EDIIS

Université Claude Bernard Lyon I

École doctorale Informatique et Information pour la Société

Mesurer la similarité de graphes

THÈSE

présentée et soutenue publiquement le 24 Novembre 2006

pour l'obtention du

Doctorat de l'Université Claude Bernard Lyon I

(spécialité informatique)

par

Sébastien Sorlin

Composition du jury

Rapporteurs :	François Fages	Directeur de recherche, INRIA Rocquencourt
	Narendra Jussien	Enseignant chercheur HDR, École des Mines de Nantes
Examinateurs :	Jean-François Puget	Vice-président Optimisation R&D, ILOG S.A.
	Bernard Péroche	Professeur, Université Claude Bernard Lyon I
Directeurs de recherche :	Christine Solnon	Maître de conférences HDR, Université Claude Bernard Lyon
	Mohand-Saïd Hacid	Professeur, Université Claude Bernard Lyon I

Laboratoire d'InfoRmatique en Images et Systèmes d'information — UMR 5205

Mis en page avec la classe thloria.

Remerciements

Je n'étais pas encore en âge de faire de la recherche (bac+2) quand j'ai eu la chance de pouvoir suivre le cours d'intelligence artificielle de Christine Solnon. Ce cours a été pour moi une révélation : je découvrais qu'il était très long de trouver la façon la plus rapide de résoudre un jeu de taquin. Ca me semblait à la fois étrange et totalement absurde : je décidais de poursuivre mes études. L'année suivante, Christine m'a fait découvrir la recherche locale. L'année d'après, je faisais mes premiers pas dans un laboratoire de recherche. L'année encore d'après, Christine encadrait mon stage de DEA. Entre temps, j'avais découvert ce qu'était un "graphe" et que jouer avec pouvait être aussi drôle que jouer au taquin. J'étais fin prêt pour des études doctorales.

La recherche est une passion intellectuellement transmissible. Christine a été pour moi le vecteur de contamination et c'est de cela que je lui suis le plus reconnaissant. Sa passion et son enthousisame m'ont donné l'envie de continuer mes études et m'ont permis de découvrir le monde de la recherche.

Je remercie également Christine pour la qualité de son encadrement de mon année de maîtrise jusqu'à la fin de ma thèse : la justesse de ses suggestions, la rigueur de ses relectures (et le temps qu'elle y a passé) m'ont permis d'apprendre ce qu'était que faire de la recherche et comment la communiquer. Finalement, je tiens à la remercier pour tous les bons moments qu'on a passé ensemble, au labo ou ailleurs, et qui ont fait de mes années de thèse autre chose qu'un simple travail.

Je tiens également à remercier Mohand-Saïd Hacid, mon co-directeur de thèse ainsi que (ensembles non disjoints, les concernés se reconnaîtront), les membres de ma famille (qui n'ont pas toujours bien compris ce que je faisais mais qui ont tout de même réussi à me soutenir et à me motiver), mes amis (qui me permettent de décompresser entre deux travaux de recherche), mes co-bureaux (avec qui c'est un plaisir de venir au labo), mes autres collègues (sans qui les pauses cafés ne seraient pas d'aussi bons moments et avec qui j'ai eu le plaisir d'enseigner) et tous les gens que j'ai eu l'occasion de rencontrer en conférence et autre école d'été.

Je rémercie également François Fages et Narendra Jussien d'avoir bien voulu être les rapporteurs de ce mémoire et Jean-François Puget et Bernard Péroche d'avoir accepté être examinateurs de ma thèse. Je remercie enfin Jean-Michel Jolion pour l'aide qu'il m'a apporté durant ma thèse.

Table des matières

Introduction

1 Contexte

Les capacités de stockage de documents informatiques augmentant, le nombre de documents électroniques manipulés croît de façon exponentielle dans de nombreux domaines d'application. Il est nécessaire de disposer d'outils informatiques permettant de retrouver ces documents ou de classifier un ensemble de documents. Ces tâches nécessitent de pouvoir mesurer la similarité ou la distance entre les documents manipulés, c.-à-d. de pouvoir quantifier les points communs et les différences entre deux documents afin de retrouver les documents les plus similaires à un autre. Mesurer la distance entre deux documents permet également de calculer la moyenne et l'écart type d'un ensemble de documents, deux opérations indispensables pour classifier les documents, c.-à-d. construire des classes de documents jugés similaires au sens de l'application considérée. Il est également parfois nécessaire de pouvoir non seulement quantifier les différences entre deux documents mais aussi de les qualifier, c.-à-d. identifier en quoi deux documents sont similaires et en quoi ils diffèrent. Ces informations permettent d'expliquer à l'utilisateur les choix du système de recherche.

Mesurer la distance entre deux documents nécessite de modéliser de façon formelle les documents à comparer et de définir une distance entre les modèles. Une première étape pour la conception d'un système de recherche d'information consiste donc à choisir une modélisation formelle des documents, c.-à-d. une modélisation pouvant être exploitée de façon automatique. Le choix de cette modélisation influe énormément sur les performances du système. En effet, cette modélisation doit contenir le maximum d'informations pertinentes sur les documents (c.-à-d. les informations sémantiquement significatives lorsque les documents devront être comparés entre eux) tout en ne gardant que les informations pouvant être traitées efficacement de façon automatique. Ces informations peuvent être extraites automatiquement des documents manipulés ou provenir d'annotations manuelles. De façon générale, plus la modélisation des documents sera sophistiquée et plus la comparaison des documents sera pertinente mais difficile.

Les vecteurs numériques, les chaînes, les arbres et les graphes sont des structures de données couramment utilisées pour modéliser les documents.

Les vecteurs numériques. La modélisation la plus souvent utilisée en recherche d'informations est la modélisation en vecteurs numériques. Un ensemble d'attributs ou de caractéristiques jugés pertinents par rapport aux besoins de l'application est retenu. La valeur de chacune de ces caractéristiques est estimée pour chaque document de la base et un document est alors représenté par un vecteur représentant toutes les valeurs pour ce document. Les tâches d'indexation et de recherche de documents se font alors par l'intermédiaire des vecteurs qui les représentent.

Les vecteurs numériques sont très souvent utilisés en recherche d'informations textuelles où le vecteur représente l'histogramme des fréquences d'apparition des mots dans un document textuel (voir par exemple la mesure $tf * idf$ de Jones [Jon72]). En reconnaissance d'images, le vecteur peut par exemple représenter un histogramme des couleurs de l'image (voir par exemple Swain

et Ballard [SB91]).

L'avantage de la représentation en vecteur est que les distances entre vecteurs numériques sont facilement et rapidement calculables et les notions de moyenne et d'écart-type d'un ensemble de vecteurs numériques sont clairement définies. Les tâches de classification et de recherche de vecteurs similaires sont donc aisées. En contrepartie de cette simplicité et de cette rapidité, les vecteurs ont un pouvoir d'expression assez pauvre. En particulier, ils ne permettent pas d'exprimer la structure d'un document, c.-à-d. le fait qu'un document puisse se décomposer en plusieurs éléments et qu'il puisse exister des relations entre ces éléments qu'il est intéressant de prendre en compte lors d'une comparaison des documents.

Les chaînes. Les chaînes permettent d'établir un ordre total entre des éléments et par conséquent de représenter une séquentialité entre les composants d'un document. Elles sont donc utilisées en recherche d'information lorsqu'il devient pertinent d'ordonner des éléments. C'est par exemple le cas pour la recherche d'information textuelle où un texte n'est alors plus considéré comme un ensemble de mots ou de termes (comme dans le cas des vecteurs) mais comme une liste ordonnée de mots. En comparaison d'images, les chaînes ont par exemple été utilisées par Ros *et al.* [RLJS05]. Étant donnée une image, les auteurs extraient automatiquement un ensemble de pixels "perceptuellement" significatifs (c.-à-d. des points de l'image sur lesquels un observateur humain aura tendance à se concentrer). Ces points d'intérêts sont ordonnés par ordre de saillance décroissante, cet ordre tentant de reproduire le parcours visuel d'un observateur humain de cette image. Chaque image est alors représentée par une chaîne de blocs de pixels (en général longue de quelques centaines de blocs).

La distance entre deux chaînes est souvent définie par la distance d'édition de chaînes de Levenshtein [Lev65]. Cette distance est facilement et rapidement calculable et la moyenne et l'écart-type d'un ensemble de chaînes, bien que longs à calculer, sont bien définies [Jol04].

Les arbres. Les arbres sont fréquemment utilisés pour représenter des documents. Ils permettent de représenter chaque document comme une liste (ordonnée ou non) de sous-documents eux-mêmes décomposables en sous-documents plus petits. La granularité de la représentation dépend de la profondeur de l'arbre. Un document (c.-à-d. un nœud de l'arbre) est alors jugé d'autant plus similaire à un autre document que leurs décompositions en sous-documents sont similaires.

Les arbres sont très utilisés en recherche d'information sur le Web (par exemple [dBSL04]) : en effet, les documents sont de plus en plus souvent décrits en XML et un document XML est un arbre sur lequel une sémantique est associée à chaque nœud. L'essentiel de l'information utilisable pour comparer deux documents réside alors dans la structure de l'arbre XML qui décrit les documents. La distance d'édition d'arbres [Sel77] est facilement calculable.

Les graphes. Les arbres ne permettent pas de représenter toutes les relations possibles entre les composants. Bien souvent, la sémantique associée à un arc (c.-à-d. à une relation fils-père) est limitée à une relation de typage (la relation "est de type") où d'appartenance (la relation "appartient à"). Les graphes orientés sont une généralisation des arbres. Il permettent une modélisation très riche des documents et de leurs structures. Un document est représenté comme un ensemble de composants (les sommets du graphe) et un ensemble de relations binaires entre ces composants (les arcs du graphe). Composants et relations peuvent être étiquetés, c.-à-d. qu'il est possible d'associer une ou plusieurs valeurs (symboliques ou numériques) sur chaque sommet et chaque arc afin de différencier les différents types de composants et les différents types de relations. Leur très forte expressivité fait que les graphes sont utilisés dans de nombreuses applications. Notons que les chaînes et les arbres sont des graphes particuliers.

2 Motivations

On s'intéresse dans ce mémoire à la comparaison automatique de graphes, c.-à-d. à quantifier et à qualifier les différences et les points communs de deux graphes.

Les mesures de similarité ou de distance de graphes existantes n'ont pas toutes la même tolérance aux dissimilarités pouvant exister entre les deux graphes à comparer. Par exemple, la mesure de similarité de graphes basée sur le problème de l'isomorphisme de graphes n'admet aucune différence dans la structure des deux graphes : deux graphes sont jugés soit identiques (si les deux graphes ont exactement la même structure) soit différents (si leur structure diffère ne serait ce que d'un seul arc ou sommet). *A contrario*, la mesure de similarité basée sur la distance d'édition de deux graphes permet une tolérance plus grande aux erreurs et admet que deux graphes puissent être similaires sans pour autant avoir une structure identique.

Toutes les mesures de la distance de deux graphes sont définies par rapport à un appariement des sommets de ces deux graphes, c.-à-d. une relation entre leurs sommets. Les mesures proposées se basent généralement sur un appariement univoque des sommets, c.-à-d. une relation où chaque sommet d'un graphe est lié à au plus un autre sommet de l'autre graphe. Néanmoins, dans de nombreuses applications, cette restriction aux appariement univoques ne permet pas d'exprimer de façon satisfaisante la distance entre les documents représentés par les graphes et il est nécessaire de définir des mesures de distance de graphes basées sur des appariements multivoques, c.-à-d. des appariements où chaque sommet d'un graphe peut être apparié à un ensemble de sommets de l'autre graphe.

Les mesures de similarité ou de distance de graphes existantes admettent des formulations très différentes les unes des autres. Par exemple, la mesure basée sur l'isomorphisme de graphe est définie par rapport à l'existence d'une fonction d'isomorphisme entre deux graphes alors que la distance d'édition de graphes est basée sur la recherche d'un plus court chemin d'édition entre deux graphes.

Un premier objectif de cette thèse a été de mettre en évidence les point communs à toutes ces mesures : chacune de ces mesures est basée sur un problème de recherche d'un meilleur appariement entre deux graphes, c.-à-d. une meilleure mise en correspondance des sommets des deux graphes permettant d'identifier les similarités de ces graphes. Selon les contraintes imposées sur l'appariement recherché, la mesure sera plus ou moins tolérante à certaines dissimilarités entre les deux graphes. Nous avons alors été amené à proposer une nouvelle définition de la distance entre deux graphes (permettant également de définir de nouvelles mesures de similarité de graphes). Nous montrons que cette nouvelle distance est générique dans le sens où elle peut être paramétrée de façon à devenir équivalente à chacune des mesures de similarité existantes.

Un second objectif de cette thèse a été de proposer des algorithmes efficaces pour calculer la similarité ou la distance entre deux graphes. En effet, dans le cas général, comparer deux graphes est un problème difficile et, bien que de nombreuses techniques aient déjà été proposées, aucune ne permet de calculer efficacement la similarité de deux graphes ayant plus d'une dizaine de sommets.

3 Plan du mémoire

On s'intéresse dans la première partie de ce mémoire aux mesures de la distance ou de la similarité de deux graphes. Le chapitre 1 présente les définitions et les notations utiles pour la lecture de ce document.

Nous proposons au chapitre 2 un état de l'art des mesures de distance/similarité de graphes.

