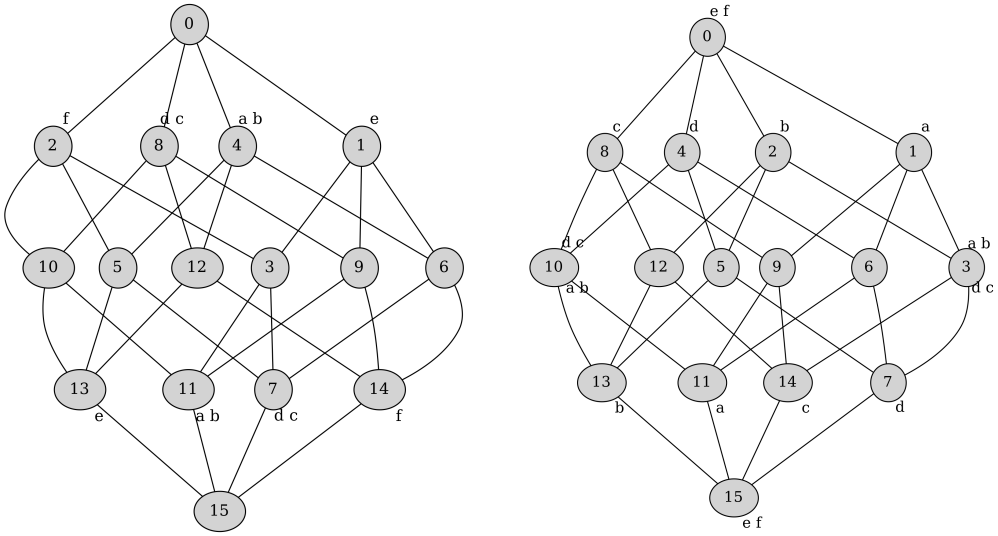


Figure 3.1 – *Left:* A small graph context on the set $S = \{a, b, c, d, e, f\}$. *Right:* its reflexive variation (where $\forall x \in S, rel(x, x)$ holds), and their corresponding concept lattices.

	a	b	c
a		×	
b	×		×
c		×	

Figure 3.2 – A small graph context on the set $S = \{a, b, c\}$.

	concepts	reflexive concepts	triplet representation
1	$\boldsymbol{\{a,b,c,d,e,f\} \times \emptyset}$	$\boldsymbol{\{a,b,c,d,e,f\} \times \{e,f\}}$	$(\{a,b,c,d\}, \emptyset, \{e,f\})$
2	$\{a,b,c,d,f\} \times \{e\}$	$\{a,b,d,e,f\} \times \{d,e,f\}$	$(\{a,b\}, \emptyset, \{d,e,f\})$
3	$\boldsymbol{\{c,d,e\} \times \{a,b,f\}}$	$\boldsymbol{\{c,d,e,f\} \times \{a,b,e,f\}}$	$(\{c,d\}, \{a,b\}, \{e,f\})$
4	$\{c,d\} \times \{a,b,e,f\}$	$\{a,d,e,f\} \times \{a,d,e,f\}$	$(\emptyset, \emptyset, \{a,d,e,f\})$
5	$\{a,b,e\} \times \{c,d,f\}$	$\{b,c,d,e,f\} \times \{b,e,f\}$	$(\{c,d\}, \emptyset, \{b,e,f\})$
6	$\{e\} \times \{a,b,c,d,f\}$	$\{a,c,d,e,f\} \times \{a,e,f\}$	$(\{c,d\}, \emptyset, \{a,e,f\})$
7	$\boldsymbol{\{a,b\} \times \{c,d,e,f\}}$	$\boldsymbol{\{a,b,e,f\} \times \{c,d,e,f\}}$	$(\{a,b\}, \{c,d\}, \{e,f\})$
8	$\emptyset \times \{a,b,c,d,e,f\}$	$\{d,e,f\} \times \{a,b,d,e,f\}$	$(\emptyset, \{a,b\}, \{d,e,f\})$
9	$\{e,f\} \times \{a,b,c,d\}$	$\{a,b,c,e,f\} \times \{c,e,f\}$	$(\{a,b\}, \emptyset, \{c,e,f\})$
10	$\{a,b,f\} \times \{c,d,e\}$	$\{b,d,e,f\} \times \{b,d,e,f\}$	$(\emptyset, \emptyset, \{b,d,e,f\})$
11	$\{c,d,e,f\} \times \{a,b\}$	$\{b,c,e,f\} \times \{b,c,e,f\}$	$(\emptyset, \emptyset, \{b,c,e,f\})$
12	$\{a,b,e,f\} \times \{c,d\}$	$\{b,e,f\} \times \{b,c,d,e,f\}$	$(\emptyset, \{c,d\}, \{b,e,f\})$
13	$\{c,d,f\} \times \{a,b,e\}$	$\{a,c,e,f\} \times \{a,c,e,f\}$	$(\emptyset, \emptyset, \{a,c,e,f\})$
14	$\{f\} \times \{a,b,c,d,e\}$	$\{c,e,f\} \times \{a,b,c,e,f\}$	$(\emptyset, \{a,b\}, \{c,e,f\})$
15	$\{a,b,c,d,e\} \times \{f\}$	$\{a,e,f\} \times \{a,c,d,e,f\}$	$(\emptyset, \{c,d\}, \{a,e,f\})$
16	$\{a,b,c,d\} \times \{e,f\}$	$\{e,f\} \times \{a,b,c,d,e,f\}$	$(\emptyset, \{a,b,c,d\}, \{e,f\})$

Table 3.1 – The 16 concepts derived from context of Figure 3.1, along with the corresponding 16 concepts derived from the reflexive graph context of Figure 3.1, and their triplet representation.

Example 3.2. The context of Figure 3.2 contains only two formal concepts, $\{b\} \times \{a,c\}$ and $\{a,c\} \times \{b\}$. When considering the reflexive relation, there are four formal concepts $\{b\} \times \{a,b,c\}$, $\{a,b,c\} \times \{b\}$, $\{a,b\} \times \{a,b\}$ and $\{b,c\} \times \{b,c\}$.

If we consider only the subcontext $\{a,b\} \times \{a,b\}$, there are two concepts ($\{b\} \times \{a\}$ and $\{a\} \times \{b\}$) but only one ($\{a,b\} \times \{a,b\}$) with a reflexive relation.

Together with example 3.1, this shows that adding reflexive relations in a formal context may increase or decrease the number of formal concepts.

Regarding the number of formal concepts: in a non-reflexive graph context, the number of formal concepts is always even, because a concept (X,Y) is equivalently represented by the concept (Y,X). However, when reflexive relations hold, a concept (X,X) may exist. This is illustrated in example 3.2, where the graph context in Figure 3.2 has different amounts of formal concepts in its reflexive and

non-reflexive versions.

Let us now define formally triplets:

Definition 3.1. *Triplet* Let us consider a concept (X, Y), formed by possibly overlapping extent and intent. This concept can be written as pairs $(A \cup C, B \cup C)$, where A, B and C are disjoint and C is the set of vertices common to the extent and the intent of the concept. We can write this formal concept as a *triplet* (A, B, C), such as:

$$\begin{aligned} A &= X \setminus Y \\ B &= Y \setminus X \\ C &= X \cap Y \end{aligned} \tag{3.1}$$

As a practical notation we will often designate the first, second and third element of a triplet as the A, B and C set. By definition, reflexivity only occurs in the C set. In a non-reflexive graph context, the intersection of the extent and the intent of a concept must be empty, hence C is empty. Conversely, if C is not empty, the question arises as to whether C provides additional power of expression in terms of graph structures apart from reflexive loops on nodes, since self-loops are considered as virtual structures.

The answer is positive. In fact, a triplet represents the union of three cross-products: $A \times B$, $(A \cup B) \times C$ and $C \times C$. For instance, even a simple triplet like $(\{a\}, \{b\}, \{c\})$ describes a reflexive link on c, but also depicts a triangle structure (abc). Using only bicliques, two are needed to describe a triangle.

However, the reflexive closure of a relation complicates the representation of simple bicliques by drowning them in many concepts, as shown in example 3.4, bringing little to no information about the graph structure. Since we work on simple graphs, the reflexive relations are not part of the data. It follows that treating reflexivity is of little interest for the graph description, unless it simplifies the conceptual representation as in the (abc) triangle example.

	a	b	c	d
a			×	×
b			×	×
c	×	×		
d	×	×		

Figure 3.3 – A small graph context on the set $S = \{a, b, c, d\}$ describing a biclique $\{a, b\} \times \{c, d\}$.

Example 3.3. Considering again the reflexive graph context in Figure 3.1 (right), formal concepts can be listed as triplets, thus obtaining for instance $(\{a, b, c, d\}, \{\emptyset\}, \{e, f\})$ as triplet representation of $\{a, b, c, d, e, f\} \times \{e, f\}$, and $(\{a, b\}, \{c, d\}, \{e, f\})$ for $\{c, d, e, f\} \times \{a, b, e, f\}$. The triplet representation of all concepts of this example are listed in the third column of Table 3.1. We will see later that not all triplet representations are useful ; in that specific example, only one of them, $(\{a, b\}, \{c, d\}, \{e, f\})$, will be necessary to describe the graph using triplets.

Example 3.4. The context of Figure 3.3 describes a single biclique and consequently contains only one formal concept, $\{a, b\} \times \{c, d\}$. If the relation becomes reflexive, there are 13 more formal concepts, including for instance $\{a\} \times \{a, c, d\}$, $\{b\} \times \{b, c, d\}$, $\{d\} \times \{a, b, d\}$ and $\{b, d\} \times \{b, d\}$.

With this goal in mind, later sections will refine notable classes of triplets, based on properties such as maximality. This will ultimately lead to the definition of *triplet concepts*. Because we will apply triplets to graphs, next section details the link between triplets and graph theory.

3.2 Triplet concepts from a graph perspective

A graph context is the adjacency matrix of an undirected graph, where relations between objects encode edges between nodes.

If the graph is not reflexive, formal concepts derived from the graph context represent maximal bicliques in the graph. Because of the symmetry, each maximal

biclique (A, B) is represented by two formal concepts (A, B) and (B, A).

Example 3.5. The graph context in Figure 3.3 has exactly two formal concepts, both corresponding to the maximal biclique $\{a, b\} \times \{c, d\}$.
In Figure 3.2, we find also two formal concepts corresponding to the maximal star $\{b\} \times \{a, c\}$.
In Figure 3.1, the non-reflexive graph context (left) describes many maximal bicliques, e.g. $\{c, d\} \times \{a, b, e, f\}$, $\{e, f\} \times \{a, b, c, d\}$ and $\{a, b\} \times \{c, d, e, f\}$.

This section details the relation between triplets and graph motifs (subsection 3.2.1), from the general case to specific ones (subsections 3.2.2 and 3.2.3).

3.2.1 The overlapping biclique motif

The formal concepts derived from a reflexive graph context can describe a more general motif than bicliques, which we have named *overlapping biclique*.

Example 3.6. As shown in example 3.4, the reflexive graph context in Figure 3.3 has many formal concepts, one being the maximal biclique $\{a, b\} \times \{c, d\}$, while the others describe overlapping bicliques such as $\{b\} \times \{b, c, d\}$, $\{c\} \times \{a, b, c\}$.
In Figure 3.2, discussed in example 3.2,the two formal concepts are extended so that the hub is now present in both sets, i.e. $\{a, c\} \times \{b\}$ becomes $\{a, b, c\} \times \{b\}$.
In Figure 3.1, and as shown in Table 3.1, the reflexive graph context (right) describes only overlapping bicliques, notably because e and f are ubiquitous (i.e. they are linked to all other nodes), therefore all concepts contain them in both sets.

In the triplet representation (A, B, C) of the concept (X, Y), A is the set of nodes specific to the first set X, B the set of nodes belonging exclusively to the second set Y, and C the set of shared nodes. This is shown in Figure 3.4. Note that from a graph compression perspective, we use a single edge to represent the whole graph as for bicliques in Power Graph.

From a pure Power Graph perspective, C is a clique and (A, B) is a biclique.

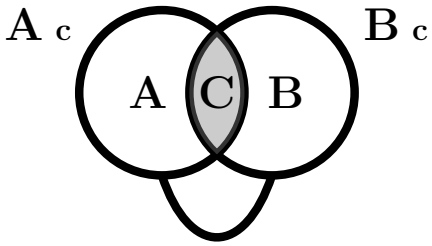


Figure 3.4 – Single-edge representation of a triplet concept (A, B, C) as two overlapping sets A_c and B_c of nodes linked to one another. $A = A_c \backslash B_c$ and $B = B_c \backslash A_c$. $C = A_c \cap B_c$ is the set of all nodes linked to all other nodes of the motif (ubiquitous nodes). A, B and C are disjoint sets.

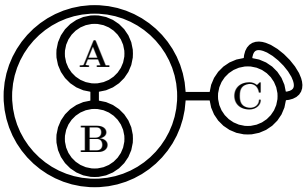


Figure 3.5 – Powergraph representation of a triplet concept (A, B, C), where C is a clique (hence the reflexive power edge) and (A, B) a biclique.

A triplet concept is thus a biclique between a clique and a biclique, as shown in Figure 3.5.

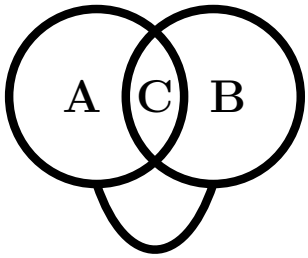
3.2.2 Special cases of overlapping bicliques

The overlapping biclique motif has three notable special cases, as shown in Figure 3.6, derived from the general case in Figure 3.6a.

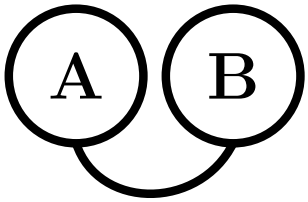
First, the biclique, shown in Figure 3.6b, corresponds to a formal concept without overlap, i.e. where $X \cap Y = \emptyset$, hence $C = \emptyset$.

Second, as shown in Figure 3.6c, when $X \subset Y$, hence $A = \emptyset$ or $B = \emptyset$, the motif becomes a nested biclique.

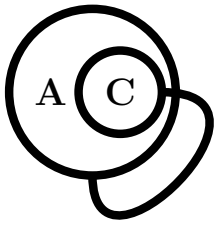
Finally, as shown in Figure 3.6d, when all nodes are linked to all others, i.e. $X = Y$, hence $A = B = \emptyset$, the motif is a simple clique.



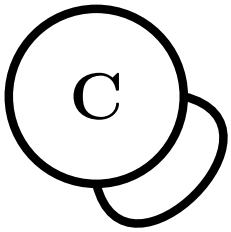
(a) The general representation of a triplet, where C is the set of ubiquitous nodes of the motif.



(b) A standard biclique, where $C = \emptyset$.



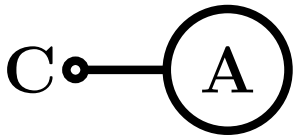
(c) A nested biclique where one set is included in the other. If $|C| = 1$, the triplet encodes a star. Reflexive links are necessary to this compact representation, but the underlying structure is to be understood as a simple graph.



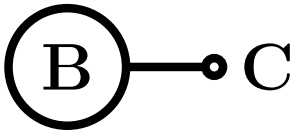
(d) A clique, where all nodes are ubiquitous in the motif, and therefore where $A = B = \emptyset$.

Figure 3.6 – Single-edge representations of overlapping biclique motif/triplet (A, B, C), along with its three special cases.

Example 3.7. As shown in example 3.6, the reflexive graph context in Figure 3.3 has one biclique $\{a, b\} \times \{c, d\}$.
In Table 3.1, listing concepts of Figure 3.1, we can find the two other special cases. The clique $\{b, d, e, f\} \times \{b, d, e, f\}$ containing four nodes ($|C| = 4$) and represented as the triplet $(\emptyset, \emptyset, \{b, d, e, f\})$. The nested biclique $\{b, c, d, e, f\} \times \{b, e, f\}$, represented as the triplet $(\{c, d\}, \emptyset, \{b, e, f\})$.



(a) The star motif, linking the nodes of A to an ubiquitous node in the C singleton.



(b) The star motif, linking the nodes of B to an ubiquitous node in the C singleton.

(c) The representation of a single edge, where A and B are singletons.

(d) The clique representation of a single edge, where $|C| = 2$. Reflexive links are necessary to this compact representation, but the underlying structure is to be understood as a simple graph.

Figure 3.7 – Single-edge representation of degenerated cases of the overlapping biclique motif/triplet (A, B, C).

3.2.3 Degenerated cases of the overlapping biclique

The overlapping biclique motif has two notable degenerated cases, as shown in Figure 3.7.

The star motif is a degenerated biclique where one set is a singleton (the *hub node*). It can be encoded using the classical biclique $A \times B \times \emptyset$, or almost any combination of two of the three sets. The only restriction is that C cannot represent a non-singleton set.

The edge motif is a degenerated star motif where both sets are singletons. As a consequence, C can be used by both sets, and, as shown in Figure 3.7d, *for* both sets.

This introduction on triplets has provided the necessary background for a complete study of triplet concepts, a sets of triplets than can be considered as an extension of formal concepts.

3.3 Triplet maximality

In Formal Concept Analysis, a formal concept (X, Y) is defined as a maximal all-ones submatrix in the incidence matrix of the relation, in the strong sense that every element that is not in one set is not connected to at least one element in the other set. Formal concepts are thus node-maximal with respect to inclusion. This property is necessary but not sufficient to build triplet concepts.

Examples 3.8 and 3.9 provide an illustration of the limits of node-maximality in the context of graphs.

Example 3.8. Taking the graph context in Figure 3.1 and the triplet concepts derived from it, listed in Table 3.1, we can note that the canonical triplet $T_7 = (\{a, b\}, \{c, d\}\{e, f\})$ (line 7) or the triplet $T_3 = (\{c, d\}, \{a, b\}, \{e, f\})$ (line 3) stricly contain the edges covered by each of the other triplets, including $(\{a, b, c, d\}, \emptyset, \{e, f\})$ (line 1), $(\{a, b\}, \emptyset, \{d, e, f\})$ (line 2) and $(\{c, d\}, \emptyset, \{a, e, f\})$ (line 6). T_3 and T_7 are thus maximal with respect to covered edges.
From a graph perspective, $T_1 = (\{a, b, c, d\}, \emptyset, \{e, f\})$ describes the biclique $\{a, b, c, d\} \times \{e, f\}$ and clique $\{e, f\}$, while $T_7 = (\{a, b\}, \{c, d\}, \{e, f\})$ describes bicliques $\{a, b\} \times \{e, f\}$ and $\{c, d\} \times \{e, f\}$, and the clique $\{e, f\}$, thus covering edges of T_1, but also the biclique $\{a, b\} \times \{c, d\}$, making T_7 a larger motif than T_1.

Example 3.9. Considering Figure 3.8, the following triplets are *node-maximal*: $(\emptyset, \emptyset, \{e, k, l\})$, $(\emptyset, \emptyset, \{f, k, l\})$ and $(\{c, h, i, j, l\}, \emptyset, \{f, k\})$. Indeed in each case, no other node can be added to one of the sets A, B or C, unless reducing one of the other set.
However, these triplets are not *edge-maximal* with respect to inclusion: they are all covered by the triplet $(\{c, h, i, j, l\}, \{d, e, f\}, \{k\})$, which is both node- and edge-maximal.

A correct notion of maximality must therefore refer only to maximality with respect to inclusions of the set of edges represented by concepts. This is the purpose of definition 3.2.

Definition 3.2. *Edge-maximal triplet* A triplet T_1 covering edges E_1 is edge-maximal iff there is no triplet T_2 covering edges E_2 so that $E_1 \subset E_2$ over the same graph context.

We will often use the term maximal triplet to refer to edge-maximal triplets in the rest of this document. Note that the two maximal triplets of examples 3.8 and 3.9 cover exactly the same edges, and are both maximal. It follows that a triplet can be maximal, yet can be written in multiple ways. This is why we will introduce the notion of canonical writing of triplets in section 3.4. Together with edge-maximality, this will lead to the formal definition of triplet concepts in section 3.5.

Due to this equivalence between several representations, determining if a given triplet is edge-maximal is not a trivial task. A later section (3.6) introduces a first restriction on triplets, *admissibility*, which is a necessary condition for edge-maximal triplets. This will ease the derivation of triplet concepts from standard formal concepts detailed in section 3.8.

	a	b	c	d	e	f	g	h	i	j	k	l
a				×	×		×	×	×			
b				×	×		×	×	×			
c				×	×	×	×	×	×	×	×	
d	×	×	×				×	×	×	×	×	×
e	×	×	×				×	×	×	×	×	×
f			×					×	×	×	×	×
g	×	×	×	×	×			×	×			
h	×	×	×	×	×	×	×		×	×	×	
i	×	×	×	×	×	×	×	×		×	×	
j			×	×	×	×		×	×		×	
k			×	×	×	×		×	×	×		×
l				×	×	×					×	

Figure 3.8 – Formal context of a dense graph, yielding a total of 93 formal concepts. Two of the maximal graph cliques are intersecting : $\{d, g, h, i\}$ and $\{c, h, i, j, k\}$.

3.4 Triplet canonical form

Previous sections defined triplets and edge-maximal triplets. However, many maximal triplets are edge-equivalent: they cover the exact same motif. This sec-

tion defines a *normal form* of triplets, per definition 3.3. Proposition 3.1 proves it *canonical*, ensuring the unique representation of any triplet concept. The computation of the normal form of a given triplet is shown in Algorithm 5.

As shown in previous sections, for instance in example 3.8, an edge-maximal triplet can be written in different forms. For instance, $(\{a\}, \{b\}, \emptyset)$, $(\{b\}, \{a\}, \emptyset)$ and $(\emptyset, \emptyset, \{a, b\})$ all describe triplet concepts of same cover. So are $(\{a, b\}, \emptyset, \{c, d\})$, $(\emptyset, \{a, b\}, \{c, d\})$ and $(\{a, b\}, \{c\}, \{d\})$. More generally, A and B can be safely inverted in any triplet, and one node of C can be moved in A (or B) if $A = \emptyset$ ($B = \emptyset$).

The normal form definition is therefore motivated by the following observations. First, if the A set or the B set is a singleton, then the corresponding node is linked to all other nodes of the triplet, therefore is ubiquitous. Because all ubiquitous nodes can belong to C, the normal form is enforcing it. It prevents them to have multiple acceptable places when A or B are empty. Second, to break symmetry over A and B, the minimal element must always be in A or C. One of the direct consequence is that A can be empty only if B is empty ($A = \emptyset \rightarrow B = \emptyset$).

Let us now define the *normal form* of a triplet:

Definition 3.3. *Normal form of a triplet* A triplet $T = (A, B, C)$ is written in normal form if and only if:

1. $|A| \neq 1$ and $|B| \neq 1$
2. $min(A \cup B) = min(A)$

For $min(X)$ being the minimal element of X in lexicographical order.

Proposition 3.1 states that, by the sole use of these two properties, the normal form identifies each triplet concept in a unique, collision-safe way. As a consequence, Algorithm 5 describes the computation of the canonical representation of a triplet.

Proposition 3.1. *The normal form defined in 3.3 is canonical: two triplets covering the same edges have the same normal form. Conversely, if two triplets do not cover the same edges, they have a different normal form.*

Proof. Let T_1 and T_2 be two triplets of respective canonical form (A, B, C) and (D, E, F), both covering the same edges. Since both nodes and edges are the same,

so are ubiquitous nodes, hence $C = F$. It follows that $A \cup B = D \cup E$, hence by the canonical form, $min(A) = min(D)$. All edges of T_1 linking $min(A)$ to $b \in B$ must also be covered by T_2, therefore $B = E$, which implies that $A = D$. □

Example 3.10. Let us consider the list of triplets in Table 3.1 (right column). Some of them are already in normal form, including $T_1 = (\{a, b, c, d\}, \emptyset, \{e, f\})$, $T_4 = (\emptyset, \emptyset, \{a, d, e, f\})$ and $T_7 = (\{a, b\}, \{c, d\}, \{e, f\})$. On the other hand, $T_{16} = (\emptyset, \{a, b, c, d\}, \{e, f\})$, $T_8 = (\emptyset, \{a, b\}, \{d, e, f\})$ and $T_3 = (\{c, d\}, \{a, b\}, \{e, f\})$ are not in normal form. Using Algorithm 5, it is easy to derive it, thus obtaining, respectively, T_1, T_2 and T_7.
In this table, all triplet representation where already ensuring that ubiquitous nodes were in the C set, hence the writing of $(\{a, b, c, d\}, \emptyset, \{e, f\})$ instead of for instance $(\{a, b, c, d\}, \{e\}, \{f\})$, already fulfilling the condition of non-singletons A and B.
Finally, note that one triplet, T_7, is both edge-maximal and in normal form.

Algorithm 5 An algorithm to get the canonical normal form of a given triplet.

Require: Triplet (A, B, C)
Ensure: (A, B, C) is in normal form
1: **if** $|A| = 1$ **then**
2: $C \leftarrow C \cup A$
3: $A \leftarrow \emptyset$
4: **end if**
5: **if** $|B| = 1$ **then**
6: $C \leftarrow C \cup B$
7: $B \leftarrow \emptyset$
8: **end if**
9: **if** $(A \cup B) \neq \emptyset$ and $min(A \cup B) = min(B)$ **then**
10: $A, B \leftarrow B, A$
11: **end if**

3.4.1 Canonical form of special and degenerated cases

As shown in sections 3.2.2 and 3.2.3, there are multiple special and degenerated cases of triplet concept/overlapping biclique. Let us review the canonical form of

these cases.

Triplet special cases

Obviously, a biclique is represented as $(A, B, \emptyset)$ and its symmetry is broken by having the minimal element in A. A clique is represented the single obvious way: $(\emptyset, \emptyset, C)$. A nested biclique (Figure 3.6c) is represented by $(A, \emptyset, C)$.

Triplet degenerated cases

A star, of central node c, as shown in figure 3.7a, is represented as $(A, \emptyset, \{c\})$, because c being ubiquitous in the motif, and because A cannot be empty while B is not. In the particular case where there are only two nodes (that is, an edge as per Figure 3.7c), both are relocated to C, since both nodes are ubiquitous: a single edge (a, b) is therefore represented as $(\emptyset, \emptyset, \{a, b\})$, i.e. a clique of two elements, as per Figure 3.7d.