Nous citons quelques applications utilisant la représentation d'objets en graphes afin de rechercher ou de classifier ces objets. Nous présentons ensuite les mesures de distance/similarité de graphes existantes dans la littérature. Nous verrons que ces mesures se basent toutes sur la recherche d'un appariement des deux graphes à comparer. Nous discutons ensuite des points communs et des différences entre ces mesures. Nous concentrons ensuite notre attention sur le cas particulier de la mesure de similarité de graphes de Champin et Solnon [CS03]. Nous avons en effet montré dans [SS05a, SS05b] que cette mesure était générique dans le sens où elle peut être paramétrée afin de la rendre équivalente à toutes les mesures existantes. Nous montrons cependant que cette mesure, bien que générique, est difficile à utiliser dans certains contextes. Nous terminons le chapitre 2 par une discussion sur la nécessité de proposer une nouvelle mesure générique de la distance de deux graphes.

Nous décrivons cette nouvelle mesure générique au chapitre 3. Cette mesure est basée sur la recherche d'un meilleur appariement multivoque de deux graphes. Elle est paramétrée par des fonctions de distance de sommets et d'arcs exprimant les contraintes et les préférences sur l'appariement recherché. Nous montrons que notre mesure est générique. En paramétrant correctement les fonctions de distance de sommets et d'arcs de cette mesure, elle peut être rendue équivalente à toutes les mesures existantes. Ce chapitre reprend en partie les résultats publiés dans [SSJ06a].

La seconde partie de ce mémoire s'intéresse au calcul de notre mesure générique de la distance de deux graphes. Calculer notre mesure consiste à trouver un meilleur appariement parmi tous les appariements possibles des deux graphes à comparer. Le chapitre 4 présente la complexité de ce problème ainsi qu'un état de l'art des approches existantes pour résoudre ces problèmes. Nous distinguons d'une part les approches complètes et d'autre part les approches incomplètes.

Les résultats que nous avons présentés dans [SCS03] montrent que, dans le cas général, les approches complètes peuvent calculer la distance que de très petits graphes (quelques sommets seulement). Nous proposons donc au chapitre 5 une recherche locale taboue réactive pour le calcul de notre distance générique de deux graphes. La généricité de notre approche nous permet de donner des résultats expérimentaux sur 4 classes de problèmes d'appariements de graphes différents. Ce chapitre reprend en partie les résultats publiés dans [SSJ06b, SSJ06a, SSSG06b, SSSG06a].

Au chapitre 6, nous nous intéressons à la résolution du problème de l'isomorphisme de graphes par la programmation par contraintes, un cas particulier de la mesure de la similarité de deux graphes. Nous proposons une contrainte globale dédiée à ce problème, nous définissons une consistance partielle pour cette contrainte (la d-IDL-consistance) et un algorithme permettant de l'établir. Cette consistance est une généralisation de deux consistances que nous avons précédemment proposées : la label-consistance [SS04a, SS04b] et l'ILL-consistance [SS06]. Nous montrons ensuite que l'utilisation de cette consistance rend la programmation par contraintes compétitive par rapport au meilleur algorithme connu dédié au problème de l'isomorphisme de graphes.

Première partie

Mesurer la distance de deux graphes

1

Définitions et notations

Ce chapitre contient toutes les définitions et notations utilisées dans la suite de ce document.

1.1 Graphes

1.1.1 Graphes orientés

Graphe orienté. Un *graphe orienté* est défini par un couple $G = (V, E)$ tel que :
- V est un ensemble fini de *sommets* ;
- $E \subseteq V \times V$ est un ensemble de couples de sommets appelés *arcs*.

Extrémités d'un arc. É donné un arc $(u, v) \in E$, les sommets u et v sont appelés les *extrémités* de l'arc (u, v).
Arcs sortants, arcs entrants. Étant donné un sommet $u \in V$, les arcs $(u, v) \in E$ sont appelés les *arcs sortants* du sommet u et les arcs $(v, u) \in E$ sont appelés les *arcs entrants* du sommet u.

1.1.2 Graphe non-orienté

Graphe non-orienté. Un graphe $G = (V, E)$ est dit **non-orienté** si les arcs (u, v) et (v, u) sont considérés comme identiques. Les arcs d'un graphe non-orienté sont appelés *arêtes*.
Arêtes incidentes. Étant donné un sommet $u \in V$, les arêtes $(u, v) \in E$ sont appelées les *arêtes incidentes* du sommet u.

1.1.3 Graphe étiqueté

Graphe étiqueté. Un graphe étiqueté est défini par un tuple $G = (V, E, L_V, \alpha, L_E, \beta)$ où :
- (V, E) est un graphe ;
- L_V est un ensemble d'étiquettes de sommets ;
- $\alpha : V \to L_V$ est une application attribuant une étiquette à chaque sommet du graphe ;
- L_E est un ensemble d'étiquettes d'arcs ;
- $\beta : E \to L_E$ est une application attribuant une étiquette à chaque arc du graphe.

Notons que sans perte de généralité, tout graphe non-étiqueté peut être représenté par un graphe étiqueté. Il suffit pour cela d'associer la même étiquette de sommets à tous les sommets et la même étiquette d'arcs à tous les arcs.
Graphe multi-étiqueté. Un graphe multi-étiqueté G est défini par un tuple $G = (V, L_V, r_V, L_E, r_E)$ où :
- V est un ensemble de sommets ;

- L_V est un ensemble d'étiquettes de sommets ;
- $r_V \subseteq V \times L_V$ est une relation associant les sommets à leurs étiquettes, *i.e.*, r_V est l'ensemble des couples (v, l) tels que le sommet v a pour étiquette l ;
- L_E est un ensemble d'étiquettes d'arcs ;
- $r_E \subseteq V \times V \times L_E$ est une relation associant les arcs à leurs étiquettes, *i.e.*, r_E est l'ensemble des triplets (v, v', l) tels que l'arc (v, v') a pour étiquette l. Sans perte de généralité, on supposera que chaque arc possède au moins une étiquette. Par conséquent, l'ensemble E des arcs est défini par $E = \{(v, v') | \exists l, (v, v', l) \in r_E\}$.

1.1.4 Sous-graphe

Sous-graphe partiel. Un graphe $G' = (V', E')$ est un *sous-graphe partiel* d'un graphe $G = (V, E)$ (et on note $G' \subseteq_p G$) si et seulement si $V' \subseteq V$ et $E' \subseteq E \cap (V' \times V')$.
Sous-graphe induit. Un graphe $G' = (V', E')$ est un *sous-graphe induit* d'un graphe $G = (V, E)$ (et on note $G' \subseteq_i G$) si et seulement si $V' \subseteq V$ et $E' = E \cap (V' \times V')$. Un sous-graphe induit $G' = (V', E')$ d'un graphe $G = (V, E)$ est le graphe qui contient tous les arcs de G ayant leurs deux extrémités dans V'. Par conséquent, un sous-graphe induit d'un graphe G est également un sous-graphe partiel de G.

Les notions de sous-graphe partiel et de sous-graphe induit peuvent être généralisées aux graphes étiquetés, les sommets et les arcs du sous-graphe G' de G devant alors avoir les mêmes étiquettes que dans G.

1.2 Appariements de graphes

Afin de comparer deux graphes, il est nécessaire de mettre en correspondance leurs sommets, c.-à-d. de définir un appariement (ou matching) entre ces graphes. Les définitions suivantes sont données pour des graphes non-étiquetés mais sont les mêmes pour les graphes étiquetés.
Appariement de graphes. Étant donnés deux graphes $G = (V, E)$ et $G' = (V', E')$, un *appariement multivoque* m entre G et G' est une relation entre V et V', c.-à-d. $m \subseteq V \times V'$. Sans perte de généralité, nous supposerons que les ensembles de sommets de G et de G' sont disjoints, c.-à-d. $V \cap V' = \emptyset$.
Sommets appariés à un sommet. Étant donné un appariement m de deux graphes $G = (V, E)$ et $G' = (V', E')$, nous notons $m(v)$ l'ensemble des sommets appariés au sommet v. Plus formellement, nous définissons :

$$\begin{aligned} \forall v \in V, m(v) &\doteq \{v' \in V' | (v, v') \in m\} \\ \forall v' \in V', m(v') &\doteq \{v \in V | (v, v') \in m\} \end{aligned}$$

Par extension, quand l'ensemble des sommets appariés au sommet v est un singleton (c.-à-d. quand $|m(v)| = 1$), nous utiliserons $m(v)$ pour dénoter l'unique élément de $m(v)$.

Quand aucune contrainte n'est ajoutée à l'appariement, c.-à-d. quand chaque sommet peut être apparié à 0, 1 ou plusieurs sommets, l'appariement est dit *multivoque*.

Cependant, il est possible d'ajouter des contraintes sur le nombre de sommets avec lesquels un sommet peut être apparié. Cela nous amène à définir les appariements *fonctionnels*, *applicatifs*, *univoques* (ou surjectifs), *injectifs* et *bijectifs*. Étant donnés deux graphes $G = (V, E)$ et $G' = (V', E')$, un appariement $m \subseteq V \times V'$ est :

– un *appariement fonctionnel* de G dans G' si m apparie chaque sommet du graphe G à au plus un sommet du graphe G' :

$$\forall v \in V \quad , \quad |m(v)| \leq 1$$

– un *appariement applicatif* de G dans G' si m apparie chaque sommet de G à exactement un sommet de G' :

$$\forall v \in V \quad , \quad |m(v)| = 1$$

– un *appariement univoque* (ou surjectif) entre G et G' si m apparie chaque sommet de G et de G' à au plus un sommet :

$$\begin{aligned} \forall v \in V \quad , \quad |m(v)| \leq 1 \wedge \\ \forall v' \in V' \quad , \quad |m(v')| \leq 1 \end{aligned}$$

– un *appariement injectif* de G dans G' si m apparie chaque sommet de G à un sommet différent de G' :

$$\begin{aligned} \forall v \in V \quad , \quad |m(v)| = 1 \wedge \\ \forall (u,v) \in V \times V \quad , \quad u \neq v \Rightarrow m(u) \neq m(v) \end{aligned}$$

Une autre définition possible d'un appariement injectif de G dans G' est un appariement m tel que :

$$\begin{aligned} \forall v \in V \quad , \quad |m(v)| = 1 \wedge \\ \forall v' \in V' \quad , \quad |m(v')| \leq 1 \end{aligned}$$

– un *appariement bijectif* entre G et G' si m apparie chaque sommet de G (resp. de G') à un sommet différent de G' (resp. de G) :

$$\begin{aligned} \forall v \in V \quad , \quad |m(v)| = 1 \wedge \\ \forall v' \in V' \quad , \quad |m(v')| = 1 \end{aligned}$$

Arcs appariés lors d'un appariement. Étant donné un appariement m des sommets de deux graphes $G = (V, E)$ et $G' = (V', E')$, un arc $(u, v) \in E$ est dit apparié à un autre arc $(u', v') \in E'$ si et seulement si $\{(u, u'), (v, v')\} \subseteq m$. Comme pour les sommets, par extension, nous notons $m(u, v)$ l'ensemble des arcs appariés à l'arc (u, v) par l'appariement m :

$$\begin{aligned} \forall (u,v) \in E, m(u,v) &\doteq \{(u',v') \in E' | u' \in m(u), v' \in m(v)\} \\ \forall (u',v') \in E', m(u',v') &\doteq \{(u,v) \in E | u \in m(u'), v \in m(v')\} \end{aligned}$$

Notons qu'un appariement de graphe est une relation entre les sommets de deux graphes et que de cette relation, on en déduit un appariement des arcs.

Sous-graphe induit par un appariement. Étant donné un appariement m de deux graphes $G = (V, E)$ et $G' = (V', E')$, le sous-graphe de G (resp. G') induit par l'appariement m est noté $G_{\downarrow m} = (V_{\downarrow m}, E_{\downarrow m})$ (resp. $G'_{\downarrow m} = (V'_{\downarrow m}, E'_{\downarrow m})$) où $V_{\downarrow m}$ et $E_{\downarrow m}$ (resp. $V'_{\downarrow m}$ et $E'_{\downarrow m}$) sont les ensembles de sommets et d'arcs de G (resp. G') appariés à au moins un sommet ou un arc de G' (resp. G) :

$$\begin{aligned} V_{\downarrow m} = \{v \in V | m(v) \neq \emptyset\} \quad , \quad E_{\downarrow m} = \{(u,v) \in E | m(u,v) \neq \emptyset\} \\ V'_{\downarrow m} = \{v' \in V' | m(v') \neq \emptyset\} \quad , \quad E'_{\downarrow m} = \{(u',v') \in E' | m(u',v') \neq \emptyset\} \end{aligned}$$

Étant donné un appariement m de deux graphes $G = (V, E)$ et $G' = (V', E')$, si le sous-graphe $G_{\downarrow m}$ de G induit par l'appariement m est identique à G (c.-à-d. quand $V_{\downarrow m} = V$ et $E_{\downarrow m} = E$), l'appariement m définit un *homomorphisme* de G dans G'. Si m est en plus un appariement injectif, m définit un *monomorphisme* (ou un isomorphisme de sous-graphe partiel) de G dans G'.

2

Mesures existantes

Ce chapitre a pour but de sensibiliser le lecteur à l'intérêt de l'utilisation des graphes pour la modélisation puis la comparaison automatique d'objets. La section 1 montre en quoi et dans quel contexte les graphes sont intéressants pour modéliser des objets. Des exemples d'applications qui utilisent la modélisation des objets en graphes sont donnés.

La section 2 dresse l'état de l'art des mesures de similarité ou de distance de graphes existantes. Ces mesures diffèrent les unes des autres par les contraintes et les préférences qu'elles posent sur l'appariement recherché entre les deux graphes à comparer. Nous avons choisi de les regrouper et de les ordonner des plus restrictives aux plus souples.