The canonical form enables a unique representation of triplets. As shown in the next section, it is necessary for a precise definition of triplet concepts.

3.4.2 Canonical form and triplet equivalences

A triplet can be written in many ways. The exact number of existing equivalent triplets depends of the number of node in each set. In most cases, the symmetry between A and B yields two different equivalent writings. Also, when A and B are empty sets (or singletons), the number of equivalent triplets is quadratic with respect to the size of the C set: it equals the number of ways to distribute nodes of C in A and B. For instance, the total number of equivalent writings for a triplet $(\emptyset, \emptyset, C)$ is $1 + |C| \times (|C| + 1)$. When exactly one of the sets A, B is not a singleton, the number of equivalent writings for a triplet is $2 + 2 \times |C|$. Also, the symmetry between A and B produces a new writing, hence there is always at least 2 possible ways to write a triplet if $A \neq B$: (A, B, C) and (B, A, C).

Example 3.11. The triplet $(\emptyset, \emptyset, \{a, b\})$ ($|C| = 2$) is equivalent to 6 others: $(\{a\}, \emptyset, \{b\})$, $(\emptyset, \{a\}, \{b\})$, $(\{b\}, \emptyset, \{a\})$, $(\emptyset, \{b\}, \{a\})$, $(\{a\}, \{b\}, \emptyset)$ and $(\{b\}, \{a\}, \emptyset)$.
The triplet $(\emptyset, \{a, b\}, \{c, d\})$ ($|B| = 2$, $|C| = 2$) is equivalent to 5 others: $(\{a, b\}, \{c\}, \{d\})$, $(\{a, b\}, \{d\}, \{c\})$, $(\{c\}, \{a, b\}, \{d\})$, $(\{d\}, \{a, b\}, \{c\})$ and $(\{a, b\}, \emptyset, \{c, d\})$.

3.5 Triplet concept definition

This section integrates the notion on triplet seen so far to introduce *triplet concepts*. We provide first a definition, then examples of triplet concepts for each of the three exposed graph contexts in examples 3.12, 3.13 and 3.14.

Definition 3.4. *Triplet concept*
A triplet concept is an edge-maximal triplet in normal form.

Example 3.12. Let us consider the list of triplets issued from Figure 3.1 in Table 3.1 (right column). The only triplet that is both edge-maximal and in normal form is T_7. It is therefore a triplet concept. Note that T_7 covers the full graph context. Therefore, no other triplet concept exists, since such a triplet concept cannot cover more edges than T_7.

Example 3.13. Only one triplet covers the graph context of Figure 3.2, namely $T_1 = (\{a, c\}, \emptyset, \{b\})$. It is therefore edge-maximal. T is also written in normal form, therefore it is a triplet concept.

Example 3.14. The graph context in Figure 3.8 issues 8 triplets concepts:

$(\{a, b, c, d, e, f, g, j, k\}, \emptyset, \{h, i\})$

$(\{d, e, f, g, j, k\}, \emptyset, \{c, h, i\})$

$(\{a, b, c, g, j, k\}, \{d, e\}, \{h, i\})$

$(\{g, j, k\}, \{d, e\}, \{c, h, i\})$

$(\{d, e, f\}, \emptyset, \{c, h, i, j, k\})$

$(\{c, h, i, j, l\}, \{d, e, f\}, \{k\})$

$(\{a, b, c\}, \{d, e\}, \{g, h, i\})$

$(\{a, b, c, g, h, i, j, k, l\}, \{d, e\}, \emptyset)$

92 of the 93 formal concepts are covered by the 7 first triplet concepts. The remaining formal concept $\{a, b, c, g, h, i, j, k, l\} \times \{d, e\}$ is also a triplet concept with the clique part empty.

Next sections will detail the theoretical study of triplet concepts. The remaining sections will address more practical aspects of triplet concepts, such as mining methods and ordering.

3.6 Triplet admissibility

Section 3.3 defined the *edge-maximality* of triplets. For a given graph context, it is possible to determine its derived edge-maximal triplets, but a naive implementation of the verification of this property could have a significant cost. This section provides four simple propositions, globally discriminating triplets that are not edge-maximal without relying on a complete comparison of all triplets. These four properties will be used for later triplet concept search implementations.

Let us start with an example (3.15) showing how some triplets may be pruned from the search space of maximal triplets.

Example 3.15. We consider the graph context in Figure 3.1 and the triplet $T_1 = (\{a, b, c, d\}, \emptyset, \{e, f\})$ derived from it (line 1 in Table 3.1). Since node a and b are both linked to c and d, $\{a, b\} \times \{c, d\}$ is a biclique (cf Figure 3.1), and B is empty in T_1. It means that the biclique can be added to T_1, by transferring $\{c, d\}$ from A to B. Once again T_{max} is covering strictly more edges than T_1. Hence, T_1 is not maximal.
Let us now consider the triplet $T_2 = (A, B, C) = (\{a, b\}, \emptyset, \{d, e, f\})$ (line 2). This triplet can be written in different forms, such as $T_2' = (\{a, b\}, \{d\}, \{e, f\})$. T_2 is not edge-maximal, since it is fully covered by triplet $T_{max} = (\{a, b\}, \{c, d\}, \{e, f\})$ (line 10). More precisely, T_2' (and therefore T_2) is not edge-maximal because c could have been added to B (it is connected to all elements of $A \cup C$).

More generally, the edge-maximability of a given triplet may be refuted using nodes external to the subgraph context associated to the triplet. Because adding a new element to a triplet implies that new edges are covered, it follows that if a node can be added to a triplet, the triplet cannot be edge-maximal. The most basic case, handled by proposition 3.2, is an outsider node e connected to all nodes of the triplet, in which case the triplet is by definition non-edge-maximal, since e could be added to C. Similarly, if e were connected to all nodes of $A \cup C$, it could be added to B. Similarly, one can observe that a node in the A set of a triplet concept cannot be linked to all other nodes. As per definition 3.3, such a node has to belong to C in the canonical form of the triplet.

Proposition 3.2. *Let $T = (A, B, C)$ be a triplet of the graph (V, E). If there exists an element of $V \setminus B \cup C$ in relation with all elements of $A \cup C$, or if there exists an element of $V \setminus A \cup C$ in relation with all elements of $B \cup C$, then T is not edge-maximal.*

Proof. Let $T = (A, B, C)$ be a triplet, $N = A \cup B \cup C$, and $e \notin N$ an element in relation with all elements of $A \cup C$. Then $T_{better} = (A, B \cup \{e\}, C)$ is a triplet covering strictly more edges than T. Same reasoning applies for e in relation with all elements of $B \cup C$. If e is in relation with all elements of N, then $(\{A\}, \{B\}, \{C \cup \{e\}\})$ is a triplet covering strictly more edges than T. If A is a clique and not a

singleton, then $T_{better} = (\emptyset, B, C \cup A)$ covers strictly more edges than T. If A is a singleton, $T_{equiv} = (\emptyset, B, C \cup A)$ is equivalent to T (but in normal form). Same reasoning applies for B. □

Proposition 3.3 enables to discriminate some triplets following the same exclusion logic, but whose representation precludes the use of proposition 3.2. In example 3.15 for instance, T_2 was not fulfilling the conditions of proposition 3.2, but the equivalent triplet T_2' was, because it was easy to add a node in B. Proposition 3.3 allows to match triplets such as T_2.

Proposition 3.3. *Let $T = (A, B, C)$ be a triplet of the graph (V, E), and $N = A \cup B \cup C$, such that the size of A or B is less than 2. If there exists an element $e \in E \setminus N$, which is in relation with all elements of N except one element of C, then T is not a maximal triplet.*

Proof. Let $T = (A, B, C)$ be a triplet, and $N = A \cup B \cup C$, such that $A = \emptyset$, $f \in C$, and $e \notin N$ an element in relation with all elements of $N \setminus \{f\}$. Then $T_{better} = (\{e, f\}, B, C \setminus \{f\})$ is a triplet covering strictly more edges than T. If A is a singleton, the node $n \in A$ can be moved to C (it is by definition linked to all other nodes of N), and since A becomes empty, proposition 3.2 applies. Same reasoning applies with B instead of A. □

Proposition 3.4 handles the T_1 case described in example 3.15, where the A set is a biclique whose two node sets can be placed in A and B in order to achieve a greater cover of the graph context.

Proposition 3.4. *Let $T = (A, B, C)$ be a triplet of the graph (V, E), such that $B = \emptyset$ (or $A = \emptyset$). If a partition $P = \{P_1, P_2\}$ of A (or B) exists, such that all elements of P_1 are in relation with all elements of P_2, then T is not edge-maximal.*

Proof. Let $T = (A, B, C)$ be a triplet, such that $B = \emptyset$. A is partitioned by $P = \{P_1, P_2\}$, and for $x \in P_1 \quad y \in P_2$ we have $rel(x, y)$. $T_{better} = (P_1, P_2, C)$ is a triplet covering strictly more edges than T. Same reasoning applies by exchanging A and B. □

From these propositions, we define the *admissible triplets*.

Definition 3.5. *Admissible triplet* Any triplet that does not match propositions 3.2, 3.3 or 3.4 is an *admissible triplet*, i.e. admissible for edge-maximality.

We have defined triplet concepts as a particular representation of edge-maximal triplets, which represent maximal overlapping bicliques in the underlying graph. The set of edge-maximal triplets has yet to be enumerated. A first approach is to try to relate their enumeration to the enumeration of formal concepts in the (reflexive) context graph. It results in a first algorithm, presented in the next section.

3.7 Triplet concept ordering

The triplets concepts, as shown in previous sections, are themselves a particular representation of a subset of the formal concepts issued from the reflexive graph context. This section aims to show that the set of triplet concepts are partially ordered, likewise to formal concepts, but cannot simply be represented as a (semi) lattice.

Definition 3.6 proposes to use a natural inclusion relation between triplet concepts based on the inclusion of the sets, which is proved to be a (partial) order. We give then a formulation of the meet and the join operators derived from their calculation on formal concepts. Example 3.16 provides an example of a triplet set whose structure, shown in Figure 3.9, is not a lattice with respect to the definitions of meet and join.

Definition 3.6. Let $T_1 = (A_1, B_1, C_1)$ and $T_2 = (A_2, B_2, C_2)$ be two triplet concepts. We say that T_1 is lower than or equal to T_2, denoted $(A_1, B_1, C_1) \leq (A_2, B_2, C_2)$, if and only if $A_1 \subseteq A_2$, $B_1 \supseteq B_2$ and $C_1 \supseteq C_2$.

Proposition 3.5. *Let $\mathcal{T}$ be the set of triplet concepts derived from a graph context. The binary relation $\leq$ over triplets is a partial order on (T).*

Proof. For any $T_x, T_y, T_z \in \mathcal{T}$, $T_x = (A_x, B_x, C_x)$, $T_y = (A_y, B_y, C_y)$ and $T_z = (A_z, B_z, C_z)$, we have:

— $(A_x, B_x, C_x) \leq (A_x, B_x, C_x)$ since $A_x \subseteq Ax, B_x \supseteq B_x, C_x \supseteq C_x$ (reflexivity)

— $A_x \subseteq A_y$ and $A_x \supseteq A_y$ is only possible if $A_x = A_y$. Same reasoning for B and C sets (antisymmetry)

— $T_x \leq T_y$ and $T_y \leq T_z$ imply that $A_x \subseteq A_y \subseteq A_z$, therefore $A_x \subseteq A_z$. Same reasoning applies for B and C set, thus $T_x \leq T_z$ (transitivity).

□

Now, let $T_1 = (A_1, B_1, C_1)$ and $T_2 = (A_2, B_2, C_2)$ be two triplet concepts. They can be written as formal concepts on the reflexive context graph: we get $F_1 = (A_1 \cup C_1, B_1 \cup C_1)$ and $F_2 = (A_2 \cup C_2, B_2 \cup C_2)$.

The meet of the two formal concepts F_1 and F_2 is therefore $\bigwedge(F_1, F_2) = ((A_1 \cup C_1) \cap (A_2 \cup C_2), ((B_1 \cup C_1) \cup (B_2 \cup C_2))'')$. By construction, $((B_1 \cup C_1) \cup (B_2 \cup C_2))'')$ is equivalent to $((A_1 \cup C_1) \cap (A_2 \cup C_2))'$, thus, for $\beta = (A_1 \cup C_1) \cap (A_2 \cup C_2)$, we get $\bigwedge(F_1, F_2) = (\beta \setminus \beta', \beta' \setminus \beta, \beta \cap \beta')$. In a similar way, we get as their join $\bigvee(T_1, T_2) = (\alpha \setminus \alpha', \alpha' \setminus \alpha, \alpha \cap \alpha')$ for $\alpha = (A_1 \cup C_1) \cap (A_2 \cup C_2)$.

The largest triplet concept per these definitions is a triplet concept $(A, \emptyset, U)$ containing all nodes of the graph in its A set, except the globally ubiquitous nodes U. If $U = \emptyset$, this maximal triplet is a dummy triplet covering no edges. Conversely, the smallest triplet concept is a dummy triplet $(\emptyset, V, V)$. However, those definitions of meet and join do not suffice to constitute a triplet concept lattice: as shown in example 3.16, the join and meet are not necessarily triplet concepts.

Example 3.16. Let us consider the set of 8 triplet concepts derived from the graph context in Figure 3.8, containing among others $T_1 = (\{a, b, c\}, \{d, e\}, \{g, h, i\})$, $T_2 = (\{a, b, c, g, j, k\}, \{d, e\}, \{h, i\})$, $T_3 = (\{c, h, i, j, l\}, \{d, e, f\}, \{k\})$ and $T_4 = (\{d, e, f\}, \emptyset, \{c, h, i, j, k\})$. According to definition 3.6, $T_1 \leq T_2$, because $\{a, b, c\} < \{a, b, c, g, j, k\}$, $\{d, e\} = \{d, e\}$ and $\{g, h, i\} > \{h, i\}$. T_3 is not comparable to T_1 and T_2. The graph of triplet concepts where larger triplet concept are above is given in Figure 3.9. However, it is not a lattice. According to Figure 3.9, the join of T_4 and T_3 it is the supremum, $(\{a, b, c, d, e, f, g, h, i, j, k, l\}, \emptyset, \emptyset)$. On the other hand, according to the formulation of the join, it is the non edge-maximal triplet $(\{c, d, e, f, h, i, j, l\}, \emptyset, \{k\})$. Indeed, this last triplet is covered by T_3.

Triplet Lattice

It follows that the triplet concepts do not form a lattice. It seems that the condition used to break the symmetry of triplet concepts precludes the lattice structure.

As shown in example 3.17, by using the symmetric (B, A, C) instead of the triplet concept (A, B, C), one can obtain a better characterization for the meet and the join.

Example 3.17. Starting again from example 3.16, but replacing $(\{def\}, \emptyset, \{chijk\})$ by its non-canonical representation $(\emptyset, \{def\}, \{chijk\})$ in the set of triplet concepts, the two definitions correctly yield $(\{chijl\}, \{def\}, \{k\})$ as the join of $(\{def\}, \emptyset, \{chijk\})$ and $(\{chijl\}, \{def\}, \{k\})$. Note that it also correctly identifies a triplet concept as the join of all pairs of triplets in the graph of inclusion. In other words, the overall graph structure in Figure 3.9 is a semi-lattice of supremum $(\{abcdefghijkl\}, \emptyset, \emptyset)$ when using the symmetric of a triplet instead of the triplet itself.

Adding symmetries

The graph of inclusion can be completed with symmetries by adding $\overline{\mathcal{T}}$, the set of symmetric triplets (B, A, C), and reversing the direction of inclusions for the cliques of $\overline{\mathcal{T}}$: $(A_1, B_1, C_1) \leq (A_2, B_2, C_2)$ if $A_1 \subseteq A_2$, $B_1 \supseteq B_2$ and $C_1 \subseteq C_2$ (note that only the clique relation changes). The resulting graph of inclusion of $\mathcal{T} \cup \overline{\mathcal{T}}$ is given in Figure 3.10. For edges linking an element in $\mathcal{T}$ and an element in $\overline{\mathcal{T}}$ (bold edges in the figure), the relation of inclusion between cliques can appear in both directions. This representation can explain the observation in example 3.17, as shown in example 3.18.

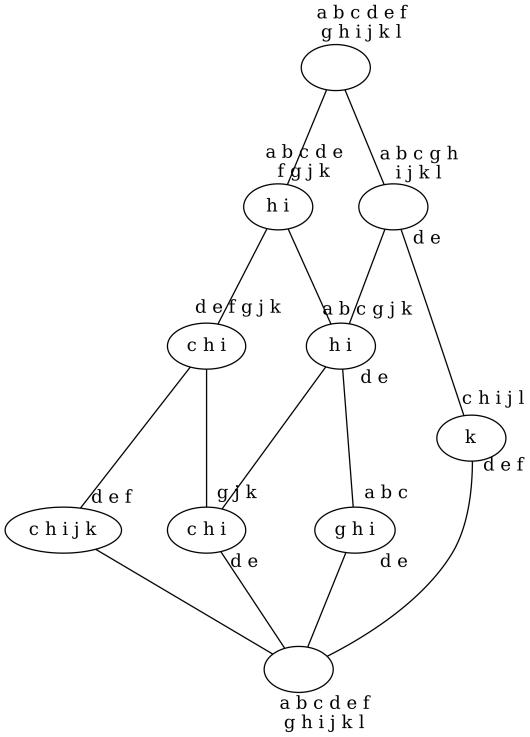


Figure 3.9 – Graph of inclusion of triplet concepts issued from the graph context in Figure 3.8. A is shown as the upper label of each node, B as the lower label, and C as the center label. The supremum is the dummy triplet covering no edge. The infimum is not a valid triplet, but is included for clarity.

Example 3.18. Let us now consider the graph of inclusion with symmetries in Figure 3.10. It appears that the join of triplets $(\emptyset, \{def\}, \{chijk\})$ and $(\{chijl\}, \{def\}, \{k\})$ is in this representation $(\{chijl\}, \{def\}, \{k\})$. This is consistent with the observation of example 3.17, where the use of $(\emptyset, \{def\}, \{chijk\})$ (rightmost dashed node on the seventh line) in place of $(\{def\}, \emptyset, \{chijk\})$ (leftmost node on the fifth line) enables to find a join that is also a triplet concept: $(\{chijl\}, \{def\}, \{k\})$.

This suggests that the canonical writing of triplets may need to be relaxed or redefined in order to get a lattice structure. This is left as an open problem.

Following sections will explore the triplet concept enumeration problem and its relation to MCE.

3.8 Triplets as a combination of formal concepts

We start with a proposition allowing, together with the admissibility constraints, the definition of admissible triplets as a combination of formal concepts issued from the non-reflexive graph context. We then propose an enumeration (Algorithm 6) derived from this proposition. An ASP implementation of the proposition, used with admissibility constraints, is detailed in section 3.9.4.

The intuition behind proposition 3.6 and subsequent algorithm 6 is that a triplet concept (A, B, C) is composed of three formal concepts derived from the non-reflexive formal context: $(A, B \cup C)$, $(B, A \cup C)$ and $(C, A \cup B)$. The proposition details the exact properties these formal concepts must fulfill.

Proposition 3.6. *Let A, B, C be three disjoint sets of vertices from a non reflexive graph context G. Using the standard derivation operation of FCA on this context graph, the following property holds:*

The triplet of disjointed non empty sets $T = (A, B, C)$ is a triplet if and only if C is a clique in the induced subgraph $G(A \cup B \cup C)$, and there exist three sets D, E and F disjoint of sets A, B and C and three concepts (A'', A'), (B'', B'), and (C'', C') such that:

$$A' = B \cup C \cup D \tag{3.2}$$
$$B' = A \cup C \cup E \tag{3.3}$$
$$C' = A \cup B \cup F \tag{3.4}$$
$$D \cap F = E \cap F = \emptyset \tag{3.5}$$
$$A'' = A \tag{3.6}$$
$$B'' = B \tag{3.7}$$
$$C'' \supseteq C \tag{3.8}$$

Proof. Assume T is an edge-maximal triplet. First, we need to prove that $A'' = A$ and $B'' = B$. Let vertex $s \in A'' \notin A$. Because $s \in A''$, s is linked to all vertices

of $A' = B \cup C \cup D$. Therefore, $(A \cup \{s\}, B, C)$ is a triplet, thus T is not maximal (Proposition 3.2). As a special case, if s is also linked to all elements of A, then $(A, B, C \cup \{s\})$ is also an edge-maximal triplet. Same reasoning applies if vertex $s \in B'' \notin B$.

For any triplet (A, B, C), $A \times (B \cup C)$, $B \times (A \cup C)$, and $C \times (A \cup B)$ are bicliques of the graph. This implies that $B \cup C \subseteq A'$, $A \cup C \subseteq B'$, and $A \cup B \subseteq C'$ and the first three equations 3.2 to 3.4 are valid. Assume that there exists an element $x \in D \cap F$. In such a case $x \in A'$ and $x \in C'$. This implies that $(A \cup C \cup \{x\}, B \cup C)$ is a valid biclique. Since the biclique $(A \cup C, B \cup C)$ is maximal, x should belong to A or C, a contradiction.

For the reciprocal, assume concepts (A, A'), (B, B'), and (C, C') exist with the properties in equations. Then $(A \cup C, B \cup C)$ is a biclique of the graph. One has to check that this biclique is maximal. If $(A \cup C \cup x, B \cup C)$ is a biclique, $x \notin A \cup C$, then $x \in E$ from the definition of B' in concept (B, B'). Now if $x \in B$, then x appears on both sides of $(A \cup C \cup \{x\}, B \cup C)$ and C is not a maximal clique on $G(A \cup B \cup C)$. Thus $x \notin B$. One can deduce $x \in F$ from the definition of C' in concept (C, C'). The conclusion is that $x \in E \cap F$, a contradiction. The same reasoning applies if one adds an element to the right of the biclique. □

One should note that the three concepts chosen for sets A, B and C are not necessarily all different. There is in fact a specific case to study. When there is no ubiquitous node in the graph, the supremum formal concept is of the form $(V, \emptyset)$ (V being the nodes of the graph), and the infimum formal concept is of the form $(\emptyset, V)$. In that case, if the infimum is used to determine B and C, then A will be associated to the supremum (others concepts will not populate A with all nodes, and therefore will not be admissible). In such a case, we obtain the triplet $(V, \emptyset, \emptyset)$, which is not an edge-maximal triplet, since it covers no edge. However, that triplet is, as it was described in section 3.7, the supremum of the triplet semi-lattice. From proposition 3.6 can be devised a brute-force algorithm to compute the triplet concepts, and an ASP encoding exploring the concept combinations and yielding one model for each triplet. Algorithm 6 presents the former, and section 3.9 the latter.

Algorithm 6 A brute-force algorithm to compute triplets from formal concepts, relying on proposition 3.6. Obtaining the normal form of yielded triplets (thus the triplet concepts) is a simple post-process. An ASP implementation of the proposition is presented in section 3.9.

Require: Nodes $\mathcal{N}$, formal concepts $\mathcal{C}$
Ensure: Computation of edge-maximal triplets

for all $A \subset \mathcal{N}$ **do**
  for all $B \subset \mathcal{N} \mid A \subseteq B'$ **do**
    for all $C \subset \mathcal{N} \mid A \cup B \subseteq C'$ *and clique*(C) **do**
      $D \leftarrow A' \setminus (B \cup C)$
      $E \leftarrow B' \setminus (A \cup C)$
      $F \leftarrow C' \setminus (A \cup B)$
      if $D \cap E = E \cap F = \emptyset$ *and* $B \cup C \subseteq A'$ *and* $A \cup C \subseteq B'$ **then**
        for all maximal clique D of C **do**
          if (A, B, D) is admissible **then**
            yield (A, B, D)
          end if
        end for
      end if
    end for
  end for
end for

3.9 Triplet concept enumeration

The enumeration of triplet concepts problem is defined as follows: given a graph context, yield all triplet concepts in that context. More generally, the enumeration of edge-maximal triplets is the most complex task, since the determination of their normal form (i.e. getting canonical form of triplets) can be managed (as shown in section 3.4, with Algorithm 5), as a simple post-process.

This section will explore approaches to the resolution of that problem. First, section 3.9.1 addresses the encoding of the admissibility conditions detailed in section 3.6.

The input graph context can be represented in multiple ways: the set of relations in the graph context, the set of formal concepts issued from it, and the set of formal concepts issued from the reflexive graph context are three equivalent representations of the same dataset. Note that the theoretical ground for generation

of triplets from the formal concepts issued from the non-reflexive graph context is given in section 3.8. This section will encode the search for admissible triplets for each of these representations. Sections 3.9.2 and 3.9.3 detail two different ways to enumerate the triplets, starting respectively from the reflexive graph context, and from the reflexive formal concept set. Section 3.9.4 proposes an implementation relying on the combination of formal concepts to get triplets concepts, as detailed in section 3.8. Final section 3.9.5 compares the different implementations.

The three described methods, aim to produce in a fairly straightforward way a sufficient set of triplets (in the worst case, all items of the right column of Table 3.1). When solved together with the encoding of admissibility conditions presented in section 3.9.1, it happens that most if not all solution triplets are edge-maximal. Admissibility is thus a property strongly related to edge-maximality, although we have not been able so far to prove an equivalence between both properties. For completeness, a filtering of non edge-maximal triplets is theoretically necessary to discard all admissible triplets that are not edge-maximal. This reduction from admissible triplets to triplet concepts requires a costly comparison of triplets' edge covers. It is straightforward, thus not detailed here. However, we should note that we never encountered a test case where this costly post-process was needed: the admissible triplets were, once written in canonical form, also the triplet concepts.

3.9.1 Encoding the admissibility conditions

As presented in section 3.6, it is possible to discard non-maximal triplets by testing some of their properties and relations with outsider elements.

Encoding 3.1 implements these conditions, invalidating the current model if the described triplet is not admissible. It is to be grounded with `a/1`, `b/1` and `c/1` atoms encoding the triplet (A, B, C), and with `rel/2` encoding the symmetric relation between elements.

Prior definitions

Lines 2 to 10 are defining helpers for later constraints. `outside/1` (line 2) is defining the elements inside and outside the triplet. Lines 3 to 5 are reducing the

```
% Prior definitions.
outside(X) :- rel(X,_) ; not a(X) ; not b(X) ; not c(X).
outside_to_a(X) :- outside(X) ; rel(X,Y): a(Y).
outside_to_b(X) :- outside(X) ; rel(X,Y): b(Y).
outside_to_c(X) :- outside(X) ; rel(X,Y): c(Y).
outside_to_c_but_1(X) :- outside(X) ; 1{ not rel(X,C): c(C) }1.
a_or_b_empty_set      :- not a(_).
a_or_b_empty_set      :- not b(_).
a_or_b_singleton      :- 1 { a(_) } 1.
a_or_b_singleton      :- 1 { b(_) } 1.

% If an outsider element is linked to all but the elements of A or B,
%  the concept is not admissible.
:- outside_to_a(X) ; outside_to_c(X).
:- outside_to_b(X) ; outside_to_c(X).
% A node of A (B) cannot be linked to all other nodes of A (B).
:- a(X) ; rel(X,Y): a(Y), X!=Y.
:- b(X) ; rel(X,Y): b(Y), X!=Y.

% If A or B is emptyset or singleton, and an outsider element
%  linked to A U B U C but one of C, the concept is not admissible.
:- a_or_b_empty_set ; outside_to_c_but_1(X)
    ; outside_to_a(X) ; outside_to_b(X).
:- a_or_b_singleton ; outside_to_c_but_1(X)
    ; outside_to_a(X) ; outside_to_b(X).

% When B is empty, the complementary graph of A
%  shouldn't have 2 or more connected components.
same_cc(M) :- M=#min{X:a(X)}.
same_cc(X) :- a(X) ; not rel(X,Y) ; same_cc(Y).
:- a(X) ; not same_cc(X) ; not b(_).
```

Encoding 3.1 – Implementation of the admissibility conditions for triplet edge-maximality.

set of outsiders to those linked to all elements of sets A, B or C. Line 6 computes the outsiders connected to all but one element of C, in order to implement proposition 3.3. Lines 7 to 10 are detecting specific cases regarding the cardinality of A and B.

Implementations of the propositions

Proposition 3.2 is implemented by lines 14-18, ensuring that no outsider (lines 14-15) nor a node in A or B (lines 17-18) is linked to all elements of $A \cup C$ (line 14) or $B \cup C$ (line 15). Proposition 3.3 is implemented by lines 22-25. The first constraints reads "*discard model if a or b is an empty set, and if X is an outsider linked to all elements of A, of B, and to all elements of C except one*". The second constraint acts likewise when A or B is a singleton. Proposition 3.4 is implemented by lines 29-31, grouping with *same_cc/2* the minimal element of A (line 30) and all that are not linked to an element of the group. The group therefore corresponds to a connected component in the complementary graph of A. If any node of A does not belong to the connected component, then A is covered by a biclique. Therefore, line 31 discards any model with an empty B and an element of A that does not belong to the connected component. Note that the same constraint for B is not implemented ; because the normal form precludes the symmetry between A and B (see second condition of definition 3.3). Therefore, it is enforced in later encodings that if A is empty, then B is empty.

Later ASP encodings will propose methods to yield the triplets. Solved together with the admissibility conditions constraints, it will reduce the output models to admissible triplets, thus diminishing greatly the amount of triplets to filter to obtain maximal triplets.

3.9.2 Triplets with a brute-force enumeration

It is possible to encode the definition of triplets in ASP quite directly (See encoding 3.2), using only the link information (`rel/2` atoms).

Input data and triplet composition Line 2 ensures the symmetry of the input relation, needed to enumerate all possible triplets. Lines 5, 6 and 7 are build-

```
% Ensure symmetry of the input relation.
rel(X,Y) :- rel(Y,X).

% Choose the composition of triplet concepts.
{ a(O): rel(O,_) }.
{ b(A): rel(A,_) }.
{ c(C): rel(C,_) }.

% Ensure that sets are disjoint.
:- a(O) ; b(O).
:- c(O) ; b(O).
:- a(O) ; c(O).

% Ensure existence of required rels between sets.
:- not rel(O,A) ; a(O) ; b(A).
:- not rel(O,C) ; a(O) ; c(C).
:- not rel(A,C) ; b(A) ; c(C).
:- not rel(D,C) ; c(D) ; c(C) ; C!=D.

% break symmetry
:- b(A) ; A<O: a(O).

% Discard supremum when empty.
:- not b(_) ; not c(_).
```

Encoding 3.2 – Generation of triplets using a full enumeration of possible sets.

ing the sets A, B and C of the triplet concept with a choice rules. The absence of cardinality constraint enables each set to contain any number of elements, including zero. Each set is built upon elements found in `rel/2`, with external constraints in lines 10-12 (e.g. `:- a(0) ; b(0)`) to preclude the three sets to overlap. Constraints in the choice rules (e.g. `{a(0): rel(0,_), not b(0)}` for A) could also have been used for that purpose. Note that the constraint on the B set ensures that the chosen elements are in relation with at least one element of A.

Triplet consistency Lines 15 to 18 ensure that relations are holding between members of different sets, and within C. Line 21 breaks the $A - B$ symmetry, discarding (B, A, C), doublon of (A, B, C), by cancelling models where an element of B is smaller than one in A. Line 24 discards the supremum $(A, \emptyset, \emptyset)$ obtained from the supremum formal concept, if it covers no edges. See section 3.7 for more about the supremum in the triplet concept ordering.

This encoding is very much like the Two-Liner implementation of the formal concept search in section 1.5.3: it encodes the full search space, with no considerations for efficiency. Together with encoding 3.1, it restricts the models to admissible triplets.

3.9.3 Triplets from the formal concepts

This approach exploits the reflexive version of the graph context, by deriving a triplet from each formal concept found in the graph context with all reflexive edges. The implementation is detailed in encoding 3.3.

Input data and triplet composition This encoding first ensures that the `rel/2` atoms are symmetric (line 2) and reflexive (line 3). Then, it describes the enumeration of formal concepts using the Two-Liner idiom (lines 6 and 7). It then computes the associated triplets using definition 3.1, obtaining the sets A , B and C encoded respectively by atoms `a/1`, `b/1` and `c/1`).

The rule in line 15 discards (B, A, C), doublon of (A, B, C). Line 18 discards the supremum triplet, exactly as in encoding 3.2.

```
1 % Ensure symmetry and reflexivity of the input relation.
2 rel(X,Y) :- rel(Y,X).
3 rel(X,X) :- rel(X,_).
4
5 % Mine formal concepts.
6 obj(O) :- rel(O,_) ; rel(O,A): att(A).
7 att(A) :- rel(A,_) ; rel(O,A): obj(O).
8
9 % Convert formal concept as triplet.
10 a(O) :- obj(O) ; not att(O).
11 b(A) :- att(A) ; not obj(A).
12 c(C) :- obj(C) ; att(C).
13
14 % Break symmetry on extent and intent.
15 :- b(A) ; A<O: a(O).
16
17 % Discard supremum when empty.
18 :- not b(_) ; not c(_).
```

Encoding 3.3 – Generation of triplets as concepts in the reflexive graph context.

Note that, unlike the two other methods, this generation does not need the constraints implementing proposition 3.2. This is because it computes triplets directly from individual concepts, that are already maximal objects, and therefore cannot have an outsider node linked to all elements.

3.9.4 Concept combination approach

As detailed in section 3.8, triplet concepts may be derived from combinations of formal concepts. Encoding 3.4 ought to be the direct translation of proposition 3.6, implemented with Algorithm 6 that describes the search for combinations of concepts forming a triplet concept, using previously mined formal concepts. As a last step, formal concepts that are not covered by triplet concepts are added to the set of triplet concepts with C as an empty set. The program is taking as input not only the relations `rel/2`, but also the formal concept encoded as `ext(N,O)` and `int(N,A)`, standing for the extent and intent of the `N`-th formal concept.

Input data and triplet composition Lines 2 and 3 compute the input relations from the input formal concepts. Relations will be needed later for clique

```
% Get the symmetric context from the concepts.
rel(O,A) :- ext(C,O) ; int(C,A) ; O!=A.
rel(A,O) :- rel(O,A).

% Choose three concepts.
1 {one(C): concept(C) } 1.
1 {two(C): concept(C) } 1.
1 {tee(C): concept(C) } 1.

% Extents of the three concepts.
a(X) :- one(C) ; ext(C,X).
b(X) :- two(C) ; ext(C,X).
cbase(X) :- tee(C) ; ext(C,X).

% There is one triplet for each maximal clique in C.
1 { c(X): cbase(X) } :- cbase(_).
:- c(X) ; c(Y) ; X<Y ; not rel(X,Y).
:- cbase(X) ; not c(X) ; rel(X,Y): c(Y).

% A U B, A U C and B U C
aub(X) :- a(X).  aub(X) :- b(X).
auc(X) :- a(X).  auc(X) :- cbase(X).
buc(X) :- b(X).  buc(X) :- cbase(X).

% First concept must be (A, B U C U X) ; same for the two others.
:- one(C) ; not int(C,A) ; buc(A).
:- two(C) ; not int(C,A) ; auc(A).
:- tee(C) ; not int(C,A) ; aub(A).

% Break symmetry.
:- b(A) ; A<O: a(O).

% Discard supremum when empty.
:- not b(_) ; not c(_).
```

Encoding 3.4 – ASP encoding of triplet search using proposition 3.6.

ensuring. Three formal concepts are chosen in lines 6-8.

Triplet consistency Lines 11-13 extract the content of chosen formal concepts to populate sets A, B and C_{base}. In the proposition, C is only the extent of a formal concept, not necessarily a clique itself. Lines 16-18 enumerate all maximal cliques in C_{base}, giving one model for each valid clique C. Lines 21-23 define the union of the sets, used in lines 26-28 to ensure that conditions 3.2, 3.3 and 3.4 of proposition 3.6 hold. The remaining rules, lines 31 and 33, discards (B, A, C), doublon of (A, B, C), and the supremum triplet, exactly as in the two previous encodings 3.2 and 3.3.

Grounded with the admissibility constraints, this generation method achieves the same results as previous encodings.

3.9.5 Comparing the three triplet concept search implementations

The three previously presented implementations are compared using a small randomly generated dataset. Results are shown in Table 3.2. The two datasets differ in size, i.e. number of objects. Both have been randomly generated using a naive uniform generation: each relation has a chance to appear equal to the density (40% in our case).

The first implementation, described in section 3.9.2, relies on the full enumeration of all possible triplets, and their filtering with many constraints. The second implementation, described in section 3.9.3, computes the triplets as formal concepts on a reflexive context, with additional constraints. Both of these methods achieve a result in an acceptable amount of space and time. On the other hand, the third method, relying on the combinations of concepts, does not seem to be usable in practice. The grounding is both 1000 times slower and 1000 times heavier than the two previous encodings.

	Size 80 ($\|O\| = \|A\| = 80$)				Size 40 ($\|O\| = \|A\| = 80$)			
	grounding		solving	total	grounding		solving	total
	size	time (s)	time (s)	time (s)	size	time (s)	time (s)	time (s)
3.9.2	1.1 Mo	**0.1**	44.0	44.1	356 Ko	**0.04**	0.3	0.36
3.9.3	**825 Ko**	**0.1**	**28.9**	**29.0**	**312 Ko**	0.06	**0.25**	**0.3**
3.9.4	1.2 Go	98.7	>2h	>2h	15 Mo	2	36	38.0

Table 3.2 – Comparison of the three triplet generation methods (full enumeration, derivation from concepts, and concepts combination), on two randomly generated datasets of density 0.4, with 80 and 40 nodes. First dataset has a graph context size of 80, issuing 176 496 formal concepts and 112 051 triplet concepts (all admissible triplets where triplet concepts). The second has a size of 40, issuing 6026 formal concepts and 3603 triplet concepts (all admissible triplets where triplet concepts).

3.10 Maximal Clique Enumeration with triplet concepts

Because triplet concepts are basically enumerating cliques and their biclique neighborhood, it is possible to enumerate (maximal) cliques of a graph given the triplet concepts as described by proposition 3.7, and illustrated in examples.

Let us first introduce lemma 3.1 that proposes the derivation of (not necessarily maximal) cliques from triplet concepts, and the subsequent definition 3.7 providing the clique operator on triplet and on the set of triplet concepts.

Lemma 3.1. *Cliques of a triplet*
Let $T = (A, B, C)$ be a triplet concept. C is a clique by construction. For any element a in A, and b in B, $D = C \cup \{a, b\}$ is also a clique.

Proof. T is a triplet concept, therefore all elements of A and B are in relation with all elements of C. $C \cup \{a\}$ is therefore a clique. All elements of A are connected to all elements of B. Hence $D = C \cup \{a, b\}$ is a clique. □

Definition 3.7. *Clique operator*
Let $\mathcal{T}$ be the set of triplet concepts derived from a graph context, and $T = (A, B, C) \in \mathcal{T}$.

$Cl(T)$ is the set of all cliques $C \cup \{a, b\}$ such that a is an element of A and b

is an element of B. $Cl(\mathcal{T})$ describes the set of all cliques derived from $\mathcal{T}$:

$$Cl(T) = \{C \cup \{a, b\} \mid a \in A, b \in B\} \tag{3.9}$$

$$Cl(\mathcal{T}) = \bigcup_{T=(A,B,C)\in\mathcal{T}} Cl(T) \tag{3.10}$$

Note that the number of nodes in the cliques of $Cl(C)$ depends on the number of nodes in the triplet. If $A = B = \emptyset$, then $|C| \geq 2$ and C is a clique of at least 2 elements. If exactly one of the two sets is empty, then $|C| \geq 1$ and there are at least 2 nodes in yielded cliques. If $C = \emptyset$, one can consider $A \times B$ as a list of cliques with 2 nodes.

The cliques obtained from the set of triplet concepts are issued from maximal objects: proposition 3.7 shows that a consequence of this property is that all maximal cliques of the graph are covered by a triplet concept, making possible their enumeration directly from the triplet concepts. However, not all cliques generated this way are maximal. The amount of cliques produced by a triplet concept is also computable, as per proposition 3.8.

Proposition 3.7. *MCE from Triplet Concepts*
Let $\mathcal{T}$ be the set of triplet concepts derived from a graph context $\mathcal{G}$. $Cl(\mathcal{T})$ contains all maximal cliques of $\mathcal{G}$.

Proof. Let Q be the nodes of a maximal clique of $\mathcal{G}$ not covered by any triplet of $\mathcal{T}$. Triplet $R = (\emptyset, \emptyset, Q) \notin \mathcal{T}$ is an edge-maximal triplet, that therefore should belong to $\mathcal{T}$. □

From proposition 3.7, it follows that triplet concepts are a compressed representation of (maximal) cliques. This link between triplet concepts and maximal cliques suggests that the number of triplet concepts remains exponential, although it is smaller than the number of cliques it describes. If enumerating the triplets is an efficient way to enumerate all maximal cliques remains an open question. Next proposition 3.8 gives an upper bound to the number of maximal cliques represented by a triplet concept. One should note that enumerating the cliques from all the triplet concepts leads to an important amount of doublons, since an edge covered

by n triplet concepts will appear in at least n cliques. Proposition 3.9 proposes a way to reduce the amount of non-maximal cliques generated from triplet concepts.

Proposition 3.8. *Number of locally maximal cliques*
Let $T = (A, B, C)$ be a triplet concept issued from graph $\mathcal{G}$. The locally maximal cliques $Cl_{locmax}(T)$ of T are the maximal cliques in the subgraph of T in $\mathcal{G}$. The number of locally maximal cliques is given by:

$$|Cl_{locmax}(T)| \ = \ max(1, |A|) \times max(1, |B|)$$

Proof. Let $T = (A, B, C)$ be a triplet concept. If $|A| > 1$ and $|B| > 1$, then only cliques of the form $C \cup \{a, b\}$, $a \in A, b \in B$ are yield by $Cl_{locmax}(T)$, since C and $C \cup x$, $x \in A \cup B$ are covering strictly less edges. Therefore, $|Cl_{locmax}(T)| \ = \ |A| \times |B|$. If B is empty, $Cl_{locmax}(T)$ only yields cliques composed of nodes $C \cup x$, $x \in A$, thus $|Cl_{locmax}(T)| \ = \ |A|$. If both A and B are empty, $Cl_{locmax}(T)$ returns the single clique C, thus $|Cl_{locmax}(T)| \ = \ 1$. □

Proposition 3.9. *Non-maximality of cliques*
Let $T = (A, B, C)$ be a triplet concept issued from graph $\mathcal{G}$. Let C be a clique described by T. C is not maximal if, for any other triplet concept $U = (U_1, U_2, U_3)$ of $\mathcal{G}$, one of the following holds:

- $C \subseteq U_i \cup U_j$, $|U_k| > 0$ *for any permutation* (i, j, k) *of* $(1, 2, 3)$
- $C = U_3$ *and* $U_1 \cup U_2 \neq \emptyset$
- $C \subset U_3$

Proof. Let $T = (U_1, U_2, U_3)$ be a triplet concept, and C the nodes of a clique. If $Cl \subseteq U_1 \cup U_2$ and $|U_3| > 0$, then a node $c \in U_3$ can be added to C, constituting $C' = C \cup c$, a clique covering more edges than Cl. Same reasoning applies for any combination of two sets of T. If $C \subseteq U_3$, then only one node in $U_1 \cup U_2$ is enough to obtain a clique covering more edges than C. If $C \subset U_3$, then U_3 is a clique covering more edges than C. □

Example 3.19. Considering the graph described in Figure 3.1 and using for instance the Bron–Kerbosch algorithm, we can enumerate the maximal cliques of the graph: $\{a, d, e, f\}$, $\{b, d, e, f\}$, $\{b, c, e, f\}$, $\{a, c, e, f\}$.
First, it appears that these maximal cliques are found as triplets in Table 3.1, at lines 4, 10, 11 and 13.
Second, the only triplet concept of that example, $(\{a, b\}, \{c, d\}, \{e, f\})$, describes all these maximal cliques.
Because there is no set of elements linked to all elements of one of these cliques, the maximal cliques are also maximal motifs in this example (line 4, 10, 11 and 13). Following lemma 3.1, the maximal cliques may be reconstructed from some of the other triplets. For instance, clique $\{a, d, e, f\}$ is also described by triplets $(\{a, b\}, \emptyset, \{d, e, f\})$ (line 2), $(\emptyset, \emptyset, \{a, d, e, f\})$ (line 4) and $(\{c, d\}, \emptyset, \{a, e, f\})$ (line 6).
But some triplets do not generate any maximal clique: triplet $(\{a, b, c, d\}, \emptyset, \{e, f\})$ (line 1) describes the four cliques $\{a, e, f\}$, $\{b, e, f\}$, $\{c, e, f\}$ and $\{d, e, f\}$.

Example 3.20. In the graph context of Figure 3.8, non-maximal cliques $\{d, i, g\}$, $\{c, j, k\}$ and $\{d, h, i, j, k\}$ cannot be derived from the 8 triplet concepts listed in example 3.14. To obtain these non-maximal cliques, the powerset of other cliques must be explored. For instance, $\{c, j, k\}$ and $\{d, h, i, j, k\}$ are subsets of the clique $\{c, d, h, i, j, k\}$ derived from $(\{d, e, f\}, \emptyset, \{c, h, i, j, k\})$.

3.10.1 ASP implementation

The enumeration of (possibly non-maximal) cliques from triplet concepts can be encoded in ASP. Encoding 3.5 proposes an implementation. Input atoms are `a/2`, `b/2` `c/2` are encoding the content of sets A, B and C for each input triplet. The encoding yields one model for each clique, encoded with `clique/1` atoms. Optionally, it is possible to provides `edge/2` atoms, indicating links between nodes, so that only the maximal cliques are provided as a result of the program.

```
% INPUTS:
%   a(N,X): X belong to A of triplet N
%   b(N,X): X belong to B of triplet N
%   c(N,X): X belong to C of triplet N
%   edge(X,Y): X is linked to Y.
%     (optional: provides only maximal cliques)
% OUTPUT: one clique per model
%   clique(X): X belong to the clique

% Choose the triplet.
triplet(N) :- a(N,_).
triplet(N) :- b(N,_).
triplet(N) :- c(N,_).
1{current(N): triplet(N)}1.

% Choose a in A and b in B.
1{clique(X): a(N,X)}1 :- a(N,_) ; current(N).
1{clique(X): b(N,X)}1 :- b(N,_) ; current(N).
clique(X) :- c(N,X) ; current(N).

% Only maximal cliques.
edge(X,Y) :- edge(Y,X).  % symmetry
outsider(X) :- edge(X,_) ; not clique(X).
:- outsider(X) ; edge(X,C): clique(C).

#show clique/1.
```

Encoding 3.5 – Generation of (maximal) cliques from the triplet concepts.

Description of encoding 3.5

First lines describe the input and output atoms. Lines 11-14 are choosing one of the input triplet as *current*, i.e as the triplet under consideration. Each triplet will be chosen once. Lines 17-18 choose one element of A and one element of B if these sets exists: if A is empty set, no element has to be chosen from it. Note that without the condition part (after `:-`) of line 17, it would be impossible to choose exactly one item of A if $A = \emptyset$, hence the model would be invalidated and all triplets with $A = \emptyset$ would be ignored. Same goes for B and line 18. Line 19 indicates that all nodes of C belong to the outputed clique. Lines 22-24 are (1) ensuring symmetry of edges, (2) defining the outsider nodes, i.e. nodes that do not belong to the outputed clique, and (3) discarding any model consisting in a non-maximal clique. These rules assume `edge/2` atoms are provided in addition. In that regard, `edge/2` atoms are optional: if not provided, the full $Cl(\mathcal{T})$ set will be computed, whereas their presence restricts to maximal cliques only. Finally the `#show` line restricts the output to the clique elements. As shown in example 3.19, the same clique can be found using different triplets. As a consequence, to ensure the unique listing of cliques, one has to project results on the `clique/1` atoms, exactly as in chapter 1, section 1.5.5, where the models of encoding 1.2 yields doublons.

Efficiency of MCE from triplet concepts

Enumeration of maximal cliques from triplet concepts, using solely ASP encoding, is not efficient compared to dedicated algorithms. On the context of size 40 used to compare triplet concept search implementation in Table 3.2, the reference implementation of Bron & Kerbosch algorithm as adapted by [105] took 2.10^{-3} seconds to find the 272 maximal cliques, where encoding 3.5 needed 20 seconds (search for triplet concepts excluded) for the exact same result. Note that, without projection to ensure the unicity of yielded cliques, encoding 3.5 would yield 1265 solutions (each clique yielded 4.6 times on average), and that without restricting the output to maximal cliques (i.e. without providing the `rel/2` atoms), 967 cliques are yielded in 8 seconds (32350 cliques without the projection).

This section ends this chapter by demonstrating the link between formal concepts and cliques, proving that the triplet concepts, built from a graph motif involving a clique component, are a descriptions of maximal cliques in the graph. More questions will be addressed in perspectives, chapter 6.

3.11 Conclusion on triplet concepts

Summary

We introduced the term *Triplet* to reference a kind of formal concepts derived from a reflexive graph context. From a graph perspective, triplets are equivalent to *overlapping bicliques*, a biclique where the two sets of nodes may overlap. Triplet concepts are represented using their triplet representation (A, B, C).

The edge-maximal triplets are a subset of triplets describing a maximal overlapping biclique in the underlying graph. Triplets can also be written in a canonical way, ensuring unicity of representation of a given overlapping biclique motif. The canonical writing of a triplet relies on two simple conditions (no ubiquitous node in A or B, and $min(A) = min(A \cup B)$).

The *triplet concepts* are simply the triplets that are both edge-maximal and written in the canonical way. The triplet concepts may be used to describe a variation of Power Graph Analysis, where instead of considering cliques and bicliques, the compression process looks for and compress overlapping bicliques. We also shown that the triplets concepts are partially ordered, but, with the current definitions of the meet/join operators and of the canonical form, they do not form a lattice. We detailed three methods to generate the triplet concepts along their ASP encoding, and finally explored the link between maximal clique enumeration and triplet concept enumeration.

Compression search space with triplet concept

In section 2.4 was proposed a definition of the compression search space, where two sets of formal concepts, issued from the non-reflexive and reflexive graph contexts, were used simultaneously to describe the compression search space. With triplet concept however, both motifs are unified and the triplet concept semi-lattice can now encode the whole compression search space alone. The motifs used to ex-

plore the search space are a subset of the triplet concepts, hence they can be used alone to constitute the search space of graph compression.

See chapter 4 for an application of triplet concepts on real data, and chapter 6 for a more in-depth perspectives of this chapter.

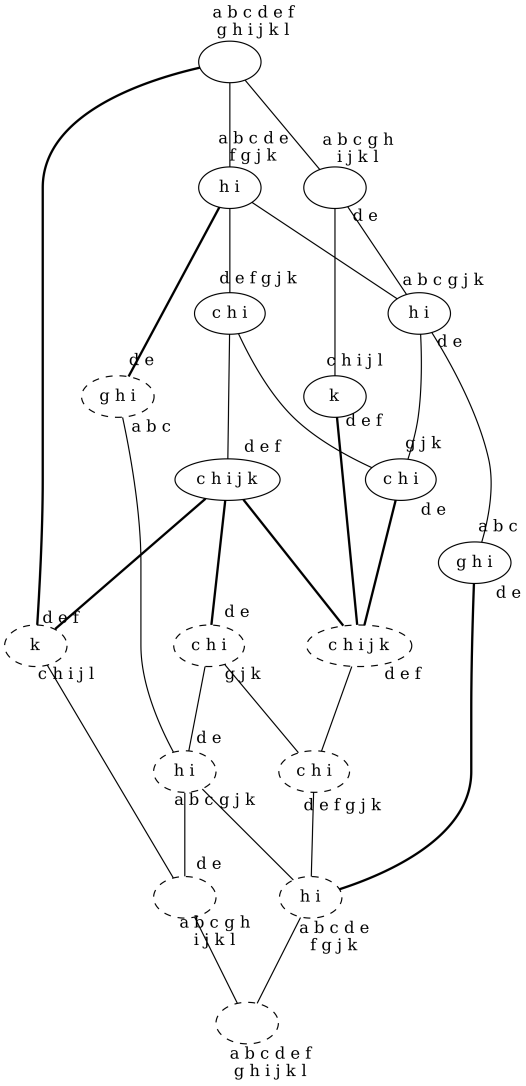


Figure 3.10 – Graph of inclusion of triplets of the graph context in Figure 3.8, including the symmetric triplets (B, A, C) as dashed node. A is shown as the upper label of each node, B as the lower label, and C as the center label. The inclusion edges between symmetrics and triplet concepts are shown in bold, to mark the reversion of inclusion order for the C set.