2.1 Représentation d'objets avec des graphes

Les graphes orientés sont une façon très riche de modéliser des documents et leurs structures. Un document est représenté comme un ensemble de composants (les sommets du graphe) et un ensemble de relations binaires entre ces composants (les arcs du graphe). Composants et relations peuvent être étiquetés, c.-à-d. qu'il est possible d'associer une ou plusieurs valeurs (symboliques ou numériques) sur chaque sommet et chaque arc afin de différencier les différents types de composants et les différents types de relations. Leur très forte expressivité fait que les graphes sont utilisés dans de nombreuses applications.

Chimie organique et molécules. Dans [Rég95], l'auteur s'intéresse à la chimie organique et à la façon de synthétiser des molécules et a besoin pour cela de tester si une molécule entre dans la composition d'une autre. Pour la synthèse de molécules, la façon dont les atomes sont assemblés est essentielle et représenter une molécule uniquement par la quantité de chaque type d'atomes qui la composent est insuffisant. Par conséquent, l'auteur représente des molécules par des graphes : chaque sommet représente un atome et chaque arc représente une liaison entre deux atomes. Les sommets sont étiquetés par le type d'atome qu'ils représentent (par exemple "O" pour oxygène, "C" pour carbone...) et les arcs sont étiquetés par le type de liaison qu'ils représentent (liaison simple, liaison double...). La figure 2.1 représente le benzène (C_6H_6). Comme on peut le voir, la représentation "classique" par attribut-valeur C_6H_6 (c.-à-d. type d'atomes - nombre de ces atomes) ne permet pas de refléter la structure de la molécule alors que la représentation en graphe le permet.

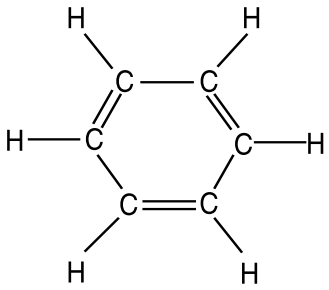


FIG. 2.1 – Exemple de représentation en graphe du benzène C_6H_6. Chaque sommet représente un atome de la molécule. Les arêtes représentent des liaisons simples ou doubles entre deux atomes.

Biologie et réseaux biochimiques Dans [DDD05a], Dooms *et al.* s'intéressent au problème de la découverte d'interactions dans des réseaux biochimiques. Ils modélisent pour cela les réseaux par des graphes (chaque sommet correspond à des composants d'une cellule ; chaque arc correspond à une interaction entre des composants lors d'une réaction biochimique) et comparent ces graphes (ou cherchent des chemins contraints) afin de découvrir de nouvelles interactions. L'analyse de ces réseaux biochimiques permet, entre autres, de découvrir des liens de parenté (au sens évolutionniste) entre différentes protéines ou de trouver des protéines jouant un rôle similaire.

Recherche d'informations et documents HTML. Les graphes ont également été utilisés par Schenker et al [SLBK04] pour comparer des documents Web au format HTML. Chaque document est représenté par un graphe où chaque mot présent dans le document est représenté par un sommet (si le mot apparaît plusieurs fois, un seul sommet représente l'ensemble des occurrences du mot) et où chaque arc entre deux sommets représente le fait qu'il existe au moins une occurrence de la succession des deux mots correspondants dans le texte. Trois étiquettes d'arcs existent : (TX) pour indiquer que la succession des deux mots est présente dans le texte de la page, (L) pour indiquer sa présence dans un lien HTML et (TI) pour indiquer sa présence dans le titre de la page. La figure 2.2 montre une représentation partielle d'une page exprimant le fait que le document contient le texte "REUTERS NEWS SERVICE REPORTS", qu'il a pour titre "YAHOO NEWS" et un lien sur le texte "MORE NEWS".

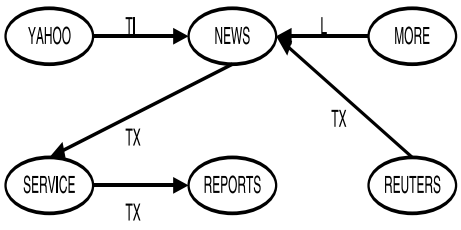


FIG. 2.2 – Exemple de représentation (partielle) d'une page Web en graphe (exemple issu de [SLBK04]). Chaque sommet représente un mot du document. Un arc est ajouté entre deux mots lorsque ceux-ci se suivent. Les arcs sont typés par TX, L ou TI selon si l'occurrence du couple de mots est présent dans du texte, dans un lien ou dans le titre de la page.

Notons que, contrairement à l'exemple de la chimie organique, la représentation des pages Web en graphes réduit fortement la quantité d'informations contenue dans la page. Par conséquent, plusieurs documents Web différents peuvent être représentés par le même graphe. Néanmoins, les auteurs supposent que deux pages Web étant représentées par le même graphe seront nécessairement sémantiquement corrélées : elles auront des titres similaires, des hyperliens similaires... L'objectif des auteurs étant de retrouver des pages Web traitant du même sujet, cette réduction de l'information n'est pas un problème : deux pages sont représentées par le même graphe que si ces deux pages traitent du même sujet.

CAO et objets de conception. Champin [Cha02] présente une application de raisonnement à partir de cas sur des objets de conception assistée par ordinateur. Dans ce contexte, l'auteur cherche à identifier de façon automatique tous les objets déjà conçus similaires à un objet à concevoir. Les objets de conception sont décrits en termes de composants (murs, boulons, poutres...) et de relations entre ces composants (une poutre est sur un mur, une poutre est à côté d'une autre poutre...). Par conséquent, afin de comparer ces objets, l'auteur les modélise par des graphes multi-étiquetés. Chaque sommet du graphe modélise un composant de l'objet et chaque arc une relation binaire entre ces composants. Un ensemble d'étiquettes est associé à chaque arc et chaque sommet afin de distinguer les différents types de relations et de composants.

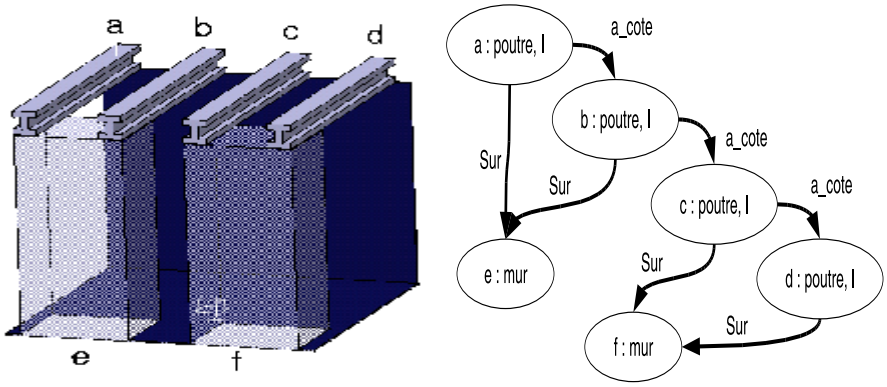


FIG. 2.3 – Exemple de représentation d'un objet de conception (à gauche de la figure) en graphe multi-étiqueté (à droite de la figure) [Cha02]. Chaque sommet représente un composant de l'objet. Différentes relations binaires peuvent exister entre les composants ("être sur", "être à côté de"...). Chaque composant (ou relation) peut avoir plusieurs étiquettes. Ici, les objets a, b, c et d sont des "poutres" ayant une forme en "I".

Vision artificielle et images. Les graphes sont également très fréquemment utilisés en reconnaissance d'images. Les images sont généralement segmentées en régions. Chaque sommet du graphe représente alors une région de l'image et les arcs représentent les différentes relations possibles entre les régions (connectivité, distance...). De nombreuses caractéristiques sont généralement associées aux sommets afin de décrire la taille des régions associées, leur forme, leur couleur... Un exemple de modélisation d'une image par un graphe est donné en figure 2.4. L'intérêt principal de l'utilisation des graphes pour la représentation d'images est l'intégration d'informations spatiales dans la modélisation. En effet, les représentations classiques telles que les histogrammes de couleurs, les descripteurs de textures, etc... ne donnent aucune information sur la façon dont les régions d'intérêt de l'image sont agencées. *A contrario*, la représentation par

graphes permet de décrire la structure de l'image, c.-à-d. la façon dont les régions sont disposées les unes par rapport aux autres. En outre, selon les types de relations choisies, cette représentation en graphe peut être invariante à certaines transformations telles que les rotations de l'image ou les translations de certaines parties de l'image.

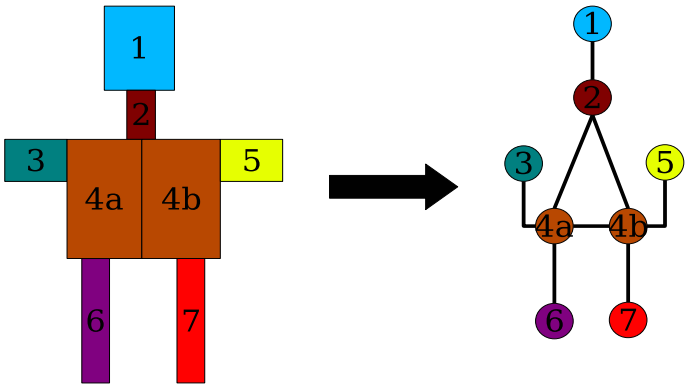


FIG. 2.4 – Exemple de représentation d'une image (à gauche) en graphe (à droite) issue de [AFB03]. L'image est segmentée en région puis chaque région de l'image est représentée par un sommet. Une arête est présente entre deux régions si ces régions sont adjacentes.

Les graphes sont également utilisés dans des contextes plus spécifiques de la reconnaissance d'images. Citons par exemple la modélisation et la reconnaissance d'empreintes digitales par Neuhaus et Bunke [NB04] ou une application de raisonnement à partir de cas de Le Ber et al [LNML03] où les cas sont des cartes représentant des environnements agricoles (c.-à-d. des cartes situant une ferme, les forêts alentours, les points d'eau...).

Conception logicielle et patrons de conception. Les patrons de conception [GHJV95] sont utilisés en génie logiciel pour décrire des méthodes génériques de résolution de problèmes récurrents en programmation orientée objets. L'utilisation des patrons de conception est fortement conseillée car ils définissent des architectures efficaces, élégantes et de qualité issues de nombreuses années d'expérience. Ils permettent une maintenance plus simple du code et de meilleures garanties de qualité, deux enjeux importants pour l'industrie du logiciel. Néanmoins, écrire du code respectant un patron de conception n'est pas aisé : cela nécessite que le développeur ait une vue globale de la tâche à réaliser et une connaissance parfaite de tous les patrons existants.

Dans [AFC98], Antoniol *et al.* présentent une application capable de retrouver, dans un code orienté objet existant, des occurrences (exactes ou dégradées) de micro-architectures définies dans des patrons de conceptions. À partir du code, un graphe est généré automatiquement : chaque sommet de ce graphe représente une classe et les arcs représentent les liens entre ces classes modélisant des relations d'instanciation, d'héritage ou de composition entre ces classes. Il s'agit ensuite de retrouver des occurrences d'un graphe modélisant un patron de conception à l'intérieur du graphe modélisant le code.

Albin-Amiot *et al.* [AACGJ01] proposent également d'identifier de façon automatique des patrons de conception. Le problème est modélisé en problème de satisfaction de contraintes. Néanmoins, comme les variables et les contraintes de ce CSP modélisent le diagramme UML du patron à reconnaître et que le domaine de ces variables représente les entités existantes dans le

code orienté objet existant, ce CSP peut être vu comme un problème d'appariement de deux graphes.

Les graphes sont également utilisés pour la conception de bases de données. Les diagrammes entités-associations de la méthode Merise ou les diagrammes UML sont des graphes. Melnik *et al.* [MGMR02] utilisent la comparaison de graphes pour la comparaison de schémas de bases de données. L'intérêt d'une telle application est de pouvoir fusionner de façon automatique des données provenant de sources différentes.

2.2 Comparaison de graphes

Toutes les applications citées dans la section précédente représentent des objets par des graphes dans le but de les comparer. Plus précisément, l'objectif est :

- de trouver, parmi un ensemble d'objets, le ou les objets les plus similaires à un autre (cas de la recherche de documents HTML, d'objets de conception ou d'images similaires) ;
- de trouver une occurrence (dégradée ou non) d'un objet dans un autre objet plus complexe (cas de l'analyse de la composition d'une molécule en chimie organique ou de la recherche de patrons de conception) ;
- d'identifier, étant donnés deux objets, leurs points communs (cas des réseaux biochimiques ou de la fusion de bases de données).

Toutes ces opérations de comparaison nécessitent de définir formellement une mesure de similarité ou de distance entre deux objets. Lorsque les objets sont modélisés par des graphes, il est donc nécessaire de définir une mesure de similarité (ou de distance) entre tout couple de graphes. Comparer deux graphes consiste à comparer leur structure, c.-à-d. la répartition de leurs arcs sur leurs sommets. Les sommets des deux graphes sont mis en correspondance et la similarité des structures dépend du nombre d'arcs que cette mise en correspondance des sommets permet de "retrouver".

Nous avons vu dans la section précédente que dans de nombreuses applications les sommets et les arcs des graphes sont valués afin de qualifier les différents composants et les différentes relations entre ces composants. Par conséquent, dans ces applications, la distance entre deux graphes dépendra de leur structure mais également des différences entre les valeurs portées par les sommets et les arcs mis en correspondance.