Chapter 4

Applications of Power Graph Analysis

His Holiness the Flying Spaghetti Monster is Eternal,
without beginning and without end,
and with a whole tangled mess in the middle

The Loose Canon,
Second Announcement Regarding Canonical Belief, 1
Evangelical Pastafarian church, 2010

Chapter 2 detailed a rational reconstruction of Power Graph Analysis relying on logic programming and formal concept analysis. This chapter presents applications of this approach.

It starts with two biological networks, presented in section 4.1, and compressed in sections 4.2 and 4.3. Those first sections are reproduced from [20].

Then, using the same datasets, triplet concepts are searched, mimicking the preliminary results of [109] with Power Graph Analysis. Finally, section 4.4 presents a benchmark of optimizations presented in section 2.7 and implemented in *PowerGrASP*.

4.1 Introduction to the data

The network named *aphid RNA* comes from a transcriptome study, in collaboration with Denis Tagu (INRA Le Rheu), on the pea aphid (*A. pisum*) [109]. It is a bipartite interaction graph linking two disjoint populations of RNA molecules (15×1810). The second study is a collaboration with N. Théret (Inserm). We

	rna		mdb
	reduced	complete	core
time (s) *PowerGrASP*	10.2	55.9	14.6
time (s) *Oog*	>1	5	2
edge reduction	93.05%	95.64%	64.08%
conversion rate	25.26	43.04	48.86
#powernode	35	50	84
#poweredge	35	50	89
#remaining	31	133	147

Table 4.1 – Network compression results with *PowerGrASP*. Implementation of Royer *et al.*, *Oog*, achieve the same results except for compression time. *Compression time*: time needed to fully compress the graph (iterative search of all concepts); *edge reduction*: given by $\frac{\#initial\ edges - \#poweredges}{\#initial\ edges}$; *#powernode*: number of non-singleton powernodes in the compressed graph; *#poweredge*: number of non-singleton poweredges; *#remaining*: number of uncompressed edges (or singleton poweredges) remaining.

	Time (s)		Edge reduction	
	Royer *et al.*	*PowerGrASP*	Royer *et al.*	*PowerGrASP*
aphid RNA	5	43.04	95.64%	95.64%
reduced RNA	>1	10	93.05%	93.05%
core MatrixDB	2	80	64.08%	64.08%

Table 4.2 – Comparison of *PowerGrASP* and Royer *et al.* implementation. They are equivalent in results, *PowerGrASP* is slower.

worked on a network called *extended MatrixDB*, extracted from the database MatrixDB [69], describing interactions between extracellular proteins. It is described in section 4.3. These two graphs are described in sections 4.3 and 4.2, and the compression results in Table 4.1. A graphical representation of the *aphid RNA* and *core MatrixDB* networks are shown in figures 4.1 and 4.3. All compressions have been run on one core of an *Intel i7-6600U* (2.60GHz).

4.1.1 What do we seek in the data ?

Power Graph Analysis in ASP

The powergraph compression code is benchmarked here using three networks coming from biological data. Although it is just a useful framework to quickly design search spaces, we show that our ASP code, gathered in the package *PowerGrASP*, is already useful in managing real graphs. As shown in Table 4.2, *PowerGrASP* reaches a score equivalent to *Oog*, the Java implementation of Royer *et al.*. In fact, since both implement the same strategy, edge reduction is very similar up to slight variations due to non-determinism of motif choices with equal scores. *Oog* uses a dedicated algorithm, making it much more scalable. On the other hand, *PowerGrASP* presents flexibility in the definition of motifs and the overall search space exploration. As a demonstration of that flexibility, stars and triplet concepts were added to the set of compressible motifs.

Bipartite stable search

As shown in section 2.7.3, the stable is the object needed by proposition 2.2 in order to restrict the largest concept search space. It is computed in presented data with a time limit of 5 seconds ; its optimality is thus not guaranteed, but not required either for the application of an optimization. Results show that stables of consequent size are found (see tables 4.3 and 4.4). In case of the *aphid RNA* network, 9 out of 15 nodes of the first set and one third of the second are involved in the stable. This result suggests that an optimization based on proposition 2.2 could be beneficial for large graphs.

Triplet concept search

As described in chapter 3, a triplet concept describes an overlapping biclique in the graph. In this chapter, we present a preliminary search in order to identify the main triplet concepts in the datasets. However, the whole networks (*aphid RNA* and *extended MatrixDB*) are too large to be treated by current implementations of triplet search. We have thus designed experiments on reduced versions of the networks (*reduced RNA* and *core MatrixDB*).

A Power Graph style compression of the largest triplets is shown in figures 4.10 and 4.7.

The search for triplet concept motifs has been implemented on *PowerGrASP*. It can therefore be searched as any other motif (biclique, star, clique, quasi-biclique). The obtained powergraphs are mostly identical to the powergraph obtained with a biclique and clique search, validating triplets as a way to encode the compression search space.

4.2 Application to a transcriptomic study of the pea aphid, *Acyrthosiphon pisum*

Aphids are crop pests using two different reproduction modes to ensure their quick proliferation during spring and summer, and the production of cold-resistant eggs before winter. Aphid females produce either sexual or asexual embryos, depending of the external conditions, especially night length. With long nights the embryos produced by the viviparous females are sexual, so next generation will use oviparous sexual reproduction to produce cold-resistant eggs needed to survive winter. With short nights however comes the parthenogenesis reproduction, where viviparous females reproduce efficiently, until autumn where longer nights trigger the reproductive mode shift.

Determination of the biochemical pathways involved in the detection of seasonal state, and the transmission to the embryos to define their (a)sexual state, is an important step toward the management of those crop pests.

The pea aphid is a model in aphid genomics, and therefore the subject of a wide range of genomic analysis, including genome sequencing and gene annotations [31, 75]. Another field of research is the post-transcriptional regulation of messenger RNAs (mRNA) by small non coding RNAs called micro RNA (miRNA). They were sequenced and shown to play an important role in reproductive polyphenism [71], thus the embryos determination of the pea aphid. In addition to sequencing data, the transcriptome can be strengthened by *in silico* predictions of interactions between mRNAs and miRNAs. By comparing the differences between transcriptomes of non-differenciated embryos and sexual and asexual embryos, it is possible to isolate the RNAs with differential expression levels in the two alternative reproduction modes [70].

Dataset

The dataset considered in that chapter, and referred as *aphid RNA*, is constituted of micro RNA and messenger RNA (mRNA) showing differential expressions between the asexual and sexual embryos development paths. Interaction predictions between those two sets produces a bipartite network. A first study with a partial Power Graph Analysis was available at the start of this thesis [109].

Reduced version

A preliminary look at the graph reveals that one miRNA, *3019-5p*, is almost ubiquitous in the network. It is linked to many nodes, most of them specific, totalizing alone one third of the network. In order to allow the study of the graph without focusing on this specific feature, which can be assimilated to noise, we have produced a reduced version of the network, *reduced rna* by removing *3019-5p* along with its specific interactants.

Both *aphid RNA* and *reduced RNA* are presented in Table 4.3.

	#node	#edge	density	#cc	bipartite	stable
reduced RNA	848	950	2.6×10^{-3}	1	14×834	834
aphid RNA	1825	2250	1.3×10^{-3}	1	15×1810	765

Table 4.3 – Statistics on *aphid RNA* and *reduced RNA*. *#node*: number of nodes in the graph; *#edge*: number of edges; *density*: global density of the graph; *#cc*: number of connected components in the graph; *stable*: size of the maximal stable found in the largest connected component in less than 5 minutes.

4.2.1 Triplet concepts in *reduced RNA* network

The *reduced RNA* network presents only 67 triplet concepts for 1246 formal concepts, none of them with 3 or more elements in A and B. As already observed, this network is composed only of stars and small bicliques. As a consequence, most triplets are constituted of a large A and a single element in C. Note that the powergraph obtained with *PowerGrASP* by compressing only triplet concepts leads to a similar powernode hierarchy and edge reduction (as shown in Figure 4.2): using the triplet motif does not change the final compression.

Figures 4.8 and 4.9 show the distribution of triplet concepts as a function of their cover in the overall network.

The largest triplet concepts can be represented in the network using the Power Graph representation, as shown in Figure 3.5 (section 3.2.1). The result is shown in figure 4.10.

4.3 Application to human extra-cellular matrix

MatrixDB is an interaction database of extracellular matrix components. This network, reduced to human-only interactions, constitutes the *core MatrixDB* dataset. In addition to this network, we added other human interactions from other databases. The obtained network is labelled *extended MatrixDB*. For a more in-depth presentation of these dataset, see Table 4.4. We also worked on subsets of this network [80], in collaboration with Nathalie Theret (Univ Rennes, Inserm, EHESP, Irset).

4.3.1 Data origin and construction

The following presentation of the data is reproduced from [80]. A Power Graph Analysis of *core MatrixDB* network is shown in Figure 4.3.

The data originate from MatrixDB [96, 26], which is a freely available interaction database on components of the extracellular matrix. It contains molecular interactions between proteins, proteoglycans and polysaccharides. In addition to MatrixDB data, we also used data from the IntAct database [83]. IntAct provides an open source database for molecular interaction data. All interactions are derived from literature curation or direct user submissions.

We used two data sets provided by S. Ricard-Blum (University of Lyon). The first one lists the interactions found in MatrixDB, we called it *core MatrixDB*. The second one is an extension of *core MatrixDB* as it also contains interaction data from IntAct. We decided to work only on human molecular interaction data and called this data set *extended MatrixDB*.

Table 4.4 gives some elementary statistics on the two data sets. Note that the extended graph is much more complex than the core one and very challenging with respect to the compression task.

	#node	#edge	density	#cc	bipartite	stable
core MatrixDB	286	657	1.6×10^{-2}	5	no	213
extended MatrixDB	8940	30427	7.6×10^{-4}	43	no	no result

Table 4.4 – Statistics on *extended MatrixDB* and *core MatrixDB*. *#node*: number of nodes in the graph; *#edge*: number of edges; *density*: global density of the graph; *#cc*: number of connected components in the graph; *stable*: size of the maximal stable found in the largest connected component in less than 5 minutes.

4.3.2 Triplet concepts in *core MatrixDB* network

The *core MatrixDB* network issues 1792 formal concepts and 435 triplet concepts, found in 12 seconds using the enumeration from reflexive formal concepts (see section 3.9.3). Among triplet concepts, 23 of them have at least 4 elements in A and B sets. There is 33 triplet concepts with at least 3 elements in the C set, 124 with at least 2 elements in the C set, and 126 with at least one element in each set (hence, at least 2 in A and B sets). Figures 4.5 and 4.6 show the distribution of triplet concepts as a function of their cover in the overall network. As expected, the triplet concepts allow an improvement of compression by both decreasing significantly the number of poweredges, and making them larger.

The largest triplet concept can be represented in the network using the Power Graph representation, as shown in Figure 3.5 (section 3.2.1). The result is shown in figure 4.7.

Interpretation of the largest triplet concept

The largest triplet found in the *core MatrixDB* network is shown in Figure 4.4. It covers 48 edges, thus is less important in size than the main stars, but it represents a more complex structure. Thanks to N. Theret, we were able to interpret this structure. It is worth to note that in the two non-clique clusters, there are very few internal edges.

P_1: clique made of Endostatin, PCOLCE and Thbs1

Thbs1 and Endostatin are known antiangiogenic factors [56], and their link with PCOLCE is not established. However, Salza et al. shown that PCOLCE is involved in new functions [96], that could therefore be shared to Thbs1 and Endostatin.

P_2: powernode made of Collagen-VI, Beta-amyloid peptide 1-42 and Collagen-I

The link between P_2 and the others may be explained by the role of angiogenesis in the construction and deconstruction of the central nervous system [107].

P_3: powernode made of Heparan Sulfate, TGM 2, BGN, Heparin, Dermatan Sulfate and Chondroitin Sulfate A

This group is mainly constituted of glycosaminoglycans, or GAGs, which are major actors in the regulation of angiogenic processes [28], and as such are expected to be in relation with antiangiogenic factors, found in P_1. This constitutes a strong angiogenesis-related clustering. The exceptions are (1) the proteoglycan BGN, which is a formation of peptides and Dermatan Sulfate, hence the internal link with it, and (2) the transglutaminase 2, which interaction with heparin is well documented [41] and found inside the group between TGM 2 and Heparin nodes.

This triplet groups together important elements in the regulation of ambiogenesis. In the regular powergraph, shown in Figure 4.3, the elements are scattered in different powernodes hierarchies. This clustering is therefore not similar to existing ones.

4.4 Benchmarks of the *PowerGrASP* optimizations

This section aims to show the efficiency gain obtained by *PowerGrASP* when using the three implemented optimizations described in section 2.7. The benchmarks are realized on *core MatrixDB*, and the results are grouped in Table 4.5, showing that optimizations allowed a better performance. The edge reduction and conversion rate are otherwise identical, whatever the optimization used. *Vanilla*

refers to *PowerGrASP* without any of the three compared optimizations activated. To obtain the results of Table 4.4, we used also other implementation-specific optimizations not presented in this thesis.

Except for the star search, all the optimizations where useful, leading to a faster compression. A more curious behavior (not presented in the table) is that both *graph filtering* and *parallel motifs* become slower when used together with *star motif*. However, the *all* case, where all three optimizations are activated, is the most efficient implementation.

One may wonder why the star motif slows the search except when coupled with the two others optimizations. We hypothesis that introducing the star motif is an important slowdown. On the other hand, the star motif bounds are extremely efficient (max node degree), so finding and enumerating the optimums, as proposed by the parallel motif optimization, may be quite efficient. Coupled with graph filtering that removes a lot of suboptimal stars, we obtain enough room for the star motif to counterbalance its cost. The same behavior is found in other networks, such as *aphid RNA* despite the fact that stars are preponderant in that dataset.

Implementation limits The implementation of graph filtering in *PowerGrASP* could be improved: removing edges in the graph may divide it into smaller connected components, and as such would allow to handle each new component as an independent graph, thus reducing the total search space. This is not handled by *PowerGrASP*, and could be especially interesting on sparse graphs.

4.5 Conclusion on the applications

Triplet concepts

When triplet concepts are used in place of the standard bicliques and cliques on a graph where no triplet structure exists, we obtain the same results on the tested graph. There is no degradation compared to Power Graph Analysis, as implemented by *Oog*, or by the formal concept approach we implemented with *PowerGrASP*. On the other hand, non-degenerated triplet concepts are present in

	core MatrixDB compression time (s)	Runtime compared to *vanilla* in percent
vanilla	51.62	100%
graph filtering (section 2.7.2)	35.38	68.54%
star search (section 2.7.1)	65.67	127.21%
parallel (section 2.7.4)	19.81	38.37%
all but star search	16.65	32.25%
all	14.56	28.2%

Table 4.5 – Runtime for *PowerGrASP* when using some optimizations, compared to the vanilla implementation, on the *core MatrixDB* network.

core MatrixDB, and associated to a particular structure of chemicals that is not present in the standard powergraph.

Overlaps with triplet concept

Triplet concepts are by construction introducing the notion of overlap in the motifs. Overlaps in the output representation were avoided in Power Graph Analysis, but in some subsequent works such as [3] this constraint was relaxed for a more generalized approach. The single-edge representation of triplets (as shown in Figure 3.4, chapter 3) introduces an overlap, but less edges than the standard Power Graph representation (as shown in Figure 3.5, chapter 3). The introduction of overlaps does not arm the visualization as long as they are not entangled in other relations with other motifs of the graph. In other words, single-edge representation of triplets may be acceptable as long as (1) the overlap part does not interact with any other part of the graph, or (2) if the overlap part interacts with other parts of the graph, it is as a whole (i.e. all interactions concern all nodes in the C set of the triplet). With such a restriction, the overlap part would never be split nor involved in other overlaps, hence limiting its visual complexity.

The preliminary results of this chapter seems to indicate that an implementation of Power Graph Analysis using triplet concepts instead of cliques and bicliques achieves the same compression result, unless triplets with nodes in the three set are searched. In such a case, one would obtain different compression results, unravelling new structures in the data.

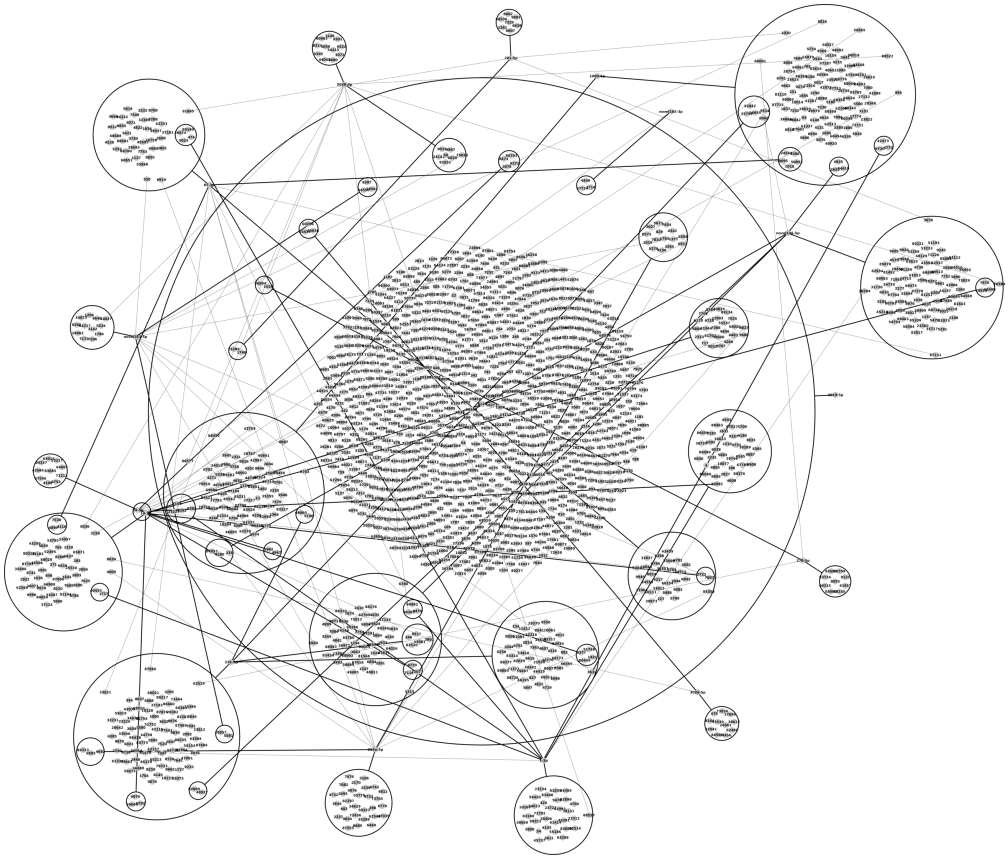


Figure 4.1 – Cytoscape view of *aphid RNA*, compressed with *PowerGrASP* where two types of molecules interact. The 15 nodes organized around the central powernode belong to the so-called *micro RNA* type and appear around the central power node that contains most of the molecules of the *mRNA* type. The messenger RNA specific to a micro RNA are in the external powernodes. The big central powernode is composed of all interactants of *3019-5p*, occupying a large portion of the network.

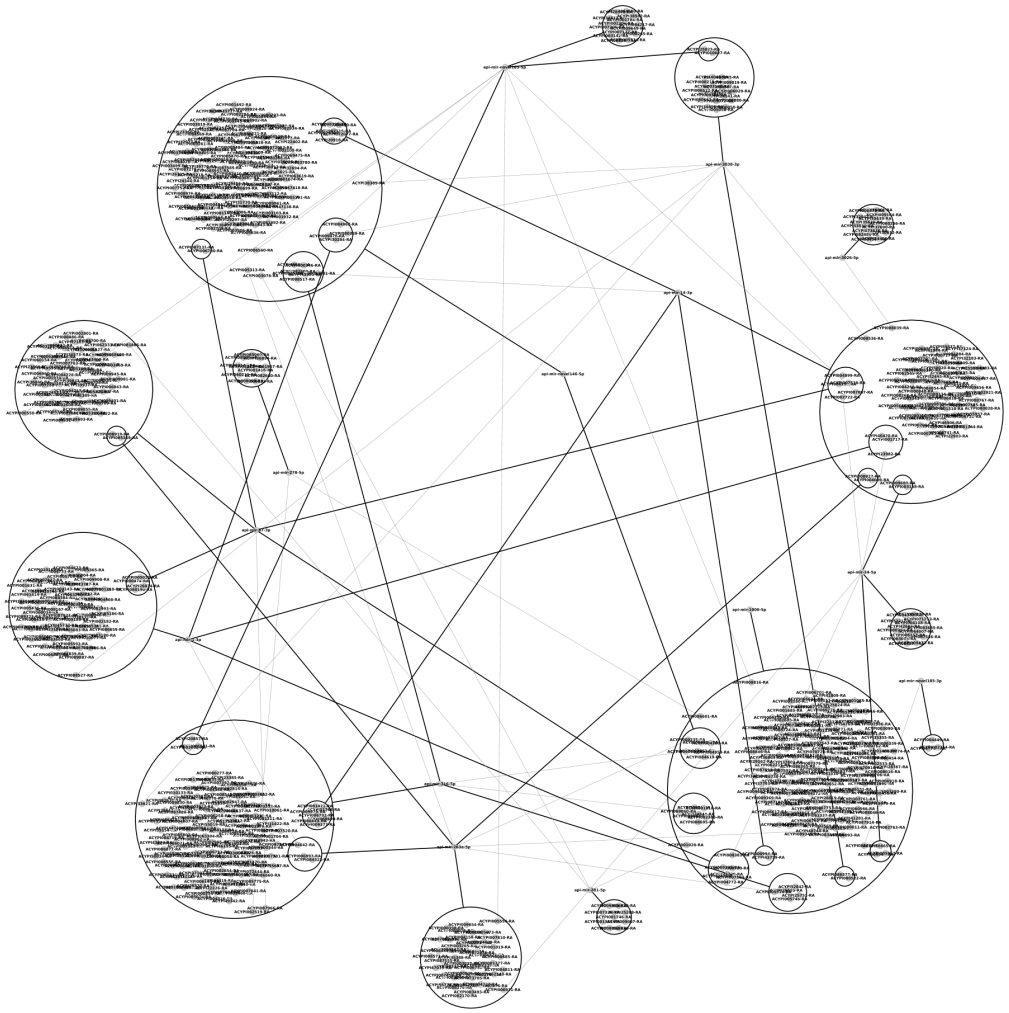


Figure 4.2 – Cytoscape view of *reduced RNA*, compressed with *PowerGrASP* where two types of molecules interact. The 14 micro RNA are stars in the graph.

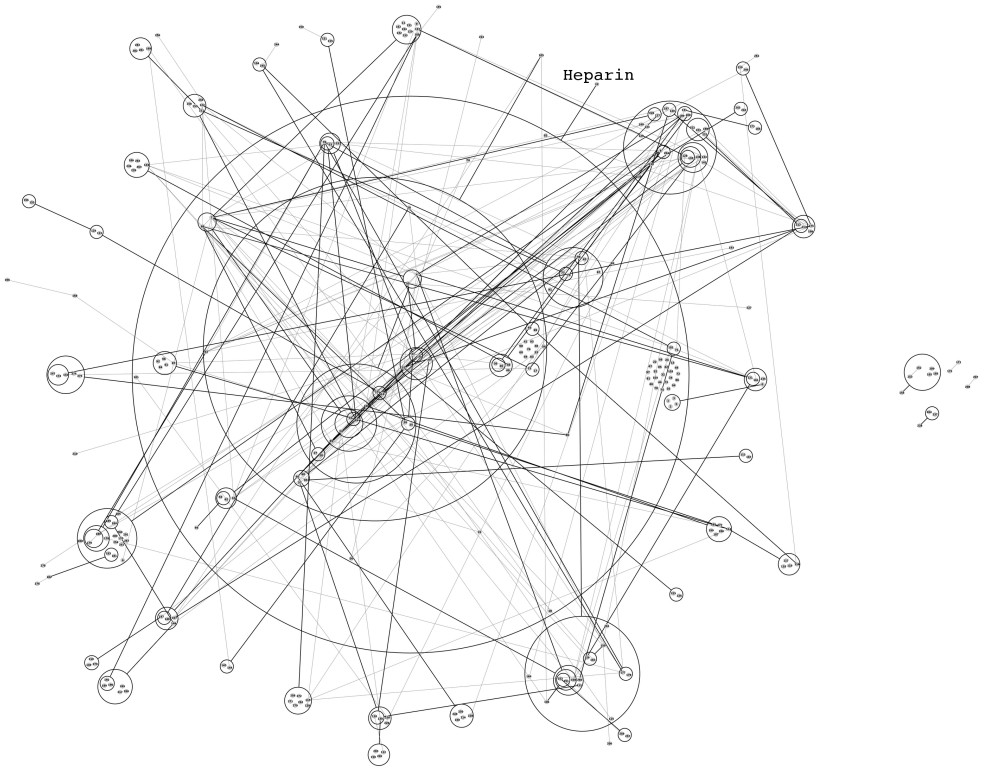


Figure 4.3 – Cytoscape view of *core MatrixDB* network. The Heparin node is linked to the big central powernode.

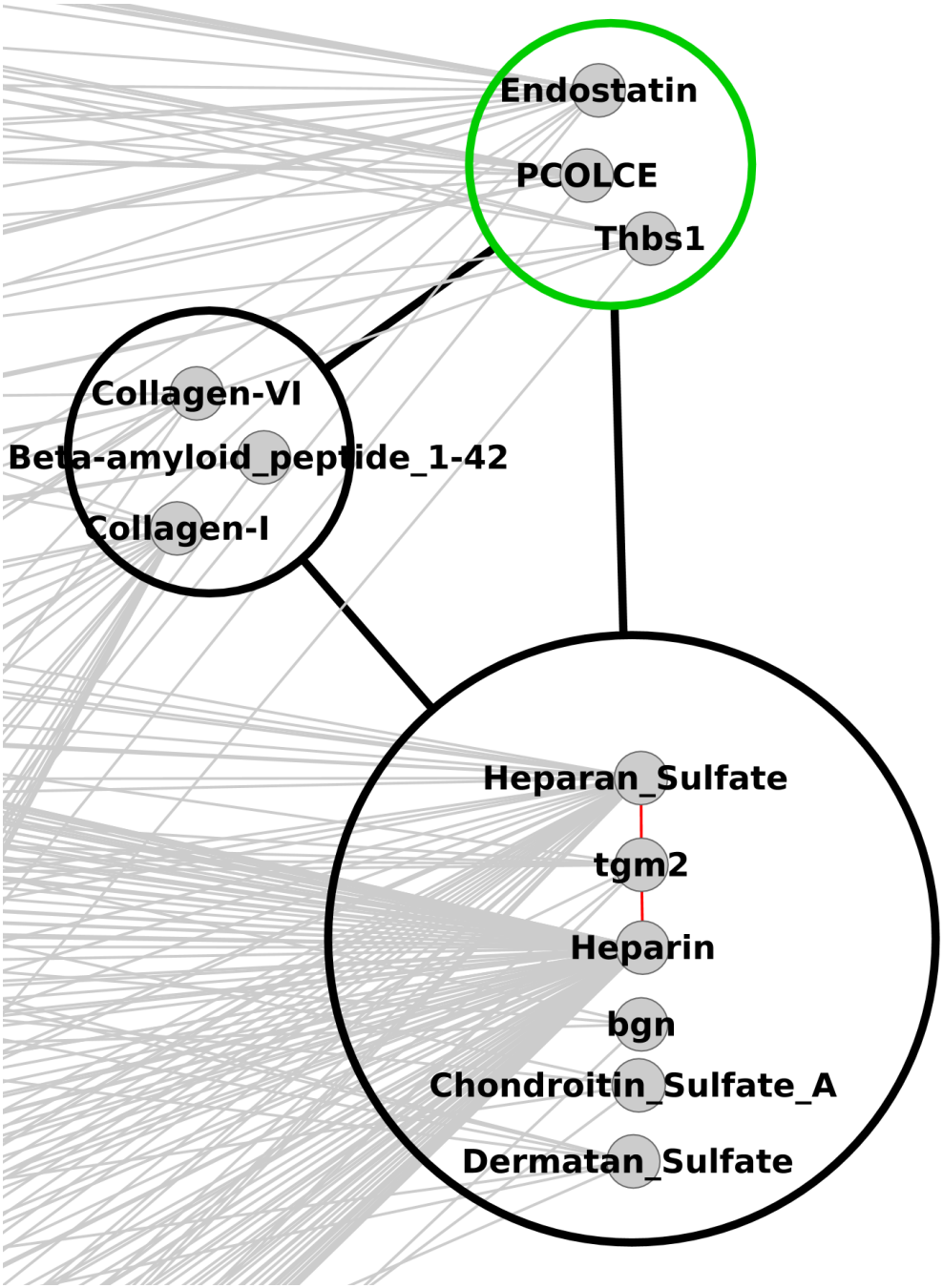


Figure 4.4 – The largest triplet in *core MatrixDB* with 3 elements in the C set (in green), discussed in section 4.3.2. There are two internal edges in the Heparin cluster involved in this motif, highlighted in red. The edges to nodes outside the triplet are shown in light grey, clearly showing the difficulty of perceiving the triplet motif in the dense entanglement of the edges.

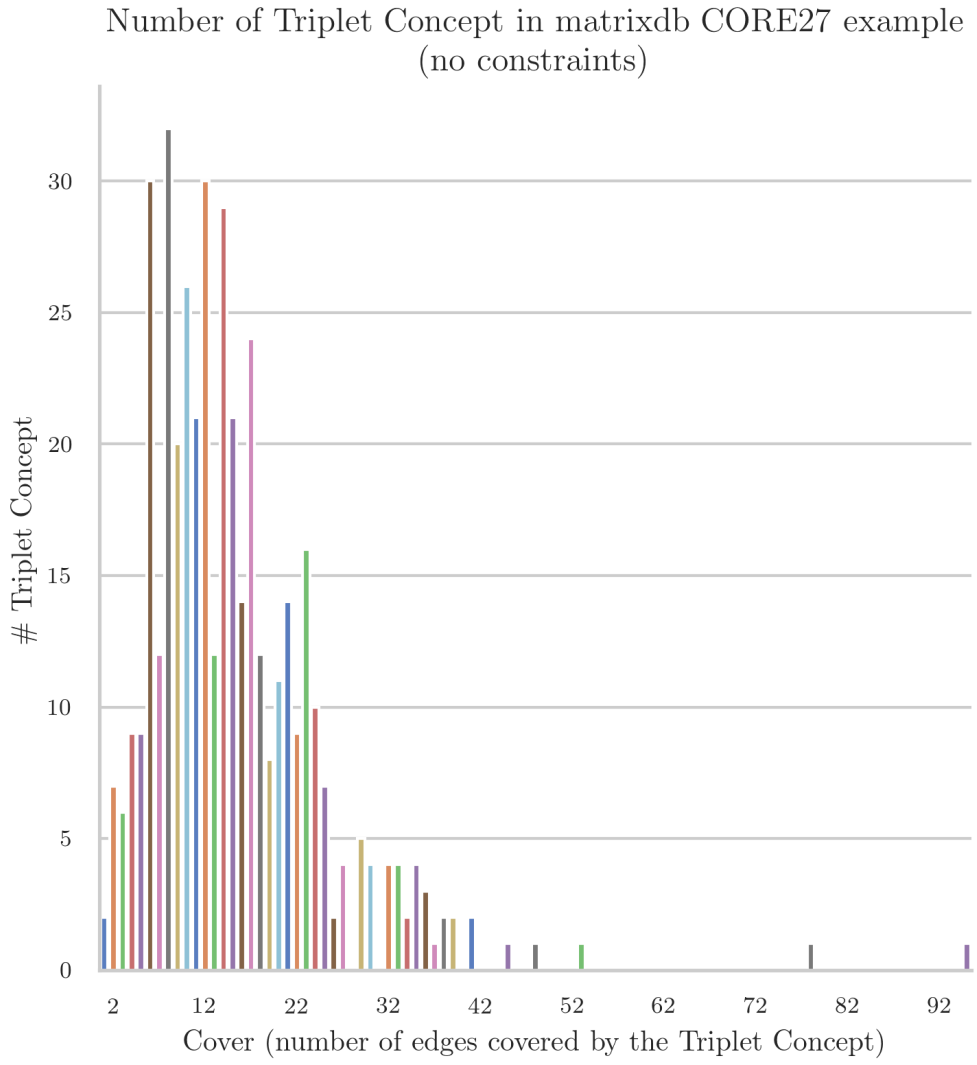


Figure 4.5 – Distribution of triplet concepts in the *core MatrixDB* network according to their cover in edges.

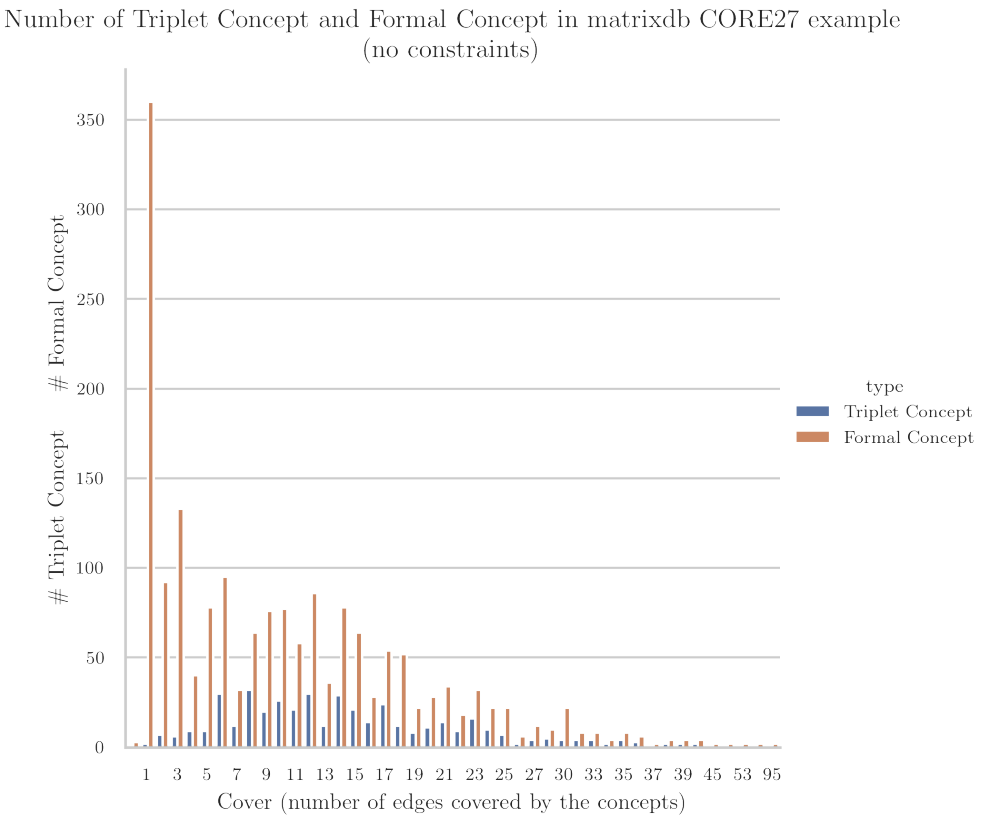


Figure 4.6 – Distribution of triplet concepts and formal concepts in the *core MatrixDB* network according to their cover in edges.

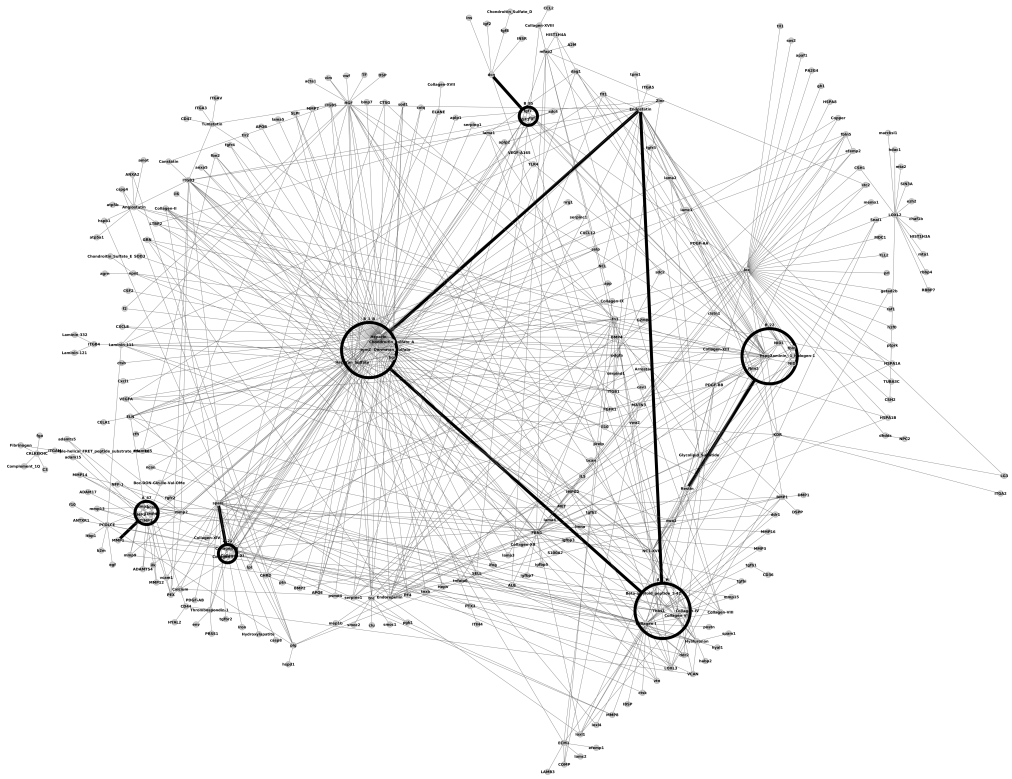


Figure 4.7 – Top-level compression of the largest triplet concepts found in *core MatrixDB* network.

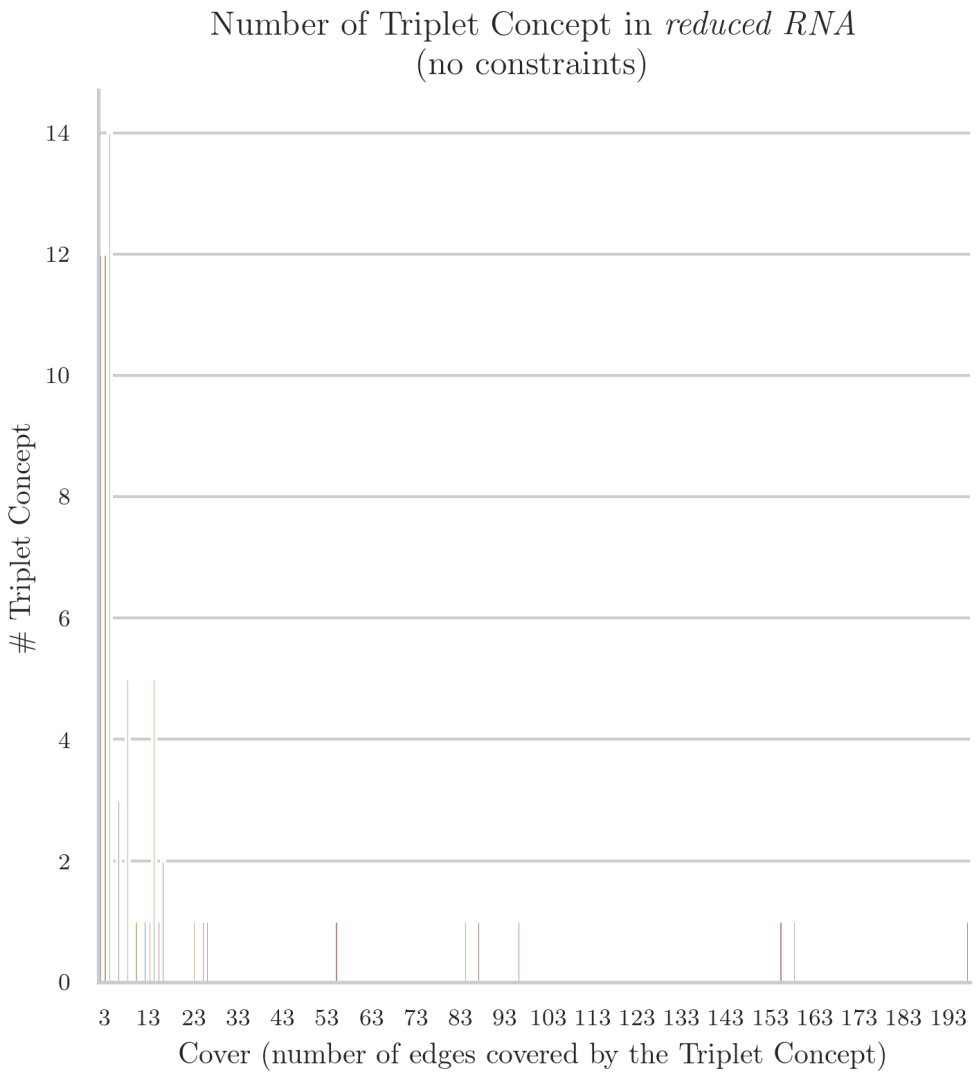


Figure 4.8 – Distribution of triplet concepts in the *reduced RNA* network according to their cover in edges.

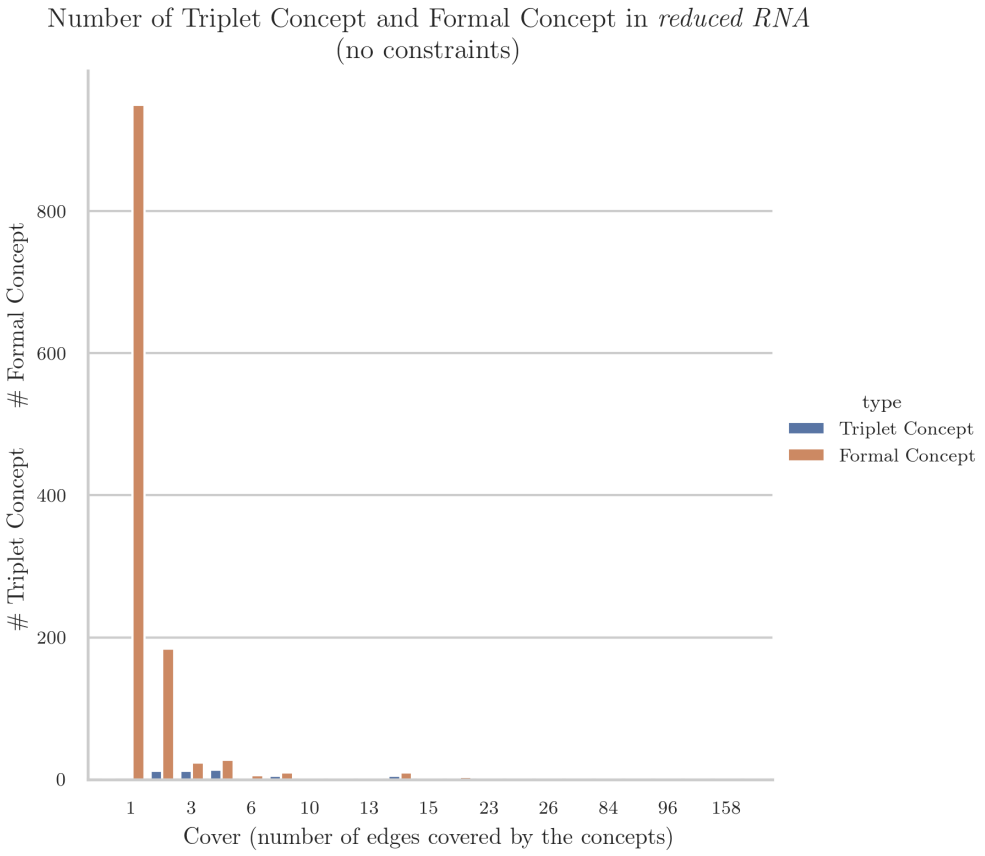


Figure 4.9 – Distribution of triplet concepts and formal concepts in the *reduced RNA* network according to their cover in edges.

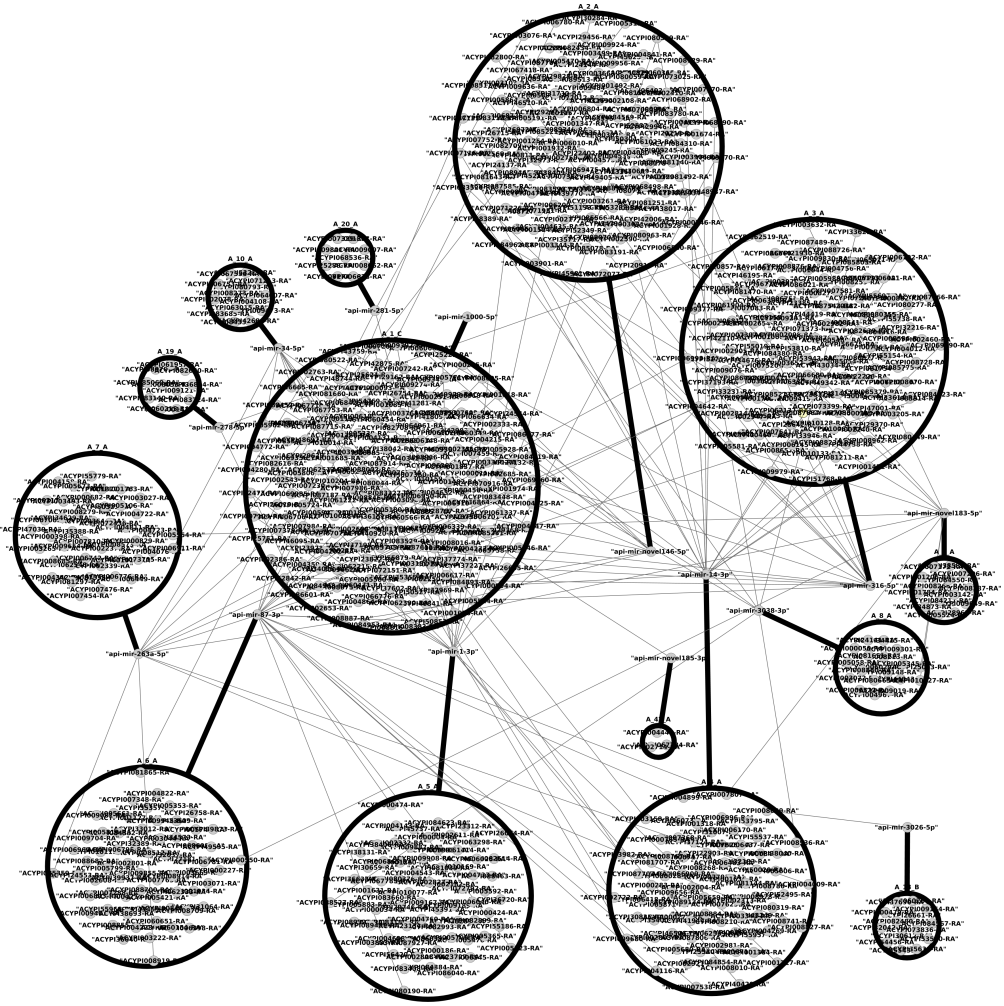


Figure 4.10 – Top-level compression of the largest triplet concepts found in *reduced RNA* network.

Chapter 5

Biseau: an ASP environment for high-level specification and visualization in graph theory

> We are stuck with technology when all we really want is just stuff that works. How do you recognize something that is still technology? A good clue is if it comes with a manual.
>
> Douglas Adams

This chapter is a reproduction of [17] with addenda. It introduces Biseau [1], a programming environment dedicated to the exploration of relations through a graphical display. The use of Answer Set Programming enables the production of small code modules which are easy to maintain and debug since they are very close to the specifications. It shows how a mathematical framework such as Formal Concept Analysis can be efficiently described at the level of its properties, without needing a costly development process.

Biseau can be considered as a by-product of this thesis but we have tried to make this general enough so that it can allow others to adapt a given code to the peculiarities of a data set, thereby speeding up the development of prototypes. Besides, it will also help the integration of the ideas of the FCA community in a readable and shareable format. We have to keep this chapter self contained so that it can be used independently of the rest of the thesis.

From a practical point of view, Biseau provides an Answer Set Programming to (graphviz) dot compiler, and uses the graphviz software [43] to render in real-time

1. Manual at `https://gitlab.inria.fr/lbourneu/biseau` !

the calculated graphs to user, for instance to produce concept lattices or AOC posets visualizations. Its relation with existing tools like LatViz and FCAbundles is also discussed.

The structure of this chapter is as follows. First, section 5.1 overviews our software contribution and the origin of Biseau, that section 5.2 introduces. Section 5.3 quickly presents the dot language and explains how Biseau takes advantage of dot and ASP to allow the user to build models. Section 5.4 proposes as a case study the reconstruction of the concept lattice from a formal context, already introduced in section 1.5. Section 5.5 shows how Biseau can easily handle some of the FCA extensions typically used in FCA applications such as knowledge processing [89]. Finally we conclude by some insights about Biseau interest when used in FCA and more generally, in graph theory.

5.1 Software contribution

Multiple programs [2] have been written during the thesis, mostly using Python and ASP, and when possible packaged, enabling their installation from the Python Package Index. Some of them are quickly presented here.

PowerGrASP, already presented in chapter 2, is a Python/ASP implementation of powergraph and derivatives, notably triplet concepts and quasi-bicliques. This program needed other developments, including `clyngor`, a Python package to enable the embedding of ASP into Python.

Many explorations of ASP usage in various contexts where developped, one of them being a collection of ASP codes to implement FCA-related tasks, like concept mining, lattice construction or AOC poset enumeration. Some examples of these encodings were already presented in section 1.4, but the increasing size of the collection, and the perspective of a library of ASP encodings for FCA, motivates us to work on a kind of ASP-for-FCA system.

The project was quickly reframed: if ASP could be used to describe a concept lattice, it would be simpler to enable the description of any graph, then if necessary

2. Listed at `https://lucas.bourneuf.net/programs`

to restrict it to concept lattices. The Biseau project is born from the observation that, if we can use ASP to describe a graph, it is possible to obtain a graphical representation of a graph from an ASP encoding. While there is no specific limitation to FCA, it is our first application for what became a general purpose tool in graph theory.

5.2 Biseau, a flexible environment for quick prototyping of concept structure search

Large scale data production requires availability of high-level visualizations for their exploration. This is usually performed by building generic visualization models, that users may later use to explore their data. Thus, software environments oriented towards data mining use efficient implementations of data structures and their visualizations. For instance, in Formal Concept Analysis, LatViz is a lattice visualization software, allowing end-user to explore the lattice structure efficiently [4]. Lattice Miner builds and visualize Galois lattices and provides data mining tools to explore data [68]. FCA Tools Bundle consists in a web interface exposing multiple FCA-related tools for contexts and (ternary) concept lattices exploration [64]. In-Close algorithm reference implementation provides a concept trees visualization of contexts encoded in standard formats [9]. All these tools work with a formal model that provides an abstract view and a fixed search space on the data. Users cannot work on the model itself, they are expected to use the implemented methods, not to design new ones. In contrast, this chapter introduces Biseau, a software focused on designing and exploring elements of the data structure, rather than the data itself. In this approach, data are only a support to the model validity, and the user's aim is the proper design of a general model. Biseau is a general purpose model builder that relies on graphs and logic languages.

Graphs are rendered in multiple ways, using field-specialized softwares like Cytoscape [99] in biology, graph-specific softwares (like LatViz for lattices), or more generalist like Tulip [10]. Another generic approach is dot, a graph description language specified by the graphviz software, which provides a gallery of visualization engines [43]. Dot is the internal graphical language used by Biseau.

Together with a graph data structure, Biseau offers through the use of Answer Set Programming (presented in section 1.4) a logical view of the associated exploration methods. It allows users to transcript the formal properties they are looking for in a straightforward way. In our approach, ASP is also used for visualization.