Cette section dresse un état de l'art des mesures de similarité ou de distance de graphes existantes. Ces mesures diffèrent les unes des autres par les contraintes et les préférences qu'elles posent sur l'appariement recherché entre les deux graphes à comparer. Nous avons choisi de les regrouper et de les ordonner des plus restrictives aux plus souples :

- la première catégorie est composée des méthodes basées sur la recherche d'un appariement univoque respectant un ensemble de contraintes dures ;
- la seconde catégorie regroupe les méthodes basées sur la recherche d'un appariement univoque respectant un ensemble de contraintes dures et satisfaisant au maximum un ensemble de contraintes souples ;
- la troisième catégorie regroupe les méthodes basées sur des appariements multivoques des sommets des deux graphes : chaque sommet peut cette fois-ci être mis en correspondance avec un ensemble quelconque de sommets de l'autre graphe ; un ensemble de contraintes dures et de contraintes souples sur les appariements possibles sont ajoutées.

Nous nous intéressons dans un premier temps à l'expressivité des mesures existantes et non pas à la complexité de leur calcul que nous introduisons au chapitre 4. Néanmoins, de façon très générale, il est possible de dire que le calcul de la majorité des mesures que nous présentons est

un problème NP-Complet ou NP-Difficile et, qu'en pratique, le calcul d'une mesure est d'autant plus difficile que la mesure est expressive.

Les mesures de distance ou de similarité de graphes que nous présentons ici ont souvent été initialement proposées pour des graphes non-étiquetés. Néanmoins, elles admettent toutes des généralisations aux graphes étiquetés. Nous considérons donc qu'un graphe non-étiqueté est un graphe où chaque sommet et chaque arc a la même étiquette et nous définissons ces mesures pour les graphes étiquetés uniquement.

La majorité des mesures de similarité/distance de graphes que nous présentons ont été initialement définies par rapport à une fonction des sommets d'un graphe vers les sommets d'un autre graphe. Cependant, par souci d'homogénéité, nous avons choisi, lorsque c'était possible, de redéfinir ces mesures par rapport à un appariement (fonctionnel, injectif, bijectif...) des sommets des deux graphes (voir le chapitre 1 pour un rappel de la définition des ces différent types d'appariements).

2.2.1 Appariements univoques sous contraintes dures

Dans certaines applications, deux objets seront jugés comparables si leurs structures et/ou le type de leurs composants et relations sont identiques. Par conséquent, des méthodes qui ne tolèrent aucune différence entre la structure de deux graphes ont été introduites. Ces problèmes sont des problèmes de décision. Étant donnés deux graphes, le problème consiste à décider si les deux graphes ont des structures totalement identiques ou si la structure d'un des deux graphes est identique à une sous partie de la structure de l'autre graphe.

Tous ces problèmes reposent sur la recherche d'un appariement univoque des sommets des deux graphes respectant un ensemble de contraintes dures.

Isomorphisme de graphes

Le problème de l'isomorphisme de deux graphes consiste à prouver que deux graphes sont isomorphes, c.-à-d. que les deux graphes sont identiques à un renommage de leurs sommets près.

Définition (Isomorphisme de graphes). Étant donnés deux graphes étiquetés $G = (V, E, L_V, \alpha, L_E, \beta)$ et $G' = (V', E', L_V, \alpha', L_E, \beta')$ tels que $|V| = |V'|$, G et G' sont isomorphes si et seulement s'il existe un appariement bijectif $m \subseteq V \times V'$ tel que $\forall (u, v) \in V^2, (u, v) \in E \Leftrightarrow (m(u), m(v)) \in E'$ et $\forall u \in V, \alpha(u) = \alpha'(m(u))$ et $\forall (u, v) \in E, \beta((u, v)) = \beta'((m(u), m(v)))$.[1]

Isomorphisme de sous-graphe (induit)

Dans certaines applications, il est intéressant de savoir si un objet contient un autre objet. Par exemple, en recherche de documents, une requête peut être vue comme un objet et les documents qui répondent à cette requête comme des objets plus complexes contenant l'objet requête. En reconnaissance d'images, il est parfois nécessaire de rechercher un objet particulier à l'intérieur d'une scène (c.-à-d. une image composée de plusieurs objets). En chimie organique [Rég95], il peut être utile de vérifier la présence d'une molécule dans la composition d'une autre molécule plus complexe.

[1] Rappelons que pour les appariements univoques (et *a fortiori* les appariements bijectifs), quand l'ensemble des sommets appariés à un sommet v est un singleton, c.-à-d. quand $|m(v)| = 1$, nous notons $m(v)$ l'unique sommet élément de $m(v)$.

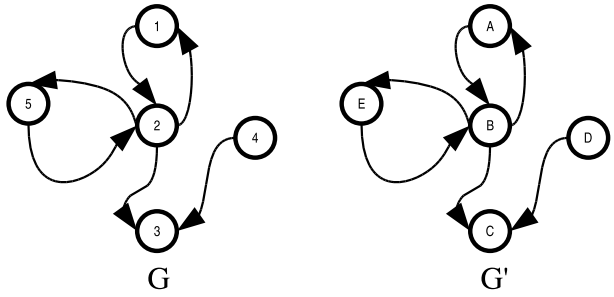


FIG. 2.5 – Exemple de deux graphes isomorphes. Un appariement d'isomorphisme possible entre ces graphes est $m = \{(1, A), (2, B), (3, C), (4, D), (5, E)\}$.

Le problème de l'isomorphisme de sous-graphe (induit) consiste à prouver qu'un graphe est isomorphe à une sous-partie d'un autre graphe, autrement dit, que le graphe est "inclus" dans l'autre graphe. Par conséquent, lorsque des objets sont modélisés par des graphes, cela permet de savoir si un objet est une partie d'un autre objet.

Définition (Isomorphisme de sous-graphe induit). Étant donnés deux graphes étiquetés $G = (V, E, L_V, \alpha, L_E, \beta)$ et $G' = (V', E', L'_V, \alpha', L'_E, \beta')$ tels que $|V| \leq |V'|$, le graphe étiqueté G est isomorphe à un sous-graphe (induit) du graphe étiqueté G' si et seulement s'il existe un appariement injectif $m \subseteq V \times V'$ tel que $\forall (u, v) \in V^2, (u, v) \in E \Leftrightarrow (m(u), m(v)) \in E'$ et $\forall u \in V, \alpha(u) = \alpha'(m(u))$ et $\forall (u, v) \in E, \beta((u, v)) = \beta'((m(u), m(v)))$. L'appariement m est alors appelé relation d'isomorphisme de sous-graphe induit.

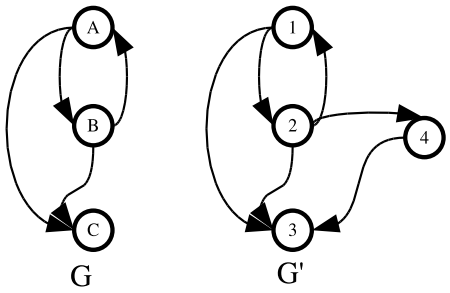


FIG. 2.6 – Exemple d'isomorphisme de sous-graphe (induit). Un appariement d'isomorphisme de sous-graphe induit possible entre ces graphes est $m = \{(A, 1), (B, 2), (C, 3)\}$.

Isomorphisme de sous-graphe partiel

Dans une relation d'isomorphisme de sous-graphe induit entre un graphe G et un graphe G', tous les arcs entre les sommets de G' appariés à un sommet de G doivent être appariés à un arc de G. La structure de G doit donc être préservée au sens strict, c.-à-d. que tous les couples de sommets de G qui ne sont pas reliés par un arc doivent être appariés à un couple de sommets de G' qui ne sont pas reliés par un arc. Les relations d'isomorphisme de sous-graphe partiel entre deux graphes G et G' n'ont pas cette contrainte.

Définition (Isomorphisme de sous-graphe partiel). Étant donnés deux graphes étiquetés $G = (V, E, L_V, \alpha, L_E, \beta)$ et $G' = (V', E', L_V, \alpha, L_E, \beta)$ tels que $|V| \leq |V'|$, le graphe G est un sous-graphe partiel du graphe G' si et seulement s'il existe un appariement injectif $m \subseteq V \times V'$ tel que $\forall (u, v) \in E, (m(u), m(v)) \in E', \forall v \in V, \alpha(v) = \alpha(m(v))$ et $\forall (u, v) \in E, \beta((u, v)) = \beta'((m(u), m(v)))$. L'appariement m est alors appelé relation d'isomorphisme de sous-graphe partiel (ou relation de monomorphisme de G dans G'). Notons que, étant donnée cette définition, toute relation d'isomorphisme de sous-graphe induit est également une relation d'isomorphisme de sous-graphe partiel. La figure 2.7 donne un exemple d'une telle relation.

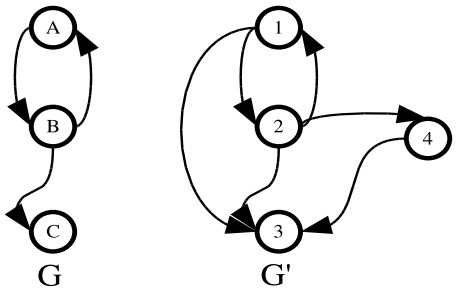


FIG. 2.7 – Exemple d'isomorphisme de sous-graphe partiel. Un appariement d'isomorphisme de sous-graphe partiel m possible est $m : \{(A, 1), (B, 2), (C, 3)\}$. L'appariement m n'est par contre pas une relation d'isomorphisme de sous-graphe induit : les sommets 1 et 3 du graphe G' sont reliés par un arc alors que leur antécédents par cette appariement injectif ne sont pas reliés par un arc.

Généralisation de l'isomorphisme de sous-graphe

Zampelli *et al.* [ZDD05] proposent une généralisation du problème de l'isomorphisme de sous-graphe (appelée "approximate subgraph matching problem") pour analyser des réseaux biochimiques. Ce problème consiste à rechercher l'occurrence d'un modèle de graphe dans un graphe cible. La spécificité de ce problème est que le graphe modèle est composé de sommets et d'arcs "obligatoires" (c.-à-d. des sommets et des arcs qui doivent nécessairement être retrouvés dans le graphe cible), des sommets "facultatifs" (c.-à-d. des sommets qui ne doivent pas nécessairement être retrouvés dans le graphe cible[2]) et des arcs "interdits" (c.-à-d. des arcs qui ne doivent être appariés à aucun arc du graphe cible).

Définition (Isomorphisme de sous-graphe généralisé). Étant donnés un graphe étiqueté modèle $G_p = (V_p, E_p, L_V, \alpha_p, L_E, \beta_p)$, un ensemble de sommets optionnels $O_p \subseteq V_p$, un ensemble d'arcs interdits $F_p \subseteq (V_p \times V_p) - E_p$ et un graphe étiqueté cible $G_t = (V_t, E_t, L_V, \alpha_t, L_E, \beta_t)$ le problème de l'isomorphisme de sous-graphe généralisé consiste à trouver un appariement[3] entre le graphe modèle et le graphe cible qui respecte toutes les contraintes du problème, c.-à-d. un appariement $m \subseteq V_p \times V_t$ tel que :

1. m est univalent ;

[2] Lorsqu'un sommet "optionnel" est apparié, tous ses arcs doivent être retrouvés. Notons finalement que cette notion de sommets optionnels n'a de sens que si le problème est sujet à d'autres contraintes car si ça n'est pas le cas, il suffit d'enlever les sommets optionnels et leurs arcs du graphe modèle pour obtenir le même résultat.

[3] Par souci de clarté et d'homogénéité, nous avons pris la liberté de changer la notation proposée dans [ZDD05].

2. $\forall u \in (V_p \backslash O_p), |m(u)| = 1$;
3. $\forall u \in V_p, m(u) \neq \emptyset \Rightarrow \alpha_p(u) = \alpha_t(m(u))$;
4. $\forall (u,v) \in E_p, m(u) \neq \emptyset \wedge m(v) \neq \emptyset \Rightarrow (m(u), m(v)) \in E_t \wedge \beta_p((u,v)) = \beta_t((m(u), m(v)))$;
5. $\forall (u,v) \in F_p, m(u) \neq \emptyset \wedge m(v) \neq \emptyset \Rightarrow (m(u), m(v)) \notin E_t$.

La contrainte 1 garantit que chaque sommet des deux graphes est apparié à au plus un sommet de l'autre graphe. La contrainte 2 garantit qu'au moins chaque sommet non optionnel du graphe modèle est apparié à un sommet du graphe cible. La contrainte 3 implique que tous les sommets du graphe modèle sont appariés à des sommets du graphe cible ayant la même étiquette. La contrainte 4 implique que tous les arcs du graphe modèle dont les extrémités sont appariées sont appariés à un arc du graphe cible ayant la même étiquette. Enfin, la contrainte 5 garantit que les arcs interdits ne sont appariés à aucun arc du graphe cible.

Ce problème peut être vu comme une généralisation du problème de l'isomorphisme de sous-graphe et de sous-graphe partiel. Un problème d'isomorphisme de sous-graphe induit entre un graphe $G = (V, E, L_V, \alpha, L_E, \beta)$ et un graphe $G' = (V', E', L_V, \alpha', L_E, \beta')$ peut être exprimé comme un problème d'isomorphisme de graphe généralisé entre G et G' avec $O_p = \emptyset$ (aucun sommet de G n'est optionnel) et $F_p = (V \times V) \backslash E$ (les couples de sommets de G qui ne sont pas des arcs ne doivent pas être appariés à des arcs de G'). Un problème d'isomorphisme de sous-graphe partiel entre un graphe $G = (V, E, L_V, \alpha, L_E, \beta)$ et un graphe $G' = (V', E', L_V, \alpha, L_E, \beta)$ peut également être exprimé comme un problème d'isomorphisme de sous-graphe généralisé entre G et G' avec $O_p = \emptyset$ et $F_p = \emptyset$ (aucune contrainte entre les couples de sommets de G qui ne sont pas des arcs n'est ajoutée).