Biseau is supplied with a graphical user interface and a command line interface to write an ASP encoding. Biseau uses this encoding as a script to generate the dot files and the resulting visualizations. The main interest of Biseau is therefore to build graph visualizations directly from formal relations. Biseau is not only dedicated to lattices and their (efficient or scalable) exploration, it provides a general purpose programming environment to visualize any ordered structure. Biseau is therefore suited for rapid design and easy testing of works or extensions in the framework of FCA. It is freely available under the GNU/GPL license [3], and installable as any Python package with the command `pip install biseau-gui`. An annotated screenshot of the GUI is shown in Figure 5.1.

3. `https://gitlab.inria.fr/lbourneu/biseau`

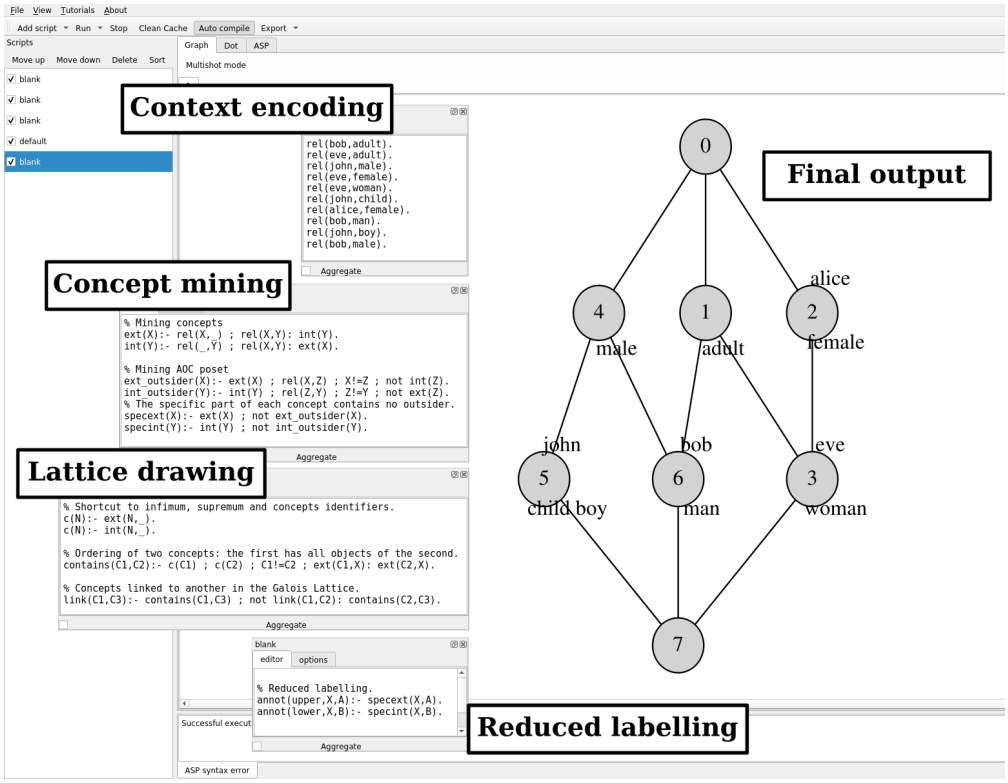


Figure 5.1 – An annotated screenshot of the graphical interface of Biseau. Four programs are shown. The first one is describing the *Vertebrate* formal context. The second one is searching for the concepts and the AOC poset. The third one is building the concept lattice, and the last one annotate the nodes of the resulting graph with the AOC poset. Each of these steps is described in section 5.4.

The running example of this chapter is presented in Table 5.1, along with its ASP encoding in Figure 5.2.

	adult	child	female	male	boy	woman	man
alice			×				
bob	×			×			×
eve	×		×			×	
john		×		×	×		

Table 5.1 – Formal context of human relations.

```
% Facts.
age(john,7). age(eve,71). age(alice,15).
male(john). male(bob). female(alice).
mother(eve,bob).
% Rules.
rel(H,child):- age(H,A) ; A<12.
rel(H,adult):- age(H,A) ; A>=18.
rel(H,male):- male(H).
rel(H,female):- female(H).
rel(H,man) :- rel(H,male) ; rel(H,adult).
rel(H,boy) :- rel(H,male) ; rel(H,child).
rel(H,woman):- rel(H,female) ; rel(H,adult).
rel(H,girl) :- rel(H,female) ; rel(H,child).
rel(H,adult):- rel(H,male) ; not rel(H,boy).
rel(H,female):- mother(H,_).
% Build the visualization in Figures 5.2 and 5.3.
link(O,A):- rel(O,A).
```

Figure 5.2 – ASP program encoding the context in Table 5.1, in the form of `rel/2` relations between objects and attributes. The last line yield links/2 atoms that are compiled by Biseau as edges in the output dot file.

5.3 Graph drawing With Biseau

Dot is a graph description language, allowing one to generate a graph visualization from the definition of its content [43]. Dot enables the control of precise visual properties, such as node and edge labelling, position, shape, or color. For instance, the dot line `woman [color="blue"]` will color in blue the node labelled *woman*.

```
Digraph biseau_graph {
  node [penwidth="0.4" width="0.1"];
  edge [penwidth="0.4" arrowhead="none"];
  john->boy; john->male; john->child;
  eve->female; eve->woman; eve->adult;
  bob->man; bob->adult; bob->male;
  alice->female;
}
```

Figure 5.3 – Dot encoding of the graph in Figure 5.4.

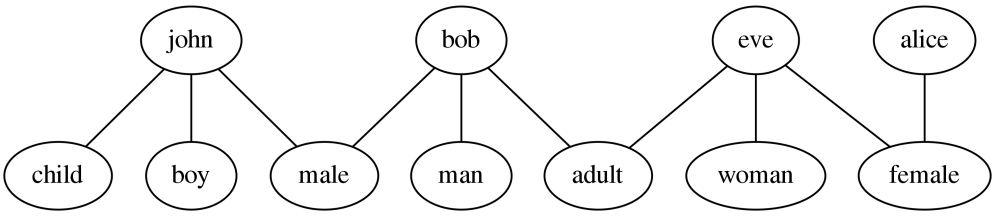


Figure 5.4 – Visualization of the relations described by context in Table 5.1.

The full language is defined by the graphviz graph visualization software, which provides multiple engines to interpret and compile dot encoded files to other formats, including images. Figure 5.3 shows an example of a working dot description, which given to a graphviz engine yields the visualization in Figure 5.4.

Biseau allows the user to interactively write some ASP encodings and display the corresponding graph visualization. To achieve this, it implements an ASP to dot compiler and a Graphical User Interface that helps writing the ASP encoding and that performs automatically all necessary compilations.

As explained in section 1.4, a given ASP encoding yields stable models consisting of true facts, which can be represented by atoms like `link(woman,human)`. For each stable model found from the ASP user encoding, Biseau will convert atoms into dot lines. For instance, the ASP atom `link(woman,human)` will translate to `woman -> human` in the dot output. This controlled vocabulary will be only partially explored in section 5.4, but note that it maps the full dot language, including colors, shapes, and general graph options. A complete documentation is available online [4].

The use of ASP allows the graph to be defined in intension: the ASP solver

4. `https://gitlab.inria.fr/lbourneu/biseau/blob/master/doc/user-doc.mkd`

infers all necessary relations for displaying the instantiated graph. More generally, Biseau internal process can be seen as a compilation from ASP models to dot, then from dot to image (the latter being delegated to graphviz software).

As a matter of example, the ASP encoding in Figure 5.2 will be compiled to the dot description in Figure 5.3, itself compiled to the image in Figure 5.4. If the ASP expression `color(A,blue):- rel(_,A).` was added to the ASP encoding in Figure 5.2, the final figure would show in blue all attributes nodes. The reader familiar with software engineering may recognize the use of ASP as a metamodel, and dot as the model.

Biseau can be extended with *scripts*, units of ASP (or Python) code to add to (or run on) the user encoding. They may expose some options to tune their behavior. Moreover, user can implement and add its own scripts to Biseau, allowing him to encapsulate ASP or Python programs that behave accordingly to his preferences. Biseau is shipped with scripts related to FCA, for data extraction from standard format like SLF or CXT, concept mining or lattice visualization (as shown in section 5.4).

5.4 Build and Visualize Concept Lattices With Biseau

This section shows how to build FCA basic mathematical relations in order to get a visualization of the Galois lattice in Biseau. The context in Table 5.1 will be used as case study, encoded in ASP using `rel/2` atoms as shown in the first five lines of Figure 5.2.

5.4.1 Mining Formal Concepts

In a formal context defined by objects O, attributes A, and the binary relation $R \subseteq O \times A$, a formal concept is a pair (X, Y), such as $X = \{o \in O \mid (o, y) \in R \ \forall y \in Y\}$ and $Y = \{a \in A \mid (x, a) \in R \ \forall x \in X\}$, where $X \subseteq O$ and $Y \subseteq A$. The search for formal concepts can be expressed in ASP like in the above definition:

```
1 ext(X):- rel(X,_) ; rel(X,Y): int(Y).
2 int(Y):- rel(_,Y) ; rel(X,Y): ext(X).
```

`rel(X,_)` fixes variable `X` as the first term of a relation, i.e. an object. Notation `rel(X,Y): int(Y)` ensures that there is a relation between `X` and all attributes of the intent. As a consequence, `ext(X)`, the extent, holds for all objects in relation with all attributes of the intent. ASP search comes with the guarantee that all minimal fixed points will be enumerated. Therefore, each answer set is a different concept, or the supremum or infimum (where extent or intent are empty sets).

These models/concepts can be aggregated in order to produce an encoding containing `ext/2` (and `int/2`) atoms, where `ext(N,A)` (`int(N,A)`) gives an element of N-th concept's extent (intent). This numbering is arbitrary and serves no other purpose than identifying the different concepts.

5.4.2 Concept Lattice

A concept lattice is defined by the partial inclusion order on the concept extents and intents, i.e. a graph with concepts as nodes, and an edge between a concept and its successors in the ordering.

```
1  % Shortcut to infimum, supremum and concepts identifiers.
2  c(N):- ext(N,_).
3  c(N):- int(N,_).
4  % Ordering of two concepts: the first has all objects of the second.
5  contains(C1,C2):- c(C1) ; c(C2) ; C1!=C2 ; ext(C1,X): ext(C2,X).
6  % Concepts linked to another in the Galois Lattice.
7  link(C1,C3):- contains(C1,C3) ; not link(C1,C2): contains(C2,C3).
8  % Annotate nodes with extent and intent.
9  annot(upper,X,A):- ext(X,A).
10 annot(lower,X,B):- int(X,B).
```

These lines yield the visualization shown in Figure 5.5. Line 2 and 3 are here to enable the access to the infinum, supremum and concepts with one atom. Line 5 yields pairs of concepts that are included, based on their extent. Line 7 ensures that a link exists in the lattice between a concept $C1$ containing another concept $C3$ if there no link between $C1$ and a concept $C2$ smaller than $C3$. Finally, the `annot/3` atoms are a Biseau convention (just as `link/2` that define an edge in the

dot output), allowing us to print the extent and intent of each concept, respectively above and below the node.

5.4.3 Reduced Labelling

The reduced labelling of a lattice is computed as the set of specific objects and attributes for each concept. This is easily defined as `specext/1` and `specint/1` atoms in ASP, using the following lines along the search for formal concepts in section 5.4.1:

```
% An outsider is any object or attribute linked to
%  an attribute or object not in the concept.
ext_outsider(X):- ext(X) ; rel(X,Z) ; X!=Z ; not int(Z).
int_outsider(Y):- int(Y) ; rel(Z,Y) ; Z!=Y ; not ext(Z).
% The specific part of each concept contains no outsider.
specext(X):- ext(X) ; not ext_outsider(X).
specint(Y):- int(Y) ; not int_outsider(Y).
```

With these lines and the collapsing into one model described in section 5.4.1, we obtain `specext/2` and `specint/2` atoms, describing the AOC poset elements, attached to each concept. We can then compute the reduced labelling of the lattice with the following lines, replacing the previously defined annot/3 definitions in section 5.4.2:

```
% Minimalist annotation of nodes with their extent/intent:
annot(upper,X,A):- specext(X,A).
annot(lower,X,B):- specint(X,B).
```

Using these definitions, Biseau produces the visualization shown in Figure 5.6.

Enumeration of the AOC poset

The following integrity constraint, added to the concept generation with AOC poset seen in section 5.4.3, will prevent concepts that do not belong to the AOC poset to be generated.

```
:- not specext(_) ; not specint(_).
```

This is achieved simply by invalidating the model if there is no atom indicating an object-concept or an attribute-concept.

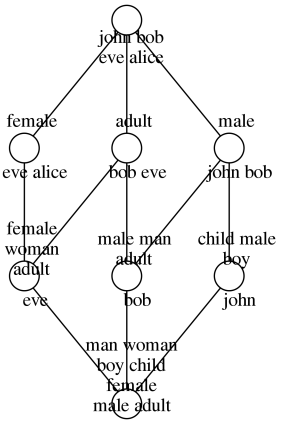


Figure 5.5 – Visualization of the concept Lattice of context in Table 5.1 using Biseau, with extent and intent shown for each node/concept.

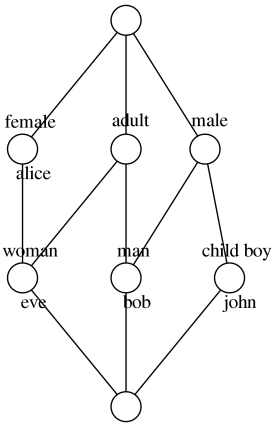


Figure 5.6 – Visualization of the concept Lattice of context in Table 5.1 using Biseau, with reduced labelling.

5.5 Pulling Constraints On The Model

This section exposes the implementation in ASP and Biseau of some FCA variants and extensions often used in knowledge processing [89].

5.5.1 Object and Property Oriented Concept Lattices

Object and property oriented concept lattices are special kind of lattices inspired from rough set theory. They have been shown isomorphic, and holding complementary information about the formal context they are issued from [108].

Following definitions from [111], it is also possible to encode the mining of object oriented concepts (X, Y) defined by $X = Y^{\Diamond}$ and $Y = X^{\Box}$, such as:

$$Y^{\Diamond} = \bigcup_{y \in Y} Ry \qquad X^{\Box} = \{y \in A | Ry \subseteq X\}$$

With $Ry = \{x \in O | (x, y) \in R\}$.

```
% Any object linked to an attribute in the intent is in the extent.
ext(X):- rel(X,Y) ; int(Y).
% Objects in the complementary set of the extent.
not_ext(Nx):- rel(Nx,_) ; not ext(Nx).
```

```
5 % Attributes solely linked to objects of the extent are in the intent.
6 int(Y):- rel(_,Y) ; not rel(Nx,Y): not_ext(Nx).
```

The code for property-oriented concepts is similar, and both replace the encoding in section 5.4.1.

5.5.2 Iceberg Lattices

Introduced in [104], they are a representation of lattices such as only the most structuring concepts are shown. The iceberg lattice is defined as the Galois lattice stripped of all concepts with a too small support (i.e. number of objects in their extents). It can be built by discarding any model containing too few objects in its extent.

For instance, Figure 3 of [104], reproduced in this paper in Figure 5.7, shows the iceberg lattice of the running example MUSHROOMS database of `nbobj` objects with a minimal support of `minsupp`%. It can be reproduced by discarding models using an integrity constraint:

```
1 #const minsupp=85.  % minimal number of objects
2 % Total number of objects in the context.
3 nb_obj(N):- N=#count{X: rel(X,_)}.
4 % Number of objects in the extent must not fall behind a minimum.
5 :- {ext(_)} < N*minsupp/100 ; nb_obj(N).
6 % Use the specext/1 atom to indicate the percent of objects,
7 %  to reproduce the labelling of the iceberg lattice.
8 specext(P) :- N={ext(_)} ; P=N*minsupp/100.
```

The first line introduces a parameter providing the minimal support for a concept. As ASP does not handle floats, it must be provided as an integer. Line 3 computes the number of objects in the formal context. Line 5 is the constraint preventing any formal concept with too few concepts to be yielded. The last line is a slight hack of `specext/1` to, instead of showing the full extent of the object-concept, to display the percent of objects belonging to the concept.

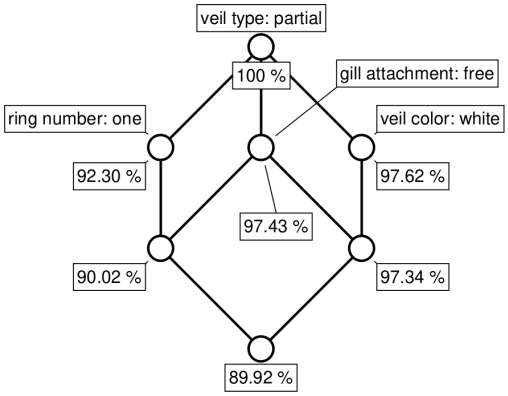


Figure 5.7 – Iceberg lattice with a minimal support of 85% of the MUSHROOMS database. Figure extracted from [104].

5.5.3 n-adic FCA

N-adic FCA is an extension of FCA increasing the number of dimensions considered [72]. Instead of 2-dimensional formal contexts, n-adic FCA now uses n dimensions, and therefore formal contexts are maximal relations of n subsets.

We show here that it can be encoded the same way as regular FCA, by extending the number of parameters for `rel` atoms. For instance, in triadic FCA, conditions are given as the third argument of `rel/3` atoms, such as `rel(O,A,C)` is true when the relation between object O, attribute A and condition C holds. Triadic concepts (not to be confused with triplet concepts introduced in chapter 3) can thus be generated using the following encoding:

```
1 ext(X):- rel(X,_,_) ; rel(X,A,C): int(A), cnd(C).
2 int(X):- rel(_,X,_) ; rel(O,X,C): ext(O), cnd(C).
3 cnd(X):- rel(_,_,X) ; rel(O,A,X): ext(O), int(A).
```

5.5.4 Pattern Structures

As introduced in [44], a pattern structure is a generalization of FCA applied on attributes structured in semi-lattices. Pattern concepts are pairs of objects and semilattices, producing the expected pattern lattice. It has been applied in particular to gene expression data with integer interval patterns [63]. Here, we reproduce the pattern lattice construction for an example of non-binary data from

the same publication:

```
1 rel(1,1,5).  rel(1,2,7).  rel(1,3,6).  % 5 objects
2 rel(2,1,6).  rel(2,2,8).  rel(2,3,4).  % 3 situations
3 rel(3,1,4).  rel(3,2,8).  rel(3,3,5).  % one value from 4 to 9
4 rel(4,1,4).  rel(4,2,9).  rel(4,3,8).
5 rel(5,1,5).  rel(5,2,8).  rel(5,3,5).
```

Note that data are encoded in `rel/3` atoms over 5 objects, 3 conditions, and expression values in the interval $[4; 9]$ associated with a given gene and condition, such as `rel(O,S,V)` holds when object O in situation S has an expression value of V. Similarly to section 5.4.1, we can enumerate the pattern concepts:

```
1  % Choose a subset of objects as the extent.
2  { ext(O): rel(O,_,_) }.
3  % The intervals of extent.
4  interval(C,Min,Max):-
5       rel(_,C,_) ;
6       Min=#min{V,O: rel(O,C,V), ext(O)} ;
7       Max=#max{V,O: rel(O,C,V), ext(O)}.
8  % Object is valid on Condition.
9  valid_on(O,C):- rel(O,C,V) ; interval(C,Min,Max) ; Min<=V ; V<=Max.
10 % Object is valid for all Conditions.
11 valid(O):- rel(O,_,_) ; valid_on(O,C): rel(_,C,_).
12 % Avoid any model that does not include maximal number of objects.
13 :- not ext(O) ; valid(O).
```

The use of the meta-programming directives `#min` and `#max` allows us to retrieve the minimal and maximal value associated to the extent. Therefore, the predicate `interval(C,Min,Max)` stands for the minimal and maximal values on condition C, e.g. 5 and 6 for condition 1 when extent is $\{1, 2, 5\}$. Unlike the concept model seen in Section 5.4.1, this model relies on an explicit choice rule for the extent with subsequent constraints to ensure its maximality. Line 2 generates an answer set for each element of the power set of the object set. Following lines will discard answer sets that are not infinum, supremum or concept. Line 4 associates for each condition the minimal and maximal values over the extent. Line 7 selects an object and a condition such as they are associated to a value in the interval. Line 9 selects all objects that are valid for all conditions, and line 11 ensures that they belong

to the extent.

The code in section 5.4.2 can be reused without modifications to produce and show the resulting pattern lattice.

5.6 Conclusion

Using the ASP language in the Biseau environment, some well-known FCA structures (concept, object-oriented, iceberg, integer pattern lattices) have been reconstructed. The main contribution of Biseau lies into the straightforward use of the structure specifications to produce a simple code and a proper visualization. To achieve that feat, Biseau is compiling a controlled subset of ASP atoms to dot lines, effectively building a dot formatted file that is compiled to an image by graphviz software. By letting the user manipulate the visualization with the full power of ASP, Biseau enables the definition of graphs in intension. This gives an abstract access to dot expressions and lets the user focus on the fast prototyping of data exploration and the elaboration of mathematical properties. In other words, Biseau allows user to work on the model processing data, instead of providing an implementation of a single model to be used on particular data, as usually performed in field-specialized software.

ASP-related Limits

ASP limits lies into the absence of float numbers handling, and scaling problems inherited from the total grounding of data before solving. However, Potassco system users may benefit from several extensions of the language like linear programming [77], propagators [51], or the integration of Python scripts through a dedicated API [47], allowing one to take advantage of other programming paradigms. It is also possible to improve performances by an iterative replacement of bottlenecks by dedicated algorithms. For instance, the concept mining can be replaced by an implementation of the in-close algorithm [8].

Biseau is not limited to FCA

While this thesis focuses on the reconstruction of FCA visualization using Biseau,

nothing prevents further developments towards other fields of application. For instance, software engineering, to draw UML graphs using ASP as a domain specific language, or semantic web, to post-process and visualize SPARQL query results, or bioinformatics, to draw and explore biological networks such as boolean networks.

Chapter 6

Conclusion and perspectives

This chapter summarizes the main contributions of this thesis and details a number of research opportunities that we believe could usefully complement or extend this work.

6.1 Conclusion

This thesis proposes a framework and practical programs for lossless graph compression for visualization purposes. It elaborates on a useful practical tool, Power Graph Analysis, which has demonstrated its relevance on many applications, especially bioinformatics. We list here our main contributions, divided into four areas.

Power Graph Analysis and Formal Concept Analysis

In chapter 2, we presented a formalization of the search space of Power Graph Analysis. We have shown two versions of this search space, whose exponential size with respect to the size of the graph leads to NP-completeness of the search. We then explained the Royer *et al.* heuristic to reach practically good approximations in this space, and presented our own optimizations to reduce it. We also introduced our implementation of a search for approximated results, *PowerGrASP*, along general optimizations that could be ported to other implementations of the powergraph compression. We also expose the concept-cycle motif, which presence in the graph precludes a greedy approach to reach an optimal compression, and could thus be an interesting extension of Power Graph Analysis.

Triplet Concepts

Introduced in chapter 3, triplet concepts formalize the overlapping biclique mo-

tif, where the two sets of nodes of a biclique are overlapping, thus defining a clique integrated in the biclique. This contribution allows to unify the search for bicliques and cliques proposed by Power Graph Analysis. In FCA, triplet concepts are a subset of the formal concepts issued from the reflexive graph context. We have shown different methods to yield the triplet concepts, starting from different representations of the input data (graph contexts, or concepts issued from it). The search for triplets as a motif has been integrated in *PowerGrASP*, and some preliminary results on it have been shown on the two main biological datasets. Triplet concepts also represent the maximal cliques of a graph in a compact way.

Applications

In chapter 4, we validated our concept-based approach (implemented in *PowerGrASP*) by reproducing the results of Power Graph Analysis. We have also shown the performance gain of the different optimizations we have proposed. Finally, we studied the presence of triplet concepts in the datasets, showing that (1) using triplet concepts is equivalent to using concurrently bicliques and cliques, and (2) that the search of triplet concepts with its three populated sets enabled us to unravel a particular structure that was hidden in the graph and in the powergraph only constituted of cliques and bicliques.

Software contribution

We have presented in chapter 5 Biseau, a general purpose software in graph theory we used to reproduce main objects of Formal Concept Analysis, such as concept lattice and derived structures. With this particular field of application, we hope we have shown that Biseau can be used for a wide range of applications as an exploratory tool for graph theory-based work.

6.2 Perspectives

Numerous developments and open problems would help to extend and deepen our work. We have organized the perspectives into several subsections, starting by theoretical issues, then applications and developments that are suggestions for further research.

6.2.1 Triplet-based Power Graph Analysis

Our proposal of explicitly representing overlapping bicliques in a graph through triplet concepts seems a powerful framework to understand and extend the graph compression process implemented in Power Graph Analysis, which is normally using two motifs, bicliques and cliques.

The largest triplet concept search has been integrated in our own implementation of this compression process, *PowerGrASP*. In the current version, triplet concepts (A, B, C) are represented simply by creating a powernode for each set and using four poweredges ($A \times B$, $A \times C$, $B \times C$ and $C \times C$). Such a direct representation is however not optimal and we lacked time for a more sophisticated approach. Our first suggestion wold be tying to group A and B sets in a common powernode, thus allowing for a more compact representation (see Figure 3.5). While it would not necessarily be feasible in all cases (sets A and B may be in different powernodes and cannot be clustered without breaking the hierarchy condition), it would provide an exact equivalent representation as a search using only bicliques and cliques. To increase the compression ratio over the standard biclique+clique search, one could even use a single-edge representation of a triplet by introducing overlapping powernodes (see Figure 3.4). However, as this compact representation is based on overlaps, it could quickly became a nightmare to understand because of the overlapping parts being in interaction. Our idea to prevent this spaghetti problem is to introduce a new constraint, ensuring that triplets are compressed in the single-edge representation if and only if the C set belongs to a hierarchy, i.e. the C set part is reused either completely or not at all by other motifs.

Open-problems

We end with three interesting open problems that the triplet concept approach has raised.

— How to find a maximal set of (triplet) concepts that are *necessary*, i.e. used in all optimal compression.