2.2.2 Appariements univoques sous contraintes dures et souples

Les comparaisons de graphes citées précédemment s'appuient toutes sur des appariements de graphes exacts, c.-à-d. des appariements mettant en évidence que les structures des deux graphes comparés sont identiques ou respectent des contraintes dures sur l'existence ou l'inexistence d'un arc ou d'un sommet. Par conséquent, deux graphes ayant des structures similaires sans pour autant être exactement identiques sont jugés incomparables par ces mesures : la similarité entre les deux graphes sera jugée nulle et leur distance infinie. Dans de nombreuses applications, la comparaison des objets se fait en fonction de leurs similarités structurelles ce qui rend les mesures de similarité de graphes basées sur des appariements exacts inutilisables.

Dans [Tve77, Lin98], il est montré que deux objets sont d'autant plus similaires que l'intersection de leurs caractéristiques est grande. La similarité de deux objets a et b respectivement décrits par des ensembles A et B de caractéristiques est définie comme $taille(A \cap B)/taille(A \cup B)$.

Cette sous-section présente des problèmes de recherche d'un meilleur appariement univoque entre deux graphes. Des préférences, c.-à-d. des contraintes souples, sont exprimées sur le type d'appariement recherché mais une contrainte dure impose à ces appariements d'être univoques. Ces problèmes permettent d'identifier et de quantifier les points communs et les différences entre les structures de deux graphes. Le but n'est plus de montrer que deux graphes sont structurellement identiques (ou qu'un graphe est une sous-partie d'un autre) mais d'évaluer à quel point leurs structures sont semblables. Ces mesures permettent donc de s'inspirer de la mesure de Tversky [Tve77] pour définir des mesures de similarité ou de distance de graphes.

Plus grand sous-graphe (induit) commun

Définition (Plus grand sous-graphe induit commun). Un sous-graphe commun à deux graphes étiquetés G et G' (noté $mcs(G, G')$ pour *maximum common subgraph*) est un graphe

G'' tel qu'il existe un isomorphisme de sous-graphe induit entre G'' et G et entre G'' et G'. Un plus grand sous-graphe commun à deux graphes G et G' est un sous-graphe commun ayant un nombre de sommets et/ou d'arcs maximum.

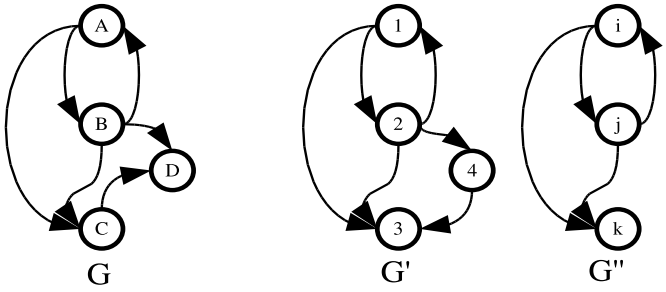


FIG. 2.8 – Exemple de plus grand sous-graphe induit commun. G'' est un plus grand (en terme de nombre de sommets) sous-graphe induit commun à G et G'.

Dans [BS98, Bun00], Bunke propose de définir la taille de l'intersection de deux graphes comme la taille de leur plus grand sous-graphe commun. Il s'appuie donc sur la mesure de Tversky [Tve77] pour proposer une définition de la distance $d(G, G')$ entre deux graphes G et G' :

$$d(G, G') = 1 - \frac{|mcs(G, G')|}{max(|G|, |G'|)}$$

où $|G|$ est la taille du graphe G (c.-à-d. son nombre de sommets et/ou d'arcs).

Dans [BS98], Bunke et Shearer montrent que, lorsque la taille d'un graphe est définie comme son nombre de sommets (quand $|G = (V, E)| = |V|$) cette distance est une métrique, c.-à-d. que quels que soient les graphes G, G' et G'' :

1. $d(G, G') = 0 \Leftrightarrow G$ est isomorphe à G' (séparation) ;
2. $d(G, G') = d(G', G)$ (symétrie) ;
3. $d(G, G') + d(G', G'') \geq d(G, G'')$ (inégalité triangulaire).

Le fait que la distance soit une métrique est une propriété intéressante : cela permet par exemple d'indexer des bases de graphes pour accélérer l'opération de recherche du graphe le plus similaire à un autre graphe [SBV00]. Cependant, si les propriétés d'une métrique sont algorithmiquement intéressantes, elle sont parfois inadaptées au domaine d'application [BS98]. C'est par exemple le cas de la symétrie en reconnaissance d'images : lorsqu'une image schématique d'un visage est présentée comme requête à un système de recherche d'images similaires, il est généralement souhaitable que le système retourne toutes les photographies de visages de la base d'images. *A contrario*, si la photographie d'un visage est soumise au système de recherche, il n'est généralement pas souhaitable d'obtenir l'image schématique d'un visage.

Plus grand sous-graphe partiel commun

Les définitions de distance (ou de similarité) s'appuyant sur le plus grand sous-graphe commun à deux graphes peuvent être aussi étendues à la notion de plus grand sous-graphe partiel commun à deux graphes. Comme pour l'isomorphisme de sous-graphe partiel, la contrainte imposant que tous les couples de sommets n'étant pas reliés par un arc doivent être reliés à des couples de sommets qui ne forment pas un arc est enlevée.

Définition (Plus grand sous-graphe partiel commun). Un sous-graphe partiel commun à deux graphes étiquetés G et G' est un graphe G'' tel qu'il existe un isomorphisme de sous-graphe partiel entre G'' et G et entre G'' et G'. Un plus grand sous-graphe partiel commun à deux graphes G et G' est un sous-graphe partiel commun ayant un nombre de sommets et/ou d'arcs maximum. Notons qu'ici, définir la taille d'un graphe comme uniquement son nombre de sommets n'a pas de sens car aucune contrainte sur les arcs n'est imposée pour le choix du graphe G''.

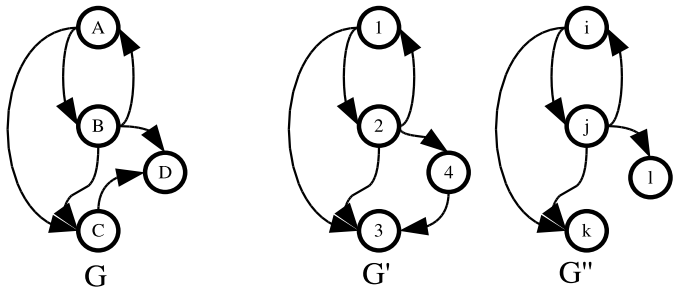


FIG. 2.9 – Exemple de plus grand sous-graphe partiel commun. G'' est un plus grand (en terme de nombre d'arcs) sous-graphe partiel commun à G et G'.

La distance de [BS98] peut être adaptée à la définition du plus grand sous-graphe partiel commun. La taille $|G|$ d'un graphe G doit alors être redéfinie comme une combinaison linéaire de son nombre de sommets et d'arcs. La distance obtenue ne sera alors pas forcément une métrique (elle ne respecte pas forcément l'inégalité triangulaire).

Distance d'édition de graphes

La distance d'édition de graphes proposée par Bunke et Jiang dans [BJ00, Bun97] est une mesure de distance inspirée de la distance d'édition de chaînes de caractères ou distance de Levenshtein [Lev65]. Étant donnés deux graphes étiquetés G et G', la distance d'édition de graphe (*ged* pour graph edit distance) entre G est G' est définie comme l'ensemble le moins coûteux d'opérations nécessaires pour rendre le graphe G isomorphe au graphe G'. Les opérations considérées sont l'insertion, la substitution (c.-à-d. le réétiquetage) et la suppression de sommets et d'arcs.

Définition (Distance d'édition de graphes). Étant donnés deux graphes étiquetés $G = (V, E, L_V, \alpha, L_E, \beta)$ et $G' = (V', E', L_V, \alpha', L_E, \beta')$, un appariement à tolérance d'erreurs est un appariement univalent $m \subseteq V \times V'$. Le sommet u est substitué par le sommet v si $m(u) = v$. Si $\alpha(u) = \alpha'(m(u))$, la substitution est dite "identique", sinon, elle est dite "non-identique". Tous les sommets $u \in V$ tels que $m(u) = \emptyset$ sont dits supprimés par m. Tous les sommets $v \in V'$ tels que $m(v) = \emptyset$ sont dits insérés par m. La même terminologie est utilisée pour les arcs des graphes.

Un coût est associé à chaque opération de substitution non-identique, d'insertion ou de suppression d'un sommet ou d'un arc. Le coût d'un appariement à tolérance d'erreurs m est alors défini comme la somme des coûts des opérations induites par m. Enfin, la distance d'édition de graphe entre deux graphes est définie comme le coût de l'appariement à tolérance d'erreurs le moins coûteux.

Bunke montre dans [Bun97] que, selon comment sont définis les coûts des opérations d'édition, la distance d'édition de graphe peut être égale à la distance basée sur le plus grand sous-graphe (induit ou partiel) commun à deux graphes (se référer à l'article [Bun97] pour plus de détails sur cette équivalence).

Petrovic *et al.* [PKY02] proposent d'utiliser un cas particulier de la distance d'édition (une distance d'édition où les poids des opérations d'édition sont fixés) pour la comparaison de cas dans un système de raisonnement à partir de cas.

2.2.3 Appariements multivoques

Les mesures de distance de graphes présentées jusqu'ici reposent toutes sur des appariements univoques de sommets. Chaque sommet d'un graphe est mis en correspondance avec au plus un sommet de l'autre graphe. Cette contrainte dure ne convient cependant pas à toutes les applications : selon la granularité de description des objets, le composant d'un objet (c.-à-d. le sommet d'un graphe) peut jouer le même "rôle" qu'un ensemble de composants d'un autre objet (c.-à-d. un ensemble de sommets).

Récemment, quatre articles se sont intéressés à des mesures de distance ou de similarité de graphes basées sur des appariements multivoques de sommets :

- Ambauen *et al.* [AFB03] proposent une distance d'édition de graphe étendue pour la comparaison d'images sur-segmentées autorisant des opérations de fusion et d'éclatement de sommets ;
- Boeres *et al.* [BRB04] et Deruyver *et al.* [DHLJ05] proposent des mesures de similarité de graphes basées sur un appariement fonctionnel non-injectif afin de comparer une image sur-segmentée à son modèle schématique ;
- Champin et Solnon [CS03] proposent une mesure de similarité de graphes basée sur un appariement multivoque des sommets des graphes afin de comparer des objets de conception décrits à différents niveaux de granularité.

Cette sous-section présente ces problèmes de recherche d'un meilleur appariement multivoque entre deux graphes.

Distance d'édition de graphe étendue

Dans [AFB03], Ambauen *et al.* s'intéressent à la définition d'une distance entre des images segmentées représentées par des graphes dans lesquels un arc relie deux sommets si les deux régions correspondantes sont adjacentes. Les auteurs soulignent que le problème de cette approche est que la segmentation des images en régions est une tâche difficile et que les images sont bien souvent sur ou sous-segmentées, c.-à-d. que le découpage automatique d'une image en régions "sémantiquement" cohérentes génère trop ou pas suffisamment de régions. Par conséquent, le sommet d'un graphe correspondant à une région d'une image sous-segmentée peut correspondre à plusieurs sommets d'un graphe correspondant à des régions d'une image sur-segmentée.

La figure 2.10 reprend l'exemple proposé par [AFB03] pour montrer (sous forme schématique) l'intérêt d'ajouter les opérations de fusion et d'éclatement de sommets dans le cadre de la comparaison d'images représentées par des graphes. Lors du processus de segmentation, la jambe gauche du personnage représenté à gauche a été sur-segmentée alors que celle du personnage de droite ne l'a pas été. De la même façon, le ventre du personnage de droite a été coupé en deux régions alors que ça n'est pas le cas pour le personnage de gauche. Par conséquent, si on utilise la distance d'édition de graphe (non-étendue) pour comparer les graphes obtenus à partir de ces

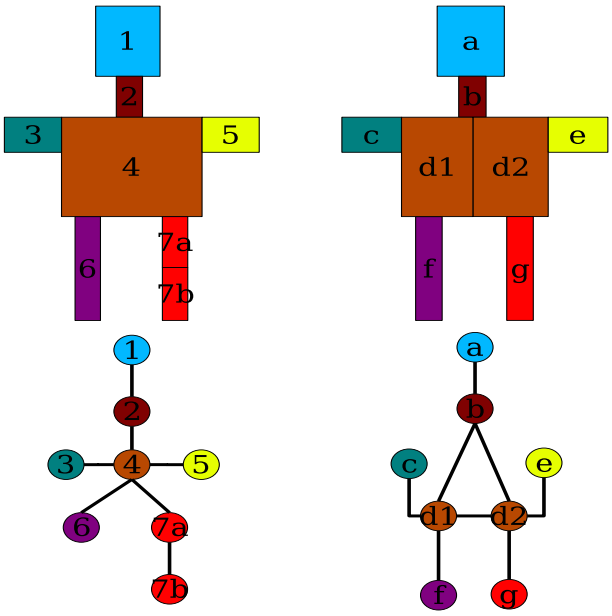


FIG. 2.10 – Exemple (issu de [AFB03]) de l'utilité de la distance d'édition de graphes étendue pour la comparaison d'images sur-segmentées. En haut, les images segmentées, en bas, les graphes correspondant à ces images.

segmentations, il faudra un grand nombre d'opérations d'édition pour rendre les deux graphes identiques et la distance entre ces deux images pourtant très similaires sera élevée.

Pour prendre en compte ces problèmes de sur et de sous-segmentation des images, les auteurs proposent une distance d'édition de graphes étendue permettant de mettre en correspondance un sommet d'un graphe à plusieurs sommets d'un autre graphe. Comme la distance d'édition de graphes, la distance d'édition de graphes étendue évalue la distance entre deux graphes en fonction du coût de l'ensemble des opérations nécessaires à rendre le premier graphe isomorphe au second graphe. En plus des 6 opérations de suppression, substitution et insertion de sommets et d'arcs, deux nouvelles opérations sur les graphes sont introduites pour prendre en compte les problèmes de segmentation : la fusion de sommets et l'éclatement de sommets.

Notons que comme un sommet peut être apparié à plusieurs sommets, l'appariement entre les deux graphes ne peut plus (comme pour la distance d'édition de graphe classique) être présenté comme une fonction injective des sommets d'un graphe dans les sommets de l'autre graphe. Le problème est donc formalisé comme la recherche d'une séquence d'opérations d'édition. La distance d'édition de graphes étendue est alors définie comme le coût de la séquence d'opérations d'édition la moins coûteuse permettant de rendre le premier graphe isomorphe au second graphe.

Si on utilise la distance d'édition de graphes étendue pour comparer les graphes de la figure 2.10, seules deux opérations d'édition (à l'exception des substitutions identiques) sont nécessaires : les fusions des sommets d1 et d2 et des sommets 7a et 7b. Sans pouvoir fusionner les sommets, il aurait fallu au moins 9 opérations, par exemple, supprimer les arcs $(3, 4)$, $(4, 6)$ et $(7a, 7b)$, supprimer le sommet $7b$, insérer un sommet $4a$ et ajouter les arcs $(3, 4a)$, $(2, 4a)$, $(4, 4a)$ et $(4a, 6)$.

Comparaison d'images sur-segmentées à des modèles

Dans [BRB04], Boeres *et al.* s'intéressent au problème de la comparaison d'une image réelle d'un cerveau au modèle d'une image d'un cerveau. Chacune des deux images est segmentée en régions puis modélisée par un graphe où chaque sommet correspond à une région de l'image et chaque arc correspond à l'adjacence des deux régions correspondant aux extrémités de l'arc.

Dans cette application, une comparaison des sommets et des arcs deux à deux ne convient pas. En effet, les deux objets à comparer sont de natures différentes et n'ont donc pas la même granularité de description. L'image modèle à un aspect schématique facile à segmenter automatiquement (ou ayant été segmenté à la main par un spécialiste du domaine) alors que l'image réelle du cerveau, généralement bruitée, est difficilement segmentable de façon automatique et sera donc généralement sur-segmentée. Par conséquent, une région du modèle correspond souvent à plusieurs régions de l'image.

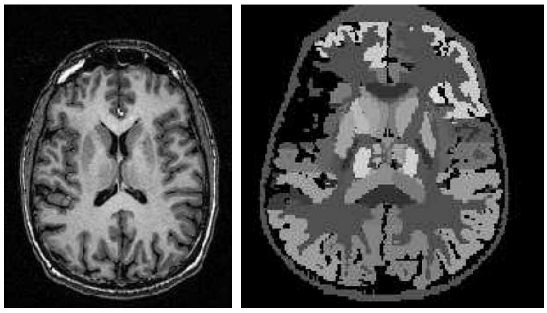


FIG. 2.11 – Exemple d'une image d'un cerveau et d'un modèle du cerveau (issu de [BRB04]). L'image réelle (à gauche) est bruitée et donc difficile a segmentée (elle sera sans doute sur-segmentée) alors que le modèle (à droite) est net, schématique et donc facile à segmenter.

Le problème de la mesure de la similarité des deux images est donc défini comme la recherche d'un meilleur appariement multivoque entre les deux graphes. L'appariement recherché doit respecter de nombreuses contraintes propres à l'application considérée et le problème consiste à trouver un meilleur appariement, c.-à-d. celui qui respecte toutes les contraintes et qui maximise une fonction de similarité calculée à partir de matrices de similarité de sommets et d'arcs.

Plus formellement, deux graphes sont considérés pour ce problème : le graphe modèle $G_1 = (V_1, E_1)$ et le graphe image $G_2 = (V_2, E_2)$ (avec $|V_1| \leq |V_2|$). Une solution de ce problème est un appariement $m \subseteq V_1 \times V_2$ entre les graphes G_1 et G_2 tel que :

1. chaque sommet de G_1 est apparié à un ensemble connecté non-vide de sommets de G_2 (c.-à-d. que $\forall v \in V_1, |m(v)| \geq 1$ et que les sommets de $m(v)$ sont connectés par des arcs de E_2) afin de ne fusionner que des régions adjacentes de l'image ;
2. chaque sommet de G_2 est apparié à exactement un sommet de G_1 (c.-à-d. que $\forall v \in V_2, |m(v)| = 1$) ;
3. certains couples de sommets sont "interdits" et ne peuvent donc pas être appariés les uns aux autres.

Étant donné un appariement qui respecte ces contraintes, une mesure de similarité $sim[Boeres]_m$ est calculée par rapport à des matrices de similarité de sommets et d'arcs $sm_v : V_1 \times V_2 \rightarrow [0, 1]$

et $sm_e : E_1 \times E_2 \rightarrow [0,1]$ comme ceci :

$$sim[Boeres]_m = \frac{\sum_{(u,v)\in m} sm_v(u,v)}{|V_1|.|V_2|} + \frac{\sum_{(u,v)\in(V_1\times V_2)-m} 1 - sm_v(u,v)}{|V_1|.|V_2|} + \frac{\sum_{((u,u'),(v,v'))\in E_1\times E_2 | \{(u,v),(u',v')\}\subseteq m} sm_e((u,u'),(v,v'))}{|E_1|.|E_2|} + \frac{\sum_{((u,u'),(v,v'))\in E_1\times E_2 | \{(u,v),(u',v')\}\not\subseteq m} 1 - sm_e((u,u'),(v,v'))}{|E_1|.|E_2|}$$

Dans [DHLJ05], Deruyver *et al.* utilisent la comparaison de graphes pour obtenir une segmentation sémantique des images. L'image est initialement découpée en (petites) régions par une méthode de segmentation basée sur des critères de bas niveaux. Les régions obtenues sont alors fusionnées tant que l'image peut être appariée au graphe modèle.

Bien que l'objectif et la méthode de résolution proposée soient totalement différents, le problème d'appariement de graphes utilisé par Deruyver *et al.* [DHLJ05] est très similaire à celui utilisé par Boeres *et al.* [BRB04]. Le problème consiste dans les deux cas à trouver un meilleur appariement des sommets d'une image modèle à un ou plusieurs sommets d'une image réelle. Dans les deux cas, l'appariement recherché doit respecter des contraintes entre les nœuds fusionnés et entre les nœuds non fusionnés.

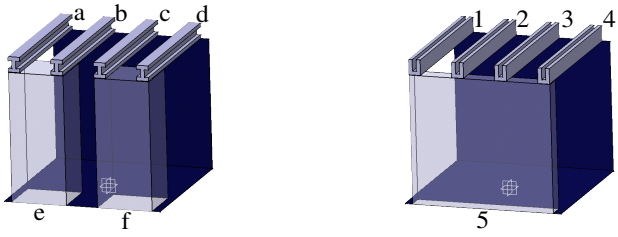
Raisonnement à partir de cas en CAO

Dans [CS03], Champin et Solnon s'intéressent à une application de raisonnement à partir de cas (RàPC) pour des objets de conception par ordinateur [Cha02]. Dans cette application, les objets sont représentés par des graphes. Chaque sommet du graphe correspond à la composante d'un objet (par exemple, un mur, une poutre...) et les arcs correspondent aux relations binaires existant entre ces composantes (par exemple des relations topologiques telles que "a côté de", "sur"...). Le paradigme du raisonnement à partir ce cas s'appuie sur le fait que des problèmes similaires admettent des solutions similaires. Dans cette application il est donc nécessaire de retrouver parmi une base de problèmes (appelés cas), le problème le plus similaire à celui à résoudre, c.-à-d., dans ce cas particulier, l'objet de conception le plus similaire à un objet de conception existant.

Chaque composant et chaque relation peut avoir plusieurs caractéristiques. Par exemple, une poutre peut être caractérisée d'une part par le fait d'être une "poutre" et d'autre part par le fait d'avoir une forme en "U". Par conséquent, les objets sont modélisés par des graphes multi-étiquetés où chaque sommet et chaque relation sont caractérisés par un ensemble non vide d'étiquettes (voir le chapitre 1 pour la définition d'un graphe multi-étiqueté).

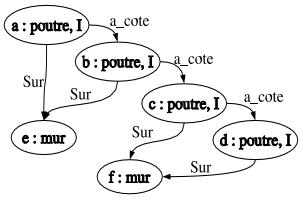
Par exemple, étant donnés l'ensemble d'étiquettes de sommets $L_V = \{poutre, mur, I, U\}$ et l'ensemble d'étiquettes d'arcs $L_E = \{sur, a\_cote\}$, les objets de conception de la figure 2.12 peuvent être représentés par les graphes de la figure 2.13.

L'exemple de la figure 2.12 montre que pour cette application, la composante d'un objet peut jouer le même rôle que plusieurs composantes d'un autre objet. Le mur de l'objet de droite joue le même rôle que les deux murs de l'objet de gauche. Par conséquent, pour comparer les graphes représentant ces deux objets, il est nécessaire de s'appuyer sur un appariement multivoque de

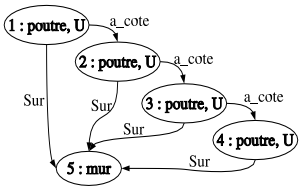


Objet 1 Objet 2

FIG. 2.12 – Deux objets de conception similaires.



$$
\begin{aligned}
G_1 = \langle \quad V_1 &= \{ \quad a, b, c, d, e, f \,\} \\
r_{V_1} &= \{ \quad (a, poutre), (b, poutre), (c, poutre), (d, poutre), \\
&\qquad (a, I), (b, I), (c, I), (d, I), \\
&\qquad (e, mur), (f, mur) \,\} \\
r_{E_1} &= \{ \quad (a, b, a\_cote), (b, c, a\_cote), (c, d, a\_cote), \\
&\qquad (a, e, sur), (b, e, sur), (c, f, sur), (d, f, sur) \,\} \,\rangle
\end{aligned}
$$



$$
\begin{aligned}
G_2 = \langle \quad V_2 &= \{ \quad 1, 2, 3, 4, 5 \,\} \\
r_{V_2} &= \{ \quad (1, poutre), (2, poutre), (3, poutre), (4, poutre), \\
&\qquad (1, U), (2, U), (3, U), (4, U), \\
&\qquad (5, mur) \,\} \\
r_{E_2} &= \{ \quad (1, 2, a\_cote), (2, 3, a\_cote), (3, 4, a\_cote), \\
&\qquad (1, 5, sur), (2, 5, sur), (3, 5, sur), (4, 5, sur) \,\} \,\rangle
\end{aligned}
$$

FIG. 2.13 – Les deux graphes permettant de représenter les deux objets de conception de la figure 2.12 : à gauche la représentation graphique des graphes multi-étiquetés, à droite leur représentation relationnelle

leurs sommets : chaque sommet pourra être mis en correspondance avec un ou plusieurs sommets de l'autre graphe.

Étant donné un appariement multivoque $m \subseteq V \times V'$, la similarité dépend de l'ensemble des étiquettes de sommets et d'arcs communes aux deux graphes par rapport à m. Cet ensemble contient toutes les étiquettes des sommets (resp. arcs) de G et de G' appariés dans m à au moins un sommet (resp. arc) ayant la même étiquette. Plus formellement, l'ensemble $G \sqcap_m G'$ des étiquettes communes à $G = (V, r_V, r_E)$ et $G' = (V', r'_V, r'_E)$ par rapport à l'appariement m est défini par :

$$
\begin{aligned}
G \sqcap_m G' \quad \doteq \quad & \{(v, l) \in r_V |\ \exists (v, v') \in m,\ (v', l) \in r_{V'}\} \\
\cup \quad & \{(v', l) \in r_{V'} |\ \exists (v, v') \in m,\ (v, l) \in r_V\} \\
\cup \quad & \{(v_i, v_j, l) \in r_E |\ \exists (v_i, v'_i) \in m, \exists (v_j, v'_j) \in m, (v'_i, v'_j, l) \in r_{E'}\} \\
\cup \quad & \{(v'_i, v'_j, l) \in r_{E'} |\ \exists (v_i, v'_i) \in m, \exists (v_j, v'_j) \in m, (v_i, v_j, l) \in r_E\}
\end{aligned}
$$

Exemple : Considérons les deux graphes étiquetés de la figure 2.13 et les deux appariement

suivants :

$$\begin{aligned} m_A = &\ \{(a,1),(b,2),(c,3),(d,4),(e,5),(f,5)\} \\ m_B = &\ \{(a,1),(b,2),(a,3),(b,4)),(e,5)\} \end{aligned}$$

Les étiquettes communes à ces deux graphes par rapport aux appariements m_A et m_B proposés ci-avant sont :

$$\begin{aligned} G \sqcap_{m_A} G' &= r_V \cup r_E \cup r'_V \cup r'_E \\ &\quad - \{\,(a,I),(b,I),(c,I),(d,I),(1,U),(2,U),(3,U),(4,U)\,\} \\ G \sqcap_{m_B} G' &= \{(a,poutre),(b,poutre),(e,mur), \\ &\quad (1,poutre),(2,poutre),(3,poutre),(4,poutre),(5,mur), \\ &\quad (a,b,a\_cote),(1,2,a\_cote),(3,4,a\_cote),(a,e,sur), \\ &\quad (b,e,sur),(1,5,sur),(2,5,sur),(3,5,sur),(4,5,sur)\,\} \end{aligned}$$

Étant donné un appariement multivoque m, la similarité dépend aussi de l'ensemble des sommets éclatés (les *splits*), c.-à-d. l'ensemble des sommets appariés à plus d'un sommet. Chaque sommet éclaté v est associé à l'ensemble s_v des sommets auquel il est apparié :

$$\begin{aligned} splits(m) = &\ \{(v,s_v) \quad | \ v \in V, \quad s_v = \{v' \in V' | (v,v') \in m\}, \quad |s_v| \geq 2\} \\ \cup &\ \{(v',s_{v'}) \quad | \ v' \in V', \quad s_{v'} = \{v \in V | (v,v') \in m\}, \quad |s_{v'}| \geq 2\} \end{aligned}$$

Par exemple, les appariements m_A et m_B de l'exemple précédent génèrent les splits suivants : $splits(m_A) = \{\ (5,\{e,f\})\ \}$ et $splits(m_B) = \{\ (a,\{1,3\}),\ (b,\{2,4\})\ \}$.