— Can one predict, without processing the compression itself, the number of poweredges that would result from compressing a given list of (triplet) con-

cepts in a given order ?

— Can one adapt the normal form of triplets, or the definition of meet and join, to obtain a lattice out of triplet concepts ?

6.2.2 Triplet concepts: admissibility = edge-maximality ?

We defined the *admissible triplets*, i.e. triplets candidates for edge-maximality: using global conditions, some non edge-maximal triplets can be discarded without comparing them to other triplets, thus reducing greatly the amount of triplets yielded by the various generation methods (see section 3.6).

One can wonder if admissibility is synonymous to edge-maximality. On all the tested cases, with all generation methods, the admissibility constraints were filtering out all non edge-maximal triplets. From this intuition, we propose conjecture 6.1 that remains to be established.

Conjecture 6.1. *Maximality of admissible triplets*
An admissible triplet is edge-maximal.

If this conjecture holds true, the triplet concepts would be easier to mine, since the admissibility constraints does not require a complete comparison of all pairs of triplet concepts. It is also possible that our definition of admissible triplets requires to be augmented with more constraints, for the conjecture to be verified.

6.2.3 Properties of (non) optimal powergraphs

Finding the optimal powergraphs is a NP-complete task. We however observed that (non) optimal powergraphs presents some peculiarities, that may be used to recognize them as such. This subsection exposes properties we found. More may exist.

Conjecture 6.2. *Optimal powergraph*
Let $\mathcal{P} = (V, E, H)$ be a powergraph of the graph G. if $\forall p \in V$ we have $degree(p) = 1$, then $\mathcal{P}$ is optimal.

With conjecture 6.2, we explore the possibility to detect whether a powergraph is optimal. For this specific conjecture, we observe that any powergraph where all (power) nodes have a single neighbor is optimal. This property can only be held by graphs with no motif overlap, since an overlap will require a motif set to be divided in multiple powernodes. The (power) nodes linked to that set will necessarily be linked to all produced powernodes, hence having a degree higher than 1.

While conjecture 6.2 only applies for a very small family of graphs, we suppose that this type of easily computable property on powergraphs may help for the compression task, for instance during the choice of concepts that could introduce suboptimal constructs recognizable as such. Next paragraphs introduce a discussion and another conjecture on the property of optimally compressed powergraphs.

Link between edge reduction and conversion rate

Edge reduction and conversion rate are two correlated metrics introduced by [94] to measure quantitatively the Power Graph Analysis efficiency to simplify graphs. Edge reduction indicates the compression ratio of edges, whereas the conversion rate is a measure of the mean reduction of edges compared to the number of powernodes introduced. With conjecture 6.3, we propose that an optimal powergraph is optimal both in edge reduction and in conversion rate. This conjecture would hold if the powernode hierarchy and the amount of powernodes are (at least partly) determined by the chosen motifs.

Conjecture 6.3. *Conversion rate = edge reduction*
An optimally compressed powergraph (i.e. having a maximal edge reduction) has a maximal conversion rate.

Modification of the powergraph

We also suppose that detecting specific non-optimal parts of a powergraph may be the basis for a post-process that tweaks the powernode hierarchy and the poweredges of an existing powergraph in order to improve its edge reduction. For instance, such operations are the merging of mergeable powernodes, or the search and optimal compression of the concept-cycle motif (which however precludes greedy approaches).

Note that changing the hierarchy of powernodes requires to also change the poweredges cover to conserve the underlying data. The rules of transformation may be quite simple, but still needs to be formalized as a algebra over the powernode hierarchy and poweredges cover, i.e. trees and graph.

6.2.4 Search strategies to achieve better compression

The search strategy for graph compression, presented in chapter 2, is relying on the ordering of concepts by decreasing cover. We propose here two variations upon this strategy, aiming at achieving better compression, and a final paragraph on our implementation of an exact algorithm for optimal powergraph compression.

Beam search: looking n step ahead

A natural variant search space exploration method is to rely on the prediction of the decomposition of later concepts in order to predict the best concept to compress at a given step. A concept could therefore not only be selected with respect to its edge cover, but also because it will favor future selection of formal concepts, leading to a smaller number of poweredges. In our compression algorithm (see Algorithm 3), the function selecting the concept to compress (*best_concept()*) may involve a recursive call to the compression algorithm itself in order to explore the consequences of the choice.

Graph motif: predicting compression effect

The selection-transformation method, presented in section 2.4.2, proposes a select-then-transform approach, where concepts to compress are first reduced and ordered, then compressed according to the order. One can devise another method, where the search space is first encoded as a *motif graph* in order to compute the worst-case effects of transformations before performing the selection. Intuitively, each node of the motif graph is a compressible motif, and there is a link between two nodes if the compressible motifs are not independent. One first obvious use of that graph is to detect a stable. All nodes of a stable in the motif graph are compressible without interference.

The motif graph can also be used to explore the consequences of the compres-

sion of a particular concept. Let each edge (x, y) have a valuation corresponding to the number of poweredges necessary to compress y if x is compressed. When a neighbor x_1 of y is compressed, the number of powernodes composing y may increase. As such, the number of poweredges created by the compression of y increases. More generally, the motif graph encodes the poweredge cost of the compression, and may be used to predict efficiently if a given ordering of concepts is better than another. Finally, detecting communities of compressible motifs in this graph (by compressing it, for instance), may unravel specific parts of the graph where heuristics fail to obtain an optimal compression, and enable to decide a better compression for the overall community.

ASP implementation of a brute-force optimal compression

We wrote an ASP implementation that searches for the optimal compressions with a brute-force approach. It is able to enumerate them, but is however quite complex and will not be detailed here. It is accessible freely for reference [1]. The approach is however quite limited in is current version.

First, the encoding does not deal well with the numerous symmetries, especially because of implementation details which requires powernodes to be identified by pairs of nodes. For instance, a clique of 5 elements will yield more than 20 optimal solutions. All describe a single powernode with a reflexive poweredge and 5 nodes inside, with sole difference being the choice of the 2 nodes that identifies the sole powernode.

The performances are low, especially regarding memory usage. For instance, a single clique of 50 nodes requires more than 15 Go of memory and at least 3 minutes of computation to be solved. With a graph of 10 nodes and a density of 0.2, 8 Go and 2 minutes are needed.

More work is necessary to determine how to get rid of symmetries, and if clever constraints on the model may improve its efficiency.

1. `lucas.bourneuf.net/thesis`

6.2.5 Justifiable graph decomposition

Conceptual clustering is a clustering method where found clusters may be discarded because they do not fulfill some (statistical) conditions. Applied to Power Graph Analysis, this could for instance change the concept choice heuristic, by selecting a motif to compress only if it leads to the creation of clusters with high enrichment.

An obvious first application is in protein-protein interaction networks. The compression process would rely on annotations on proteins encoding their roles, and, using an hypergeometric test, would discard any motif which clusters likelihood does not meet a given p-value requirement.

The output powergraph would then be justified by data annotations: powernodes encoding statistically strong modules may help the understanding of the overall (power) graph structure.

6.2.6 Quasi-motifs

As described in section 2.2.3, quasi-motifs, i.e. motifs with few missing edges, can be used to reveal larger relations. *PowerGrASP* supports quasi-bicliques as motif to be searched and compressed, but no application of these features was shown. Future development in that direction for this software should enable the definition of quasi-triplets, and one obvious application is the search for quasi-motifs in data presenting less stars and more complex and incomplete structures.

More generally, quasi-motifs introduces the notion of *exception*, i.e. constructs that are not strictly corresponding to existing data in the input graph. Their presence precludes the compression to be *lossless*, unless they are added to the final representation. In the case of quasi-motifs, the exceptions are edges missing in the input graph. Graphically, these quasi-motifs could be rendered as normal motifs, augmented by a set of edges indicating those implied by the poweredges, but not present in the input data.

Quasi-triplets

Likewise to [74] which defines μ-tolerance bicliques, a μ-tolerance quasi-triplet

could be defined as per definition 6.1, where μ limits the missing edges in the overall triplet.

Definition 6.1. *μ-tolerance quasi-triplet*
Let $T_\mu = (V_1, V_2, V_3)$ be a quasi-triplet, and μ a small integer number. Then T_μ is a μ-tolerance quasi-triplet if for each $v \in V_i$, $i = 1$ or 2 or 3,

1. v is disconnected from at most μ number of vertices in $V_j \cup V_k$, $j \neq k \neq i$,
2. v is adjacent to at least μ number of vertices in $V_j \cup V_k$

It is easy to imagine a finer definition involving different limits for the missing links, allowing for instance a lot of missing links in the biclique formed by sets A and B, but requiring very few missing links in C.

6.2.7 AOC poset

Because of the possibly exponential number of concepts, some applications of FCA proposed to limit themselves to the AOC-poset instead of working on the full lattice [33, 55], since the growth of the AOC-poset is linearly bounded, and its extraction from the data is achievable in time bounded by the size of the context [13].

Such a simplification could be tested on Power Graph Analysis. Apart for diminishing the number of concepts to consider, it seems adapted to graphs essentially composed of stars such as the *aphid RNA* network presented in chapter 4. Indeed, the AOC poset contains only bicliques covering at least all edges adjacent to a node, typically stars. On the other hand, bicliques covering only strict subsets of adjacent edges were needed to reach an optimal compression in some examples we shown (see for instance the graph in Figure 2.3). In other word, by considering only the AOC-poset, optimality cannot be guaranteed and a good approximation is likely to be reachable only in graph involving mostly stars (see for instance the AOC-poset-first compression shown in Figure 2.6).

AOT-poset

The object triplets and the attribute triplets, the triplet counterparts of object

and attribute concepts, remain to be defined properly, which will only be possible once a proper definition of the triplet concept lattice will be proposed. The obtained Atribute Object Triplet poset (AOT poset) structure may propose the same advantage as the AOC poset regarding the minimization of the number of motifs to consider, along the interests of triplet concepts regarding the unraveled graph motifs.

6.2.8 Strict MCE from triplet concepts

We have proposed an algorithm building the maximal cliques of a graph from its triplet concepts. In that regard, triplet concepts act as a compressed version of cliques (see section 3.10).

Two main questions remain. First, is it possible to enumerate maximal *bicliques* using a variation of the same method ? That is, from the triplet concepts ? Second, is it possible to use the triplet concepts as an efficient representation of the sole maximal cliques ?

6.2.9 Benchmark of triplet concept enumeration methods

The generation of random contexts, used in section 3.9.5 to compare the different implementations of the triplet concept search task, has been made using a naive uniform randomization that targets a given density.

Because of the particular structure found in real data, a completely random context and therefore our benchmark is not representative of real world data, which may present some global properties. A better generation, not uniformly random and mimicking the structure of real data, may impact the behavior of the triplet concept enumeration programs.

For instance, the small-world property, where the distribution of nodes' degree follows a power-law, are often found in bioinformatics and social network graphs. Graph contexts derived from this type of graph would improve the relevance of the benchmark. Another method could rely on a synthetic formal context generator dedicated to FCA tasks [91].

6.2.10 Further optimizations for powergraph compression

We described some methods to reduce the compression search space (see section 2.7). We propose here some ideas for other optimizations methods that may enable a better optimization of *PowerGrASP* or other implementation of the compression process, in order to compress larger networks, including *extended MatrixDB* in our applications.

Graph filtering for triplets

Section 2.7.2 described the graph filtering optimization for cliques, bicliques and stars. Having an equivalent for triplets may greatly decrease the amount of edges to consider in a graph. Since a triplet that is not a biclique or a star is made of triangles, a good filter for a non-biclique triplet may be to discard any node not involved in at least one triangle (this is similar to the graph filtering for cliques).

Subgraph postponing

Some graph families may be easier to compress. Intuitively, a tree may be compressed efficiently with stars centered on the parent, and children clustered in a powernode. One can detect and ignore them in a first step, and compress them, if necessary, once the full graph has been treated.

However, as for the concept-cycle motif we described in chapter 2 that a greedy approach cannot compress optimally, a tree needs a particular compression process to be compressed in a readable way. In fact, a tree being readable by nature, there is even little interest into compressing it.

Some other graph families may be detected and compressed efficiently with a dedicated algorithm exploiting their properties. An example of such graph family is lattices, which cannot have any biclique with 3 or more elements in both sets.

More generally, the compression of subgraphs having special properties may be postponed to after the compression of the graph without these easily compressed parts.

Hierarchy-free graph first

Most optimizations we presented were only valid when treating an uncompressed graph. But as soon as hierarchy is introduced, the optimizations are deactivated

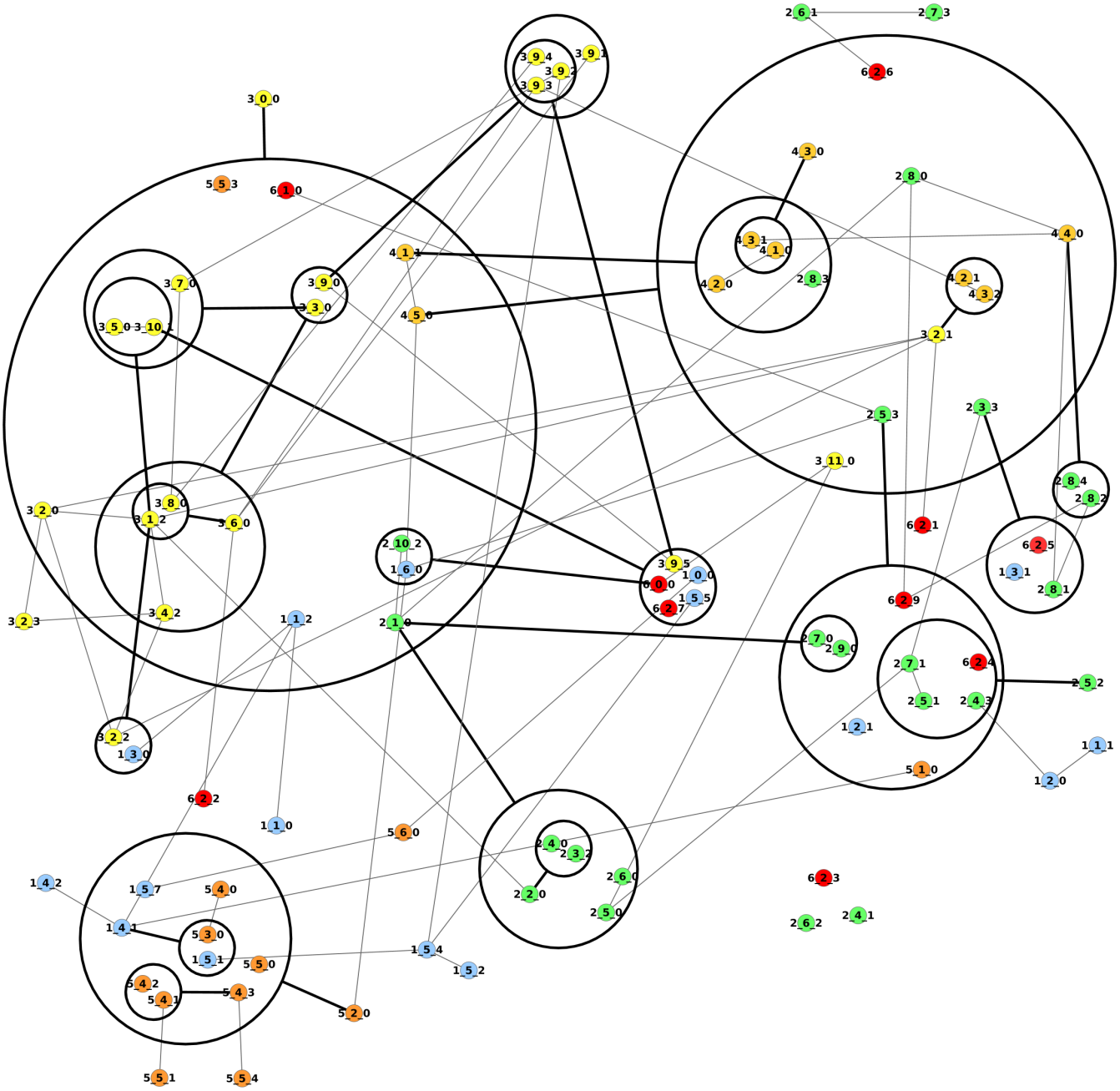
because the properties they exploit cease to hold on a powergraph. This drawback may be avoided by anew two-step approach. It consists into a first compression pass where the already compressed powernodes and their neighborhood are ignored, then a second step where the graph is refined to take into account the powernode hierarchy constraint.

6.2.11 Biseau

We presented in the final chapter an important software contribution we developed and applied to FCA, Biseau. Because of the overall complexity of this program, there is room for supplementary developments and features. Since we hope that software will be useful for later users, we have provided an online repository that we hape will help further developments. We list here those that seems to us of greatest importance.

Biseau is mainly developed as an interface to assemble small scripts performing specific and parametrizable tasks, for instance concept mining and lattice generation. A possible extension is to provide an automated system to easily publish and retrieve scripts from other users, i.e. a centralized library of scripts allowing the community of users to easily share scripts.

To improve significantly the user experience on Biseau and its graphical user interface, developments on UX testing, error management, and features typically found in text editors and IDEs could help user to write, understand and debug produced ASP code. Also, implementing Biseau support for more graph description languages such as GML or GEXF would enable the outsourcing of the visualization to other (field-)specialized software.



The compressed section graph of this thesis, where each node is a (sub)section, and with a link between two sections whenever there is a reference from one to the other (direct reference, or reference of figure/table). Node color depends of the referenced chapter. Nodes without edges are sections without any references to other sections.

Bibliography

[1] James Abello, Alex J Pogel, and Lance Miller, « Breadth First Search Graph Partitions and Concept Lattices. », *in*: *J. UCS* 10.*8* (2004), pp. 934–954.

[2] P. K. Agarwal et al., « Can visibility graphs Be represented compactly? », *in*: *Discrete & Computational Geometry* 12.*3* (1994), pp. 347–365, ISSN: 1432-0444, DOI: 10.1007/BF02574385, URL: http://dx.doi.org/10.1007/BF02574385.

[3] Sebastian E Ahnert, « Generalised power graph compression reveals dominant relationship patterns in complex networks », *in*: *Scientific reports* 4 (2014).

[4] Mehwish Alam, Thi Nhu Nguyen Le, and Amedeo Napoli, « Steps Towards Interactive Formal Concept Analysis with LatViz », *in*: *FCA4AI@ECAI*, 2016.

[5] Gabriela Alexe et al., « Consensus algorithms for the generation of all maximal bicliques », *in*: *Discrete Applied Mathematics* 145.*1* (2004), Graph Optimization {IV}, pp. 11–21, ISSN: 0166-218X, DOI: http://dx.doi.org/10.1016/j.dam.2003.09.004, URL: http://www.sciencedirect.com/science/article/pii/S0166218X04000629.

[6] J Amilhastre, M.C Vilarem, and P Janssen, « Complexity of minimum biclique cover and minimum biclique decomposition for bipartite domino-free graphs », *in*: *Discrete Applied Mathematics* 86.*2–3* (1998), pp. 125–144, ISSN: 0166-218X, DOI: http://dx.doi.org/10.1016/S0166-218X(98)00039-0, URL: http://www.sciencedirect.com/science/article/pii/S0166218X98000390.

[7] Simon Andrews, « A 'Best-of-Breed' Approach for Designing a Fast Algorithm for Computing Fixpoints of Galois Connections », *in*: *Inf. Sci.* 295.*C*

(Feb. 2015), pp. 633–649, ISSN: 0020-0255, DOI: 10.1016/j.ins.2014.10.011, URL: https://doi.org/10.1016/j.ins.2014.10.011.

[8] Simon Andrews, « In-close, a fast algorithm for computing formal concepts », *in*: *International Conference on Conceptual Structures*, 2009.

[9] Simon Andrews and Laurence Hirsch, « A tool for creating and visualising formal concept trees », *in*: *CEUR Workshop Proceedings*, vol. 1637, Tilburg University, 2016, pp. 1–9.

[10] David Auber et al., « TULIP 5 », *in*: (Aug. 2017), ed. by Reda Alhajj and Jon Rokne, pp. 1–28, DOI: 10.1007/978-1-4614-7163-9_315-1, URL: https://hal.archives-ouvertes.fr/hal-01654518.

[11] Marie-Aude Aufaure and Bénédicte Le Grand, « Advances in FCA-based applications for social networks analysis », *in*: *International Journal of Conceptual Structures and Smart Applications (IJCSSA)* 1.*1* (2013), pp. 73–89.

[12] Jocelyn Bernard and Hamida Seba, « Résolution de problèmes de cliques dans les grands graphes », *in*: *EGC 2018*, Paris, France, Jan. 2018, URL: https://hal.archives-ouvertes.fr/hal-01886724.

[13] Anne Berry et al., « Hermes: a simple and efficient algorithm for building the AOC-poset of a binary relation », *in*: *Annals of Mathematics and Artificial Intelligence* 72.*1* (Oct. 2014), pp. 45–71, ISSN: 1573-7470, DOI: 10.1007/s10472-014-9418-6, URL: https://doi.org/10.1007/s10472-014-9418-6.

[14] Maciej Besta and Torsten Hoefler, « Survey and Taxonomy of Lossless Graph Compression and Space-Efficient Graph Representations », *in*: *CoRR* abs/1806.01799 (2018), arXiv: 1806.01799, URL: http://arxiv.org/abs/1806.01799.

[15] Georg Boenn et al., « Anton: Answer Set Programming in the Service of Music », English, *in*: *Proceedings of the Twelth International Workshop on Non-Monotonic Reasoning*, ed. by M Pagnucco and M Thielscher, Also Tech report UNSW-CSE-TR-0819, University of New South Wales, Sept. 2008, pp. 85–93.

[16] Daniel Borchmann, « A Generalized Next-Closure Algorithm–Enumerating Semilattice Elements from a Generating Set », *in*: *arXiv preprint arXiv:1111.3975* (2011).

[17] Lucas Bourneuf, « An Answer Set Programming Environment for High-Level Specification and Visualization of FCA », *in*: *FCA4AI'18 - 6th Workshop "What can FCA do for Artificial Intelligence?"*, Stockholm, Sweden, July 2018, pp. 1–12, URL: https://hal.archives-ouvertes.fr/hal-01945938.

[18] Lucas Bourneuf, « Biseau: An Answer Set Programming Environment for High-Level Specification and Graph Visualization applied to FCA », *in*: *Supplementary Proceedings of ICFCA 2019 Conference and Workshops, Frankfurt, Germany, June 25-28, 2019*, 2019, pp. 39–44, URL: http://ceur-ws.org/Vol-2378/shortAT2.pdf.

[19] Lucas Bourneuf and Jacques Nicolas, « Concept Lattices as a Search Space for Graph Compression », *in*: *Formal Concept Analysis: 15th International Conference, ICFCA 2019, Frankfurt, Germany, June 25-28, 2019, Proceedings*, Springer International Publishing, 2019.

[20] Lucas Bourneuf and Jacques Nicolas, « FCA in a Logical Programming Setting for Visualization-Oriented Graph Compression », *in*: *Formal Concept Analysis: 14th International Conference, ICFCA 2017, Rennes, France, June 13-16, 2017, Proceedings*, Springer International Publishing, 2017, pp. 89–105, ISBN: 978-3-319-59271-8, DOI: 10.1007/978-3-319-59271-8_6, URL: https://doi.org/10.1007/978-3-319-59271-8_6.

[21] Assia Brighen et al., « Listing all maximal cliques in large graphs on vertex-centric model », *in*: *The Journal of Supercomputing* (2019), pp. 1–29.

[22] Coen Bron and Joep Kerbosch, « Algorithm 457: finding all cliques of an undirected graph », *in*: *Communications of the ACM* 16.*9* (1973), pp. 575–577.

[23] Dongbo Bu et al., « Topological structure analysis of the protein–protein interaction network in budding yeast », *in*: *Nucleic Acids Research* 31.*9*

(2003), pp. 2443–2450, ISSN: 0305-1048, URL: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC154226/ (visited on 06/03/2015).

[24] F. Calimeri et al., « Angry-HEX: An Artificial Player for Angry Birds Based on Declarative Knowledge Bases », *in*: *IEEE Transactions on Computational Intelligence and AI in Games* 8.*2* (June 2016), pp. 128–139, ISSN: 1943-068X, DOI: 10.1109/TCIAIG.2015.2509600.

[25] C Carpineto and G Romano, *Concept Data Analysis: Theory and Applications*, John Wiley & Sons, 2004, DOI: 10.1002/0470011297.ch3.

[26] Emilie Chautard et al., « MatrixDB, a database focused on extracellular protein–protein and protein–carbohydrate interactions », en, *in*: *Bioinformatics* 25.*5* (Mar. 2009), pp. 690–691, ISSN: 1460-2059, 1367-4803, DOI: 10.1093/bioinformatics/btp025, URL: https://academic.oup.com/bioinformatics/article-lookup/doi/10.1093/bioinformatics/btp025 (visited on 02/20/2018).

[27] Giampiero Chiaselotti, Davide Ciucci, and Tommaso Gentile, « Simple Undirected Graphs as Formal Contexts », *in*: *International Conference on Formal Concept Analysis*, vol. 9113, June 2015, pp. 187–302, DOI: 10.1007/978-3-319-19545-2_18.

[28] Paola Chiodelli et al., « Heparin/Heparan sulfate proteoglycans glycomic interactome in angiogenesis: biological implications and therapeutical use », *in*: *Molecules* 20.*4* (2015), pp. 6342–6388.

[29] F.R.K. Chung, « On the coverings of graphs », *in*: *Discrete Mathematics* 30.*2* (1980), pp. 89–93, ISSN: 0012-365X, DOI: http://dx.doi.org/10.1016/0012-365X(80)90109-0, URL: http://www.sciencedirect.com/science/article/pii/0012365X80901090.

[30] Philipp Cimiano et al., « Conceptual Knowledge Processing with Formal Concept Analysis and Ontologies », *in*: *Concept Lattices*, ed. by Peter Eklund, Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 189–207, ISBN: 978-3-540-24651-0.

[31] International Aphid Genomics Consortium et al., « Genome sequence of the pea aphid Acyrthosiphon pisum », *in*: *PLoS biology* 8.*2* (2010), e1000313.