La **similarité** de deux graphes G et G' par rapport à un appariement m est alors définie par :

$$sim_m(G,G') = \frac{f(G \sqcap_m G') - g(splits(m))}{f(r_V \cup r_E \cup r'_V \cup r'_E)} \tag{2.1}$$

où f et g sont deux fonctions dépendantes de l'application considérée. Par exemple, si f est la fonction cardinalité et g la fonction nulle, la similarité est alors proportionnelle au nombre d'étiquettes communes aux deux graphes par rapport au nombre total de caractéristiques. Si g est une fonction de cardinalité, alors la similarité décroîtra proportionnellement au nombre de sommets éclatés.

Finalement, la similarité $sim(G,G')$ de deux graphes G et G' est définie comme la plus grande similarité possible, c.-à-d. celle obtenue par le meilleur appariement :

$$sim(G,G') = \max_{m \subseteq V \times V'} \frac{f(G \sqcap_m G') - g(splits(m))}{f(r_V \cup r_E \cup r'_V \cup r'_E)} \tag{2.2}$$

Prenons l'exemple de la figure 2.13 et supposons que les fonctions f et g soient des fonctions de cardinalité. Le meilleur appariement entre ces deux graphes consiste à apparier les poutres entre elles et les deux murs de l'objet de gauche avec le mur de l'objet de droite : c'est donc l'appariement m_A. En effet, cet appariement permet de retrouver 25 étiquettes de sommets et d'arcs sur les 33 présentes en ne générant qu'un seul éclatement de sommet. La similarité entre ces deux graphes par rapport à cet appariement est donc de 24/33. Comme il n'existe pas de meilleur appariement de ces deux graphes, la similarité de ces deux graphes est donc de 24/33.

2.3 Discussion

Bien qu'elles aient été généralement initialement définies dans un autre formalisme, nous avons vu que toutes les mesures de distance/similarité de graphes présentées dans la section précédente reposent sur la recherche d'un "meilleur" appariement de deux graphes. Néanmoins, ces mesures diffèrent sur les contraintes et les préférences qu'elles imposent à l'appariement recherché.

2.3.1 Structure des graphes

Les mesures de distance/similarité de graphes existantes n'ont pas la même tolérance aux variations existantes dans la structure des graphes à comparer. Par exemple, le problème de l'isomorphisme de graphes (resp. de sous-graphe) se base sur un appariement mettant en évidence le fait qu'un graphe a une structure rigoureusement identique à celle d'un autre graphe (resp. d'une partie d'un autre graphe). Retrouver tous les sommets et tous les arcs d'un ou des deux graphes comparés est une contrainte dure et l'appariement recherché est une application injective (resp. bijective dans le cas de l'isomorphisme de graphe) devant retrouver tous les arcs d'un graphe (resp. des deux graphes).

A contrario, d'autres mesures, comme par exemple celles basées sur la recherche d'un plus grand sous-graphe commun, sont plus souples vis-à-vis de la structure. Les graphes comparés ne sont pas nécessairement identiques dans leur *globalité*. L'appariement recherché met en évidence les *parties* de graphes identiques : seuls les sous-ensembles des sommets et des arcs des deux graphes appariés doivent avoir une structure identique. L'appariement n'est donc pas nécessairement une application mais il est nécessairement univalent et fait en sorte de maximiser le nombre de sommets et/ou d'arcs retrouvés dans les deux graphes.

Les mesures les plus souples vis-à-vis de la structure des graphes reposent sur des appariements multivoques [CS03, AFB03]. Même les parties des graphes mis en correspondance n'ont plus nécessairement une structure identique : plusieurs sommets (resp. arcs) peuvent être mis en correspondance avec un même sommet (resp. arc). Afin de maximiser tout de même la similarité structurelle, les mesures basées sur des appariements multivoques tentent de limiter ces mises en correspondance multiples en les pondérant (par un coût de fusion et d'éclatement pour la distance d'édition de graphes étendue ou par la fonction g pour la mesure de Champin et Sorlin).

2.3.2 Similarité des sommets et des arcs appariés

Nous avons vu que les graphes manipulés par les mesures de distance/similarité que nous avons présentées étaient souvent étiquetés ou multi-étiquetés. Les contraintes et les préférences posées sur l'appariement recherché peuvent également porter sur ces étiquettes.

Certaines mesures imposent aux appariements de n'apparier que des éléments rigoureusement identiques. C'est notamment le cas de l'isomorphisme de graphe ou de sous-graphe ou des mesures basées sur la recherche d'un plus grand sous-graphe commun aux deux graphes.

D'autres mesures, comme par exemple la distance d'édition de graphes, autorise les appariements de sommets et d'arcs n'ayant pas la même étiquette (c'est une substitution non-identique). Afin de minimiser tout de même les différences entre les graphes, un coût est associé à ces substitutions d'étiquettes.

Lorsque les éléments sont multi-étiquetés (cas de la mesure de Champin et Solnon), l'appariement recherché tente de maximiser le nombre d'étiquettes retrouvées dans les deux graphes.

De façon plus générale, comme dans le cas de la similarité de [BRB04], la qualité d'un appariement peut être évaluée à l'aide de matrices de similarité des sommets et des arcs des deux graphes. L'appariement recherché doit donc maximiser la similarité des éléments appariés entre eux. Les sommets et les arcs peuvent également être caractérisés par tout autre type de valeurs (nombres réels, vecteurs...), l'appariement recherché devant alors minimiser une fonction de comparaison des valeurs des éléments mis en correspondance.

Dans le cas des appariements multivoques, la mesure doit être capable de comparer un ensemble de sommets (resp. d'arcs) à un autre ensemble de sommets (resp. d'arcs). Dans la mesure de Champin et Solnon où il faut maximiser le nombre d'étiquettes de sommets et d'arcs retrouvées, cette comparaison se fait en faisant l'intersection des unions des étiquettes des éléments fusionnés.

2.3.3 La mesure de Champin et Solnon

Sorlin *et al.* montrent dans [SS05a, SS05b] que la mesure de Champin et Solnon [CS03] est générique. En effet, cette mesure est paramétrée par les fonctions de similarité f et g et toutes les autres mesures de similarité proposées dans la littérature peuvent être vues comme un cas particulier de cette mesure en instanciant ces fonctions de façon adéquate.

Cependant, bien qu'elle soit générique, la mesure de Champin et Solnon [CS03] est difficile à utiliser dans certains contextes. En effet, cette mesure s'applique à des graphes multi-étiquetés et la similarité de deux graphes est évaluée par rapport à l'ensemble des étiquettes qu'un "meilleur" appariement des deux graphes permet de retrouver. Or, certaines contraintes ou préférences sur l'appariement recherché ne peuvent être exprimées "directement" en termes d'étiquettes retrouvées. Il est donc souvent nécessaire d'introduire des étiquettes de sommets "artificielles" permettant de "recalculer" l'appariement réalisé dans la fonction f et de déléguer entièrement le calcul de la qualité d'un appariement à cette fonction.

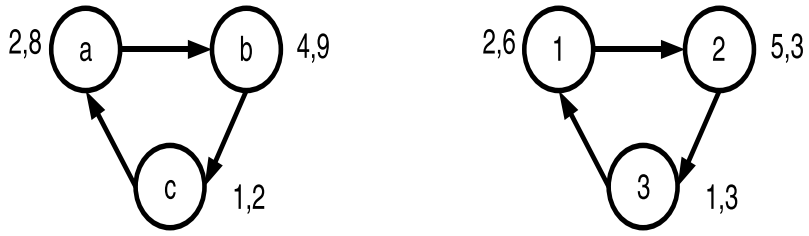


FIG. 2.14 – Deux graphes G et G' dont les sommets sont valués. On recherche un appariement univoque entre G et G' qui minimise l'écart entre les valeurs des sommets appariés.

Exemple. Prenons les deux graphes G et G' de la figure 2.14. Des nombres réels sont associés à chaque sommet de ces graphes. Supposons que l'on veuille trouver un appariement univoque de ces deux graphes qui minimise l'écart entre les valeurs des sommets appariés. Cette contrainte ne peut pas s'exprimer directement en termes d'étiquettes retrouvées. En effet, si les sommets sont étiquetés par leur valeur, seules les étiquettes des sommets appariés ayant une valeur rigoureusement identique sont retrouvées par un appariement et il n'est pas possible de mesurer l'écart entre les valeurs de ces sommets. Il faut donc ajouter des étiquettes artificielles permettant de retrouver l'appariement dans la fonction f.

Les étiquettes de sommet vv' que nous proposons d'ajouter sont retrouvées par un appariement m si et seulement si $(v, v') \in m$. De façon plus formelle, étant donnés les deux graphes

$G = (V, E)$ et $G' = (V', E')$, il faut ajouter à chaque sommet v de V (resp. $v' \in V'$), l'ensemble des étiquettes vv' telles que $v' \in V'$ (resp. $v \in V$).

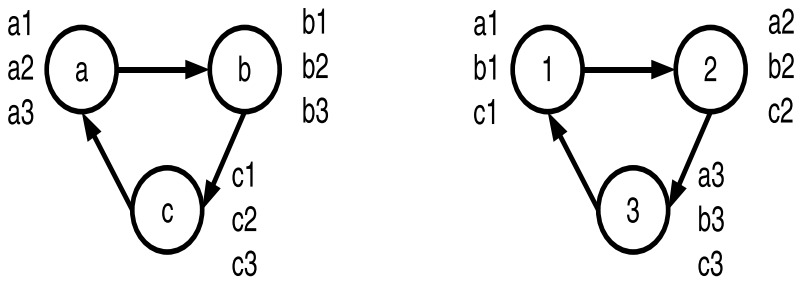


FIG. 2.15 – Des étiquettes "artificielles" doivent être ajoutées aux sommets des graphes G et G' afin de pouvoir retrouver l'appariement réalisé dans la fonction f.

La figure 2.15 montre comment il faut étiqueter les sommets des graphes G et G' de la figure 2.14 afin de pouvoir retrouver l'appariement réalisé dans la fonction f. Il est facile de montrer qu'une étiquette (v, vv') appartient à l'ensemble $G \sqcap_m G'$ si et seulement si $(v, v') \in m$.

Une fois les graphes artificiellement étiquetés, il est possible de définir la fonction f afin de minimiser l'écart entre les valeurs des sommets appariés [4].

$$f(S) = \sum_{(v,vv') \in S | v \in V} val(v) + val(v') - |val(v) - val(v')|$$

La fonction g doit être définie de façon à interdire les appariements non-bijectifs :

$$\begin{aligned} g(\emptyset) &= 0 \\ g(S) &= +\infty \text{ si } S \neq \emptyset \end{aligned}$$

Résoudre ce problème d'appariement avec la mesure de Champin et Solnon est possible mais fastidieux à mettre en œuvre : il est nécessaire d'utiliser de nombreuses étiquettes "artificielles" à seule fin de pouvoir calculer l'écart des valeurs de deux sommets appariés entre eux. D'autres contraintes "classiques" sont également difficiles à exprimer : par exemple, imposer un appariement "bijectif" (et non plus simplement univoque) nécessite également l'introduction d'étiquettes de sommets artificielles.

Cette difficulté réside dans le fait que la qualité d'un appariement est évaluée en fonction des étiquettes de sommets et d'arcs que cet appariement permet de retrouver. Cette approche a deux conséquences néfastes :

- La comparaison de deux sommets (ou arcs) est réalisée par rapport à l'intersection de leurs ensembles d'étiquettes. Deux étiquettes "similaires" mais non-identiques ne peuvent donc pas "directement" être prises en compte dans la mesure de distance/similarité de l'appariement de ces deux sommets (voir l'exemple ci-dessus) ;
- Lorsque deux sommets (resp. arcs) sont fusionnés, l'agrégation de leurs ensembles d'étiquettes se fait par l'opérateur ensembliste d'union et il n'est pas possible d'utiliser "directement" un autre opérateur. La fusion de deux sommets ayant une étiquette "Taille=1" ne permet pas de retrouver l'étiquette "Taille=2" du sommet auxquels ils sont appariés.

[4] Nous supposerons que les valeurs associées aux sommets sont strictement supérieures à 0.