[32] Simone Daminelli et al., « Drug repositioning through incomplete bi-cliques in an integrated drug–target–disease network », en, *in*: *Integrative Biology* 4.*7* (June 2012), pp. 778–788, ISSN: 1757-9708, DOI: 10.1039/C2IB00154C, URL: http://pubs.rsc.org/en/content/articlelanding/2012/ib/c2ib00154c (visited on 06/03/2015).

[33] Xavier Dolques, Florence Le Ber, and Marianne Huchard, « AOC-posets: a scalable alternative to Concept Lattices for Relational Concept Analysis », *in*: *CLA: Concept Lattices and their Applications*, La Rochelle, France, Oct. 2013, pp. 129–140, URL: https://hal.archives-ouvertes.fr/hal-00916850.

[34] Dheeru Dua and Casey Graff, *UCI Machine Learning Repository*, 2017, URL: http://archive.ics.uci.edu/ml.

[35] Didier Dubois and Henri Prade, « Formal Concept Analysis from the Standpoint of Possibility Theory », *in*: *Formal Concept Analysis*, ed. by Jaume Baixeries, Christian Sacarea, and Manuel Ojeda-Aciego, Cham: Springer International Publishing, 2015, pp. 21–38, ISBN: 978-3-319-19545-2.

[36] Tim Dwyer et al., « Edge Compression Techniques for Visualization of Dense Directed Graphs », *in*: *IEEE Transactions on Visualization and Computer Graphics* 19.*12* (Dec. 2013), pp. 2596–2605, ISSN: 1077-2626, DOI: 10.1109/TVCG.2013.151, URL: http://dx.doi.org/10.1109/TVCG.2013.151.

[37] Tim Dwyer et al., « Improved Optimal and Approximate Power Graph Compression for Clearer Visualisation of Dense Graphs », *in*: *CoRR* abs/1311.6996 (2013), URL: http://arxiv.org/abs/1311.6996.

[38] Thomas Eiter et al., « The DLVHEX System », *in*: *KI - Künstliche Intelligenz* (May 2018), ISSN: 1610-1987, DOI: 10.1007/s13218-018-0535-y, URL: https://doi.org/10.1007/s13218-018-0535-y.

[39] David Eppstein, « Arboricity and bipartite subgraph listing algorithms », *in*: *Information Processing Letters* 51.*4* (1994), pp. 207–211, ISSN: 0020-0190, DOI: http://dx.doi.org/10.1016/0020-0190(94)90121-

X, URL: http://www.sciencedirect.com/science/article/pii/002001909490121X.

[40] Esra Erdem, Michael Gelfond, and Nicola Leone, « Applications of answer set programming », *in*: *AI Magazine* 37.*3* (2016), pp. 53–68.

[41] Giulia Furini and Elisabetta AM Verderio, « Spotlight on the transglutaminase 2-heparan sulfate interaction », *in*: *Medical Sciences* 7.*1* (2019), p. 5.

[42] Julien Gagneur et al., « Modular decomposition of protein-protein interaction networks », en, *in*: *Genome Biology* 5.*8* (July 2004), R57, ISSN: 1465-6906, DOI: 10.1186/gb-2004-5-8-r57, URL: http://genomebiology.com/2004/5/8/R57/abstract (visited on 06/03/2015).

[43] Emden R. Gansner and Stephen C. North, « An Open Graph Visualization System and Its Applications to Software Engineering », *in*: *Softw. Pract. Exper.* 30.*11* (Sept. 2000), pp. 1203–1233, ISSN: 0038-0644, DOI: 10.1002/1097-024X(200009)30:11<1203::AID-SPE338>3.3.CO;2-E, URL: http://dx.doi.org/10.1002/1097-024X(200009)30:11%3C1203::AID-SPE338%3E3.3.CO;2-E.

[44] Bernhard Ganter and Sergei O Kuznetsov, « Pattern structures and their projections », *in*: *International Conference on Conceptual Structures*, Springer, 2001, pp. 129–142.

[45] Bernhard Ganter and Rudolf Wille, *Formal concept analysis: mathematical foundations*, Springer Science & Business Media, 2012.

[46] Bruno Gaume, Emmanuel Navarro, and Henri Prade, « A Parallel between Extended Formal Concept Analysis and Bipartite Graphs Analysis », *in*: *Computational Intelligence for Knowledge-Based Systems Design*, ed. by Eyke Hüllermeier, Rudolf Kruse, and Frank Hoffmann, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 270–280, ISBN: 978-3-642-14049-5.

[47] M. Gebser et al., *Potassco User Guide*, 2015.

[48] M. Gebser et al., « Potassco: The Potsdam Answer Set Solving Collection », *in*: *AI Communications* 24.*2* (2011), pp. 107–124.

[49] MARTIN GEBSER et al., « Multi-shot ASP solving with clingo », *in*: *Theory and Practice of Logic Programming* 19.*1* (2019), pp. 27–82, DOI: 10.1017/S1471068418000054.

[50] Martin Gebser et al., « Routing driverless transport vehicles in car assembly with answer set programming », *in*: *Theory and Practice of Logic Programming* 18.*3-4* (2018), pp. 520–534.

[51] Martin Gebser et al., « Theory Solving Made Easy with Clingo 5 », *in*: *Technical Communications of the 32nd International Conference on Logic Programming (ICLP 2016)*, ed. by Manuel Carro et al., vol. 52, OpenAccess Series in Informatics (OASIcs), Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016, 2:1–2:15, ISBN: 978-3-95977-007-1, DOI: 10.4230/OASIcs.ICLP.2016.2, URL: http://drops.dagstuhl.de/opus/volltexte/2016/6733.

[52] M. Gelfond and V. Lifschitz, « Logic Programs with Classical Negation », *in*: *Proc. of the 7th International Conf. on Logic Programming (ICLP)*, 1990, pp. 579–97.

[53] Alain Gély, Lhouari Nourine, and Bachir Sadi, « Enumeration aspects of maximal cliques and bicliques », *in*: *Discrete Applied Mathematics* 157.*7* (2009), pp. 1447–1459, ISSN: 0166-218X, DOI: https://doi.org/10.1016/j.dam.2008.10.010, URL: http://www.sciencedirect.com/science/article/pii/S0166218X08004563.

[54] Helen Gibson, Joe Faith, and Paul Vickers, « A survey of two-dimensional graph layout techniques for information visualisation », *in*: *Information Visualization* 12.*3-4* (2013), pp. 324–357, DOI: 10.1177/1473871612455749, eprint: https://doi.org/10.1177/1473871612455749, URL: https://doi.org/10.1177/1473871612455749.

[55] Robert Godin and Hafedh Mili, « Building and Maintaining Analysis-level Class Hierarchies Using Galois Lattices », *in*: *SIGPLAN Not.* 28.*10* (Oct. 1993), pp. 394–410, ISSN: 0362-1340, DOI: 10.1145/167962.165931, URL: http://doi.acm.org/10.1145/167962.165931.

[56] MA Grant and R Kallur, « Structural basis for the functions of endogenous angiogenesis inhibitors », *in*: *Cold Spring Harbor symposia on quantitative biology*, vol. 70, Cold Spring Harbor Laboratory Press, 2005, pp. 399–417.

[57] Michel Habib and Christophe Paul, « A survey of the algorithmic aspects of modular decomposition », *in*: *Computer Science Review* 4.*1* (2010), pp. 41–59.

[58] Pascal Hitzler and Markus Krötzsch, « Querying Formal Contexts with Answer Set Programs », *in*: *Proc. of the 14th Int. Conf. on Conceptual Structures: Inspiration and Application*, ICCS'06, Aalborg, Denmark: Springer-Verlag, 2006, pp. 260–273, DOI: 10.1007/11787181_19, URL: http://dx.doi.org/10.1007/11787181_19.

[59] Sheridan Houghten et al., « Compression of Biological Networks using a Genetic Algorithm with Localized Merge », *in*: *2019 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*, IEEE, 2019, pp. 1–8.

[60] Bernard Jocelyn and Hamida Seba, « Solving the Maximal Clique Problem on Compressed Graphs », *in*: *24th International Symposium on Methodologies on Intelligent (ISMIS 2018)*, Foundations of Intelligent Systems, LImassol, Cyprus, Oct. 2018, pp. 45–55, DOI: 10.1007/978-3-030-01851-1_5, URL: https://hal.archives-ouvertes.fr/hal-01886654.

[61] S. Jukna and A.S. Kulikov, « On covering graphs by complete bipartite subgraphs », *in*: *Discrete Mathematics* 309.*10* (2009), pp. 3399–3403, ISSN: 0012-365X, DOI: http://dx.doi.org/10.1016/j.disc.2008.09.036, URL: http://www.sciencedirect.com/science/article/pii/S0012365X08005566.

[62] Isaac Karth and Adam M. Smith, « WaveFunctionCollapse is Constraint Solving in the Wild », *in*: *Proceedings of the 12th International Conference on the Foundations of Digital Games*, FDG '17, Hyannis, Massachusetts: ACM, 2017, 68:1–68:10, ISBN: 978-1-4503-5319-9, DOI: 10.1145/3102071.3110566, URL: http://doi.acm.org/10.1145/3102071.3110566.

[63] Mehdi Kaytoue et al., « Mining gene expression data with pattern structures in formal concept analysis », *in*: *Information Sciences* 181.*10* (2011), Special Issue on Information Engineering Applications Based on Lattices, pp. 1989–2001, ISSN: 0020-0255, DOI: https://doi.org/10.1016/j.ins.2010.07.007, URL: http://www.sciencedirect.com/science/article/pii/S0020025510003257.

[64] Levente Lorand Kis, Christian Sacarea, and Diana Troanca, « FCA Tools Bundle-a Tool that Enables Dyadic and Triadic Conceptual Navigation », *in*: *Proc. of FCA4AI* (2015).

[65] Jens Kötters, Heinz Schmidt, and David McG. Squire, « Context Graphs — Representing Formal Concepts by Connected Subgraphs », *in*: *Formal Concept Analysis*, ed. by Sébastien Ferré and Sebastian Rudolph, Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 178–193, ISBN: 978-3-642-01815-2.

[66] Danai Koutra et al., « Summarizing and understanding large graphs », *in*: *Statistical Analysis and Data Mining: The ASA Data Science Journal* 8.*3* (Mar. 2015), pp. 183–202.

[67] Maren Krone and Gregor Snelting, « On the Inference of Configuration Structures from Source Code », *in*: *Proceedings of the 16th International Conference on Software Engineering*, ICSE '94, Sorrento, Italy: IEEE Computer Society Press, 1994, pp. 49–57, ISBN: 0-8186-5855-X, URL: http://dl.acm.org/citation.cfm?id=257734.257742.

[68] Boumedjout Lahcen and Leonard Kwuida, « Lattice miner: a tool for concept lattice construction and exploration », *in*: *Suplementary Proceeding of International Conference on Formal concept analysis (ICFCA'10)*, 2010.

[69] G. Launay et al., « MatrixDB, the extracellular matrix interaction database: updated content, a new navigator and expanded functionalities », *in*: *Nucleic Acids Research* 43.*D1* (2015), pp. D321–D327, DOI: 10.1093/nar/gku1091, eprint: http://nar.oxfordjournals.org/content/43/D1/D321.full.pdf+html, URL: http://nar.oxfordjournals.org/content/43/D1/D321.abstract.

[70] Gael Le Trionnaire, Valentin Wucher, and Denis Tagu, « Genome expression control during the photoperiodic response of aphids », *in*: *Physiological Entomology* 38.*2* (2013), pp. 117–125, DOI: 10.1111/phen.12021, eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/phen.12021, URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/phen.12021.

[71] Fabrice Legeai et al., « Bioinformatic prediction, deep sequencing of microRNAs and expression analysis during phenotypic plasticity in the pea aphid, Acyrthosiphon pisum », *in*: *BMC Genomics* (May 2010), DOI: 10.1186/1471-2164-11-281.

[72] Fritz Lehmann and Rudolf Wille, « A triadic approach to formal concept analysis », *in*: *Conceptual structures: applications, implementation and theory*, ed. by G. Ellis et al., Lecture Notes in Artificial Intelligence 954, Berlin–Heidelberg–New York: Springer–Verlag, 1995, pp. 32–43.

[73] J. Li et al., « Maximal Biclique Subgraphs and Closed Pattern Pairs of the Adjacency Matrix: A One-to-One Correspondence and Mining Algorithms », *in*: *IEEE Transactions on Knowledge and Data Engineering* 19.*12* (Dec. 2007), pp. 1625–1637, ISSN: 1041-4347, DOI: 10.1109/TKDE.2007.190660.

[74] Jinyan Li et al., « Maximal quasi-bicliques with balanced noise tolerance: Concepts and co-clustering applications », *in*: *Proceedings of the 2008 SIAM International Conference on Data Mining*, SIAM, 2008, pp. 72–83.

[75] Yiyuan Li et al., « Gene Family Evolution in the Pea Aphid Based on Chromosome-Level Genome Assembly », *in*: *Molecular biology and evolution* 36.*10* (2019), pp. 2143–2156.

[76] Vladimir Lifschitz, *Answer Set Programming*, Jan. 2019, ISBN: 978-3-030-24657-0, DOI: 10.1007/978-3-030-24658-7.

[77] Guohua Liu, Tomi Janhunen, and Ilkka Niemelä, « Answer Set Programming via Mixed Integer Programming. », *in*: *KR*, 2012, pp. 32–42.

[78] Saket Navlakha, Michael C Schatz, and Carl Kingsford, « Revealing biological modules via graph summarization », *in*: *Journal of Computational Biology* 16.*2* (2009), pp. 253–264.

[79] Mark J. Nelson and Adam M. Smith, « ASP with Applications to Mazes and Levels », *in*: *Procedural Content Generation in Games*, Cham: Springer International Publishing, 2016, pp. 143–157, ISBN: 978-3-319-42716-4, DOI: 10.1007/978-3-319-42716-4_8, URL: https://doi.org/10.1007/978-3-319-42716-4_8.

[80] Linh-Chi Nguyen, « Compression of large interaction graphs for the identification of protein modules in the extracellular matrix », Internship report, Université de Rennes 1, Inria Rennes Bretagne, 2018, URL: http://example.com/2013/example_semester_report_12.pdf.

[81] Van Nguyen et al., « Generalized Target Assignment and Path Finding Using Answer Set Programming », *in*: *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, AAAI Press, 2017, pp. 1216–1223.

[82] H. Ogata et al., « A heuristic graph comparison algorithm and its application to detect functionally related enzyme clusters », eng, *in*: *Nucleic Acids Research* 28.*20* (Oct. 2000), pp. 4021–4028, ISSN: 1362-4962.

[83] Sandra Orchard et al., « The MIntAct project–IntAct as a common curation platform for 11 molecular interaction databases », eng, *in*: *Nucleic Acids Research* 42.*Database issue* (Jan. 2014), pp. D358–363, ISSN: 1362-4962, DOI: 10.1093/nar/gkt1115.

[84] Rainer Osswald and Wiebke Petersen, « A Logical Approach to Data-Driven Classification », *in*: *KI 2003: Advances in Artificial Intelligence: 26th Annual German Conference on AI, KI 2003, Hamburg, Germany, September 15-18, 2003. Proceedings*, ed. by Andreas Günter, Rudolf Kruse, and Bernd Neumann, Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 267–281, ISBN: 978-3-540-39451-8, DOI: 10.1007/978-3-540-39451-8_20, URL: https://doi.org/10.1007/978-3-540-39451-8_20.

[85] Hideaki Otsuki and Tomio Hirata, « The biclique cover problem and the modified Galois lattice », *in*: *IEICE TRANSACTIONS on Information and Systems* 98.*3* (2015), pp. 497–502.

[86] C. PAN et al., « BiModule: biclique modularity strategy for identifying transcription factor and microRNA co-regulatory modules », *in*: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* (2019), pp. 1–1, DOI: 10.1109/TCBB.2019.2896155.

[87] Charis Papadopoulos and Constantinos Voglis, « Drawing Graphs Using Modular Decomposition », *in*: *Graph Drawing*, ed. by Patrick Healy and Nikola S. Nikolov, Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 343–354, ISBN: 978-3-540-31667-1.

[88] René Peeters, « The maximum edge biclique problem is NP-complete », *in*: *Discrete Applied Mathematics* 131.*3* (2003), pp. 651–654, ISSN: 0166-218X, DOI: https://doi.org/10.1016/S0166-218X(03)00333-0, URL: http://www.sciencedirect.com/science/article/pii/S0166218X03003330.

[89] Jonas Poelmans et al., « Formal concept analysis in knowledge processing: A survey on applications », *in*: *Expert Systems with Applications* 40.*16* (2013), pp. 6538–6560, ISSN: 0957-4174, DOI: https://doi.org/10.1016/j.eswa.2013.05.009, URL: http://www.sciencedirect.com/science/article/pii/S0957417413002959.

[90] Matthias Reimann et al., « Visualization and Interactive Analysis for Complex Networks by means of Lossless Network Compression », *in*: *Computational Network Theory: Theoretical Foundations and Applications, M. Dehmer, F. Emmert-Streib, and S. Pickl, Editors* (2015).

[91] Andrei Rimsa, Mark AJ Song, and Luis E Zárate, « Scgaz-A synthetic formal context generator with density control for test and evaluation of FCA algorithms », *in*: *2013 IEEE International Conference on Systems, Man, and Cybernetics*, IEEE, 2013, pp. 3464–3470.

[92] Mohamed Rouane-Hacene et al., « Relational concept analysis: mining concept lattices from multi-relational data », *in*: *Annals of Mathematics and Artificial Intelligence* 67.*1* (Jan. 2013), pp. 81–108, ISSN: 1573-7470, DOI:

10.1007/s10472-012-9329-3, URL: https://doi.org/10.1007/s10472-012-9329-3.

[93] Loic Royer et al., « Network Compression as a Quality Measure for Protein Interaction Networks », *in*: *PLoS ONE* 7.6 (2012), e35729, DOI: 10.1371/journal.pone.0035729, URL: http://dx.doi.org/10.1371/journal.pone.0035729 (visited on 06/08/2015).

[94] Loïc Royer et al., « Unraveling Protein Networks with Power Graph Analysis », *in*: *PLoS Comput Biol* 4.7 (2008), e1000108, DOI: 10.1371/journal.pcbi.1000108, URL: http://dx.plos.org/10.1371/journal.pcbi.1000108 (visited on 05/29/2015).

[95] Sebastian Rudolph, Christian Săcărea, and Diana Troancă, « Membership Constraints in Formal Concept Analysis », *in*: *Proc.of the 24th Int. Conf. on Artificial Intelligence*, IJCAI'15, Buenos Aires, Argentina: AAAI Press, 2015, pp. 3186–3192, ISBN: 978-1-57735-738-4, URL: http://dl.acm.org/citation.cfm?id=2832581.2832693.

[96] Romain Salza et al., « Extended interaction network of procollagen C-proteinase enhancer-1 in the extracellular matrix », *in*: *Biochemical Journal* 457.1 (Dec. 2013), pp. 137–149, ISSN: 0264-6021, DOI: 10.1042/BJ20130295, eprint: https://portlandpress.com/biochemj/article-pdf/457/1/137/678750/bj4570137.pdf, URL: https://doi.org/10.1042/BJ20130295.

[97] Arunpreet Sandhu, Zeyuan Chen, and Joshua McCoy, « Enhancing Wave Function Collapse with Design-level Constraints », *in*: *Proceedings of the 14th International Conference on the Foundations of Digital Games*, FDG '19, San Luis Obispo, California: ACM, 2019, 17:1–17:9, ISBN: 978-1-4503-7217-6, DOI: 10.1145/3337722.3337752, URL: http://doi.acm.org/10.1145/3337722.3337752.

[98] S. Schieweck, G. Kern-Isberner, and M. Ten Hompel, « Using answer set programming in an order-picking system with cellular transport vehicles », *in*: *2016 IEEE International Conference on Industrial Engineering and En-*

gineering Management (IEEM), Dec. 2016, pp. 1600–1604, DOI: 10.1109/IEEM.2016.7798147.

[99] Paul Shannon et al., « Cytoscape: A Software Environment for Integrated Models of Biomolecular Interaction Networks », *in*: *Genome Research* 13.*11* (2003), pp. 2498–2504, DOI: 10.1101/gr.1239303, eprint: http://genome.cshlp.org/content/13/11/2498.full.pdf+html, URL: http://genome.cshlp.org/content/13/11/2498.abstract.

[100] Satyaki Sikdar, Justus Hibshman, and Tim Weninger, « Modeling Graphs with Vertex Replacement Grammars », *in*: *ArXiv* abs/1908.03837 (2019).

[101] Prem Kumar Singh, Cherukuri Aswani Kumar, and Abdullah Gani, « A comprehensive survey on formal concept analysis, its research trends and applications », *in*: *International Journal of Applied Mathematics and Computer Science* 26.*2* (2016), pp. 495–516.

[102] A. M. Smith and M. Mateas, « Answer Set Programming for Procedural Content Generation: A Design Space Approach », *in*: *IEEE Transactions on Computational Intelligence and AI in Games* 3.*3* (Sept. 2011), pp. 187–200, DOI: 10.1109/TCIAIG.2011.2158545.

[103] Adam M. Smith, Student Member, and Michael Mateas, « Answer set programming for procedural content generation: A design space approach », *in*: *In Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*, 2010.

[104] Gerd Stumme et al., « Conceptual Clustering with Iceberg Concept Lattices », *in*: *In: Proc. of GI-Fachgruppentreffen Maschinelles Lernen'01, Universität Dortmund*, 2001.

[105] Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi, « The worst-case time complexity for generating all maximal cliques and computational experiments », *in*: *Theoretical Computer Science* 363.*1* (2006), Computing and Combinatorics, pp. 28–42, ISSN: 0304-3975, DOI: https://doi.org/10.1016/j.tcs.2006.06.015, URL: http://www.sciencedirect.com/science/article/pii/S0304397506003586.

[106] George Tsatsaronis et al., « Efficient Community Detection Using Power Graph Analysis », *in*: *Proc. of the 9th Workshop on Large-scale and Distributed Informational Retrieval*, LSDS-IR '11, Glasgow, Scotland, UK: ACM, 2011, pp. 21–26, ISBN: 978-1-4503-0959-2, DOI: 10.1145/2064730.2064738, URL: http://doi.acm.org/10.1145/2064730.2064738.

[107] Mario Vallon et al., « Developmental and pathological angiogenesis in the central nervous system », *in*: *Cellular and Molecular Life Sciences* 71.*18* (2014), pp. 3489–3506.

[108] Ling Wei and Min-Qian Liu, « Attribute Characteristics of Object (Property) Oriented Concept Lattices », *in*: *Rough Sets and Current Trends in Computing: 8th International Conference, RSCTC 2012, Chengdu, China, August 17-20, 2012.Proceedings*, ed. by JingTao Yao et al., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 341–348, ISBN: 978-3-642-32115-3, DOI: 10.1007/978-3-642-32115-3_40, URL: https://doi.org/10.1007/978-3-642-32115-3_40.

[109] Valentin Wucher, « Modeling of a gene network between mRNAs and miRNAs to predict gene functions involved in phenotypic plasticity in the pea aphid », Thesis, Université Rennes 1, Nov. 2014, URL: https://tel.archives-ouvertes.fr/tel-01135870.

[110] Valentin Wucher, Denis Tagu, and Jacques Nicolas, « Edge Selection in a Noisy Graph by Concept Analysis: Application to a Genomic Network », *in*: *Data Science, Learning by Latent Structures, and Knowledge Discovery*, ed. by Berthold Lausen, Sabine Krolak-Schwerdt, and Matthias Böhmer, Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 353–364, ISBN: 978-3-662-44983-7.

[111] YY Yao, « Concept lattices in rough set theory », *in*: *Fuzzy Information, 2004. Processing NAFIPS'04. IEEE Annual Meeting of the*, vol. 2, IEEE, 2004, pp. 796–801.

[112] Yun Zhang et al., « Genome-scale computational approaches to memory-intensive applications in systems biology », *in*: *SC'05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, IEEE, 2005.

UNIVERSITE BRETAGNE LOIRE MATHSTIC

UNIVERSITÉ DE RENNES 1

Titre : Un espace de recherche des motifs de graphes pour la compression de graphe : de Powergraph aux concepts triplets

Mot clés : Compression de Graphes, Visualisation de Graphes, Cliques et Bicliques, Analyse de Concepts Formels, Programmation par Ensembles Réponses

Résumé : L'Analyse Power Graph est une technique de compression sans perte de graphe visant à réduire la complexité visuelle d'un graphe. Le processus consiste à détecter des motifs, les cliques et les bicliques, qui permettent d'établir des groupes de nœuds organisés hiérarchiquement, des groupes d'arcs, et finalement un graph réduit à ces groupes. Cette thèse propose tout d'abord la formalisation de l'espace de recherche de l'Analyse Power Graph, en utilisant l'Analyse de Concepts Formels comme base théorique pour exprimer le processus de compression. Le traitement indépendant de deux motifs présente des difficultés et nous proposons une notion unificatrice, les concepts triplets, qui conduiront à un motif unique plus général pour la compression. L'Analyse Power Graph et la nouvelle approche ont été implémentés dans un formalisme logique de Programmation par Ensembles Réponses (ASP), et nous présentons quelques applications en bioinformatique pour les deux approches. La thèse se clot sur la présentation d'un environnement de visualisation et de spécification de haut-niveau en théorie des graphes.

Title: A search space of graph motifs for graph compression: From Powergraphs to triplet concepts

Keywords: Graph Compression, Graph Visualization, Cliques and Bicliques, Formal Concept Analysis, Answer Set Programming

Abstract: Power Graph Analysis is a lossless graph compression method aiming at reducing the visual complexity of a graph. The process is to detect motifs, cliques and bicliques, which enables the hierarchical clustering of nodes, the grouping of edges, and ultimately a graph reduced to these groups. This thesis exposes first the formalization of the Power Graph Analysis search space, using Formal Concept Analysis as a theoretical ground to express the compression process. Because the independent treatment of two motifs presents some caveats, we propose a unification framework, triplet concepts, which encode a more general motif for compression. Both Power Graph Analysis and the new approach have been implemented in Answer Set Programming (ASP), a logical formalism, and we present some applications in bioinformatics of these two approaches. This thesis ends on the presentation of an high-level specification and visualization environment for graph theory.