2.3.4 Synthèse

La distance ou la similarité de deux graphes est un problème récurrent en recherche d'informations et de nombreuses mesures de distance ou de similarité de graphes ont été proposées dans la littérature. Cependant, dans le contexte général de la comparaison de deux graphes, aucune des mesures existantes ne nous a paru satisfaisante. Chaque mesure a été définie dans un contexte particulier et ne peut pas être réutilisée facilement dans un autre contexte. Seule la mesure de Champin et Solnon [CS03] est générique [SS05a, SS05b] dans le sens où elle permet d'exprimer les autres mesures de similarité de graphes. Néanmoins, cette mesure est difficile à utiliser dans certains contextes : elle compare des graphes multi-étiquetés et la comparaison des sommets et des arcs entre eux est limitée à une opération d'intersection de l'union de leurs ensembles d'étiquettes (limitant l'expressivité de cette mesure). Il nous est donc apparu nécessaire de proposer une nouvelle mesure générique de la distance de deux graphes.

La mesure que nous proposons est capable d'approximer à la fois la structure des deux graphes et les propriétés (les étiquettes, les valeurs numériques ou tout autre type d'objets) portées sur les sommets et les arcs de ces graphes : elle est paramétrée par des fonctions de distance de sommets et d'arcs exprimant les préférences et les contraintes sur l'appariement recherché. Ce paramétrage rend notre mesure générique : elle permet de calculer toutes les mesures de similarité et de distance de graphes présentées en première partie de ce document (nous le verrons à la section 3.2 de ce mémoire).

Notre mesure repose sur la recherche d'un meilleur appariement multivoque des sommets des deux graphes à comparer. Néanmoins, des contraintes (introduites dans des fonctions de distances de sommets et d'arcs) peuvent être ajoutées afin de limiter la comparaison des graphes à certains types d'appariements. Contrairement à la mesure de Champin et Solnon [CS03], notre mesure de distance de graphes ne présume pas de la façon de comparer et de fusionner les étiquettes ou les valeurs portées par les sommets et les arcs des graphes.

Notre mesure permet d'exprimer simplement et de façon homogène toutes les façons de comparer deux graphes. Homogénéiser les définitions des mesures de distance de graphes présente deux avantages : comparer les mesures entre elles devient plus simple et un algorithme capable de calculer notre mesure générique permet également de calculer toutes les mesures de distances de graphes existantes.

Nous présentons notre nouvelle mesure de la distance de deux graphes dans le chapitre suivant. Nous montrons ensuite que cette mesure est générique dans le sens où elle permet de modéliser de nombreux problèmes d'appariement de graphes et donc de calculer les mesures de distance ou de similarité de graphes existantes.

3

Une nouvelle mesure générique

3.1 Définition de la mesure

Dans cette section, nous introduisons une nouvelle mesure de distance entre deux graphes. Cette distance est générique dans le sens où elle est paramétrée par des fonctions de distance de sommets et d'arcs. Ces fonctions permettent d'introduire des connaissances propres au domaine d'application de la mesure. Notre mesure permet la comparaison de graphes où les sommets et les arcs possèdent un ou plusieurs attributs (tels que des étiquettes, des valeurs numériques...).

3.1.1 Fonctions de distance de sommets et d'arcs

La première étape lorsqu'on souhaite calculer la distance entre deux graphes est d'apparier leurs sommets afin d'identifier leurs points communs. Nous considérons ici des appariements multivoques, c.-à-d. que chaque sommet d'un graphe peut être apparié à un ensemble (éventuellement vide) de sommets de l'autre graphe.

Étant donné un appariement m, il est nécessaire de savoir pour chaque sommet et chaque arc des deux graphes à quel point les propriétés du sommet ou de l'arc sont retrouvées par l'appariement m. Dès lors, nous supposons l'existence d'une fonction δ_v (resp. δ_e) de distance de sommets (resp. d'arcs) retournant pour chaque sommet v (resp. arc (u, v)) des deux graphes et chaque ensemble de sommets s_v (resp. d'arcs s_e) de l'autre graphe un nombre réel dans l'intervalle $[0, +\infty[$ exprimant la distance entre le sommet v (resp. l'arc (u, v)) et l'ensemble s_v (resp. s_e). De façon plus formelle, nous supposons l'existence des deux applications suivantes :

$$\begin{aligned} \delta_v &: (V, \wp(V')) \cup (V', \wp(V)) \to \mathbb{R}^+ \\ \delta_e &: (E, \wp(E')) \cup (E', \wp(E)) \to \mathbb{R}^+ \end{aligned}$$

Généralement, la distance sera égale à $+\infty$ si le sommet v (resp. l'arc (u, v)) n'est pas "comparable" avec l'ensemble de sommets s_v (resp. d'arcs s_e), c.-à-d. quand cet appariement viole une contrainte dure du problème. *A contrario*, la distance sera égale à 0 quand toutes les propriétés de v (resp. de (u, v)) sont retrouvées par l'ensemble s_v (resp. s_e).

Les fonctions δ_v et δ_e expriment les préférences locales sur la façon dont les sommets et les arcs doivent être appariés. Ces fonctions dépendent de l'application considérée et sont utilisées pour exprimer les connaissances de similarité propres au domaine ainsi que les contraintes que l'appariement doit respecter.

Par exemple, si l'appariement recherché est un appariement univoque (c.-à-d. que chaque sommet doit être apparié à au plus un sommet de l'autre graphe) qui préserve le plus grand

nombre de sommets et d'arcs, les fonctions δ_v et δ_e doivent être définies comme ceci :

$$\begin{aligned}
\forall v \in V \cup V', \forall s_v \in \wp(V) \cup \wp(V'), \delta_v(v, s_v) &= 1 \text{ si } s_v = \emptyset \\
&= 0 \text{ si } |s_v| = 1 \\
&= +\infty \text{ sinon} \\
\forall (u,v) \in E \cup E', \forall s_e \in \wp(E) \cup \wp(E'), \delta_e((u,v), s_e) &= 1 \text{ si } s_e = \emptyset \\
&= 0 \text{ si } |s_e| = 1 \\
&= +\infty \text{ sinon}
\end{aligned}$$

Chacune de ces fonctions retourne $+\infty$ si un élément est apparié à plus d'un autre élément (ce qui violerait la contrainte d'appariement univoque), 0 s'il est apparié à exactement un élément, 1 s'il est apparié à aucun élément (afin d'"encourager" les sommets et les arcs à être appariés).

3.1.2 Distance de graphes

Étant donnés un appariement $m \subseteq V \times V'$ de deux graphes $G = (V, E)$ et $G' = (V', E')$ et les deux fonctions de distance δ_v et δ_e, la distance entre les graphes G et G' par rapport à l'appariement m dépend de la distance entre chaque sommet (resp. arc) et l'ensemble des sommets (resp. arcs) auxquels il est apparié :

$$\begin{aligned}
\delta_m(G, G') &= \otimes(S_v, S_e) \\
S_v &= \{(v, \delta_v(v, m(v)))|v \in V \cup V'\} \\
S_e &= \{((u,v), \delta_e((u,v), m(u,v)))|(u,v) \in E \cup E'\}
\end{aligned} \tag{3.1}$$

où $\otimes : \wp(V \cup V' \times \mathbb{R}^+) \times \wp(E \cup E' \times \mathbb{R}^+) \rightarrow \mathbb{R}^+$ est une fonction dépendant du domaine d'application. Cette fonction est introduite pour agréger les calculs de distance de sommets et d'arcs. Elle permet d'exprimer les préférences globales sur les distances entre les sommets et les arcs des deux graphes. La fonction $\otimes$ est définie de façon à ce que la distance minimale entre deux graphes par rapport à un appariement soit égale à 0. La distance par rapport à un appariement m de deux graphes G et G' est égale à $+\infty$ si et seulement si l'appariement m n'est pas un appariement valide par rapport à l'application considérée, c.-à-d. quand m viole une contrainte dure du problème.

Le plus souvent, la fonction $\otimes$ est définie comme une somme ou une somme pondérée de la distance de chacun des arcs et des sommets. Cependant, afin d'exprimer des distances plus sophistiquées, nous ne nous restreignons pas à ce cas particulier. Par exemple, la fonction $\otimes$ peut être choisie de façon à rendre la distance entre deux graphes proportionnelle au nombre de sommets ayant au moins un arc entrant ou sortant avec une distance supérieure à un certain seuil.

La formule (3.1) définit la distance entre deux graphes par rapport à un appariement m entre les sommets de ces graphes. Nous définissons la distance entre deux graphes G et G' comme la distance induite par le meilleur appariement, c.-à-d. celui qui induit une distance minimale :

$$\delta(G, G') = \min_{m \subseteq V \times V'} \delta_m(G, G') \tag{3.2}$$

Finalement, étant donnés deux graphes G et G', une mesure de distance entre G et G' est définie comme un triplet $\delta =< \delta_v, \delta_e, \otimes >$ où δ_v est la fonction de distance de sommets, δ_e est la

fonction de distance d'arcs et $\otimes$ est la fonction agrégeant les distances des sommets et des arcs des deux graphes.

Notons que le terme "distance" est utilisé ici dans son sens commun : la distance entre deux graphes est faible lorsque les deux graphes partagent de nombreuses propriétés et est égale à 0 (le minimum) quand un appariement parfait des deux graphes peut être trouvé (par rapport à l'application considérée). Dans le cas général, notre mesure de distance n'a pas les propriétés mathématiques d'une distance "classique" et n'est pas une métrique. Par exemple, la distance entre deux graphes peut être infinie ; elle peut ne pas respecter l'inégalité triangulaire et ne pas être symétrique et la distance entre un graphe et lui même peut ne pas être égale à 0. Cependant, selon les définitions des fonctions δ_v, δ_e et $\otimes$, notre distance peut devenir une métrique.

3.1.3 Similarité de graphes

Nous avons choisi de définir une distance entre deux graphes. Cependant, distance et similarité sont deux concepts duaux et notre mesure de distance peut être utilisée pour définir une mesure de similarité entre deux graphes.

Par exemple, dans de nombreuses applications, la distance entre deux graphes G et G' est toujours inférieure ou égale à la somme des distances entre chacun des deux graphes et le graphe vide $G_\emptyset$ ($G_\emptyset = (\emptyset, \emptyset)$). Par conséquent, cette propriété peut être utilisée pour normaliser la distance entre deux graphes et ainsi définir la mesure de similarité suivante :

$$sim(G, G') = 1 - \frac{\delta(G, G')}{\delta(G, G_\emptyset) + \delta(G', G_\emptyset)}$$

3.1.4 Analyse et critique

Notre distance de graphe calcule la distance entre chaque élément (sommets et arcs) des graphes et l'ensemble des éléments auquel il est apparié puis agrège ces distances avec la fonction $\otimes$ pour en déduire la distance entre les deux graphes. Cette façon de faire présuppose que les distances entre les éléments des graphes sont indépendantes les unes des autres et peuvent donc être évaluées localement. Par conséquent, notre mesure ne permet pas d'exprimer *simplement* des contraintes ou des préférences portant sur plusieurs éléments à la fois. Par exemple, il n'est pas possible d'exprimer simplement la contrainte : "ces deux sommets doivent être appariés au même nombre de sommets".

Décomposer la distance entre deux graphes en une agrégation de la distance de chacun de leurs éléments peut donc paraître d'une part arbitraire et d'autre part une limitation de notre modèle. Néanmoins, nous justifions notre choix par les trois raisons suivantes :

- Nous verrons dans le chapitre suivant que la majorité des problèmes d'appariement peuvent être exprimés simplement avec notre mesure. Excepté pour la distance d'édition de graphes étendue, toutes les contraintes et les préférences sur les appariements recherchés peuvent être exprimés de façon locale et l'opérateur d'agrégation des distances est tout simplement une somme des distances ;
- Cette façon de faire permet d'exprimer simplement les connaissances de similarité d'un domaine d'application. Le problème de la distance de deux graphes n'a pas à être exprimé dans sa globalité : l'utilisateur définit juste ses préférences pour chaque composant et chaque relation entre ces composants ;
- Nous verrons que dans tous les cas, il est toujours possible de définir des fonctions de distances des sommets et des arcs de façon à ce que les arguments de la fonction $\otimes$ contiennent l'information nécessaire pour reconstituer l'appariement considéré. Par conséquent, toute

distance de deux graphes dépendant d'un appariement des deux graphes pourra tout de même être exprimée dans notre formalisme.

3.2 Équivalence avec d'autres mesures

Dans ce chapitre, nous montrons que notre mesure de la distance entre deux graphes peut être utilisée pour résoudre de nombreux problèmes d'appariement de graphes et peut donc être rendue équivalente aux mesures de distance et de similarité de graphes présentées au chapitre 1.

Pour chaque problème présenté auparavant, nous donnons une formulation de ce problème en terme de recherche d'un appariement (ou d'un "meilleur" appariement) et nous proposons une définition de notre mesure de distance de graphe telle que le calcul de cette mesure permette la résolution du problème.

Dans ce chapitre, la fonction $\otimes$ de nos mesures de distance de graphes est toujours définie comme la fonction $\otimes_{\sum}$ faisant la somme des distances de chacun des éléments des deux graphes. De façon plus formelle, la fonction $\otimes_{\sum} : \wp(V \cup V' \times [0, +\infty[) \times (E \cup E' \times [0, +\infty[) \rightarrow [0, +\infty[$ est définie par :

$$\otimes_{\sum}(S_v, S_e) \quad = \quad \sum_{(u,d) \in S_v} d + \sum_{((u,v),d) \in S_e} d$$

3.2.1 Appariements exacts de graphes

Nous montrons dans cette section comment instancier notre distance générique de deux graphes pour résoudre des problèmes d'appariement exact de deux graphes. Dans tous ces problèmes, nous recherchons un appariement univoque des sommets des deux graphes. Par conséquent, les fonctions de distance de sommets et d'arcs sont définies de telle façon qu'un appariement multivoque induise toujours une distance infinie. En outre, ces problèmes étant des problèmes de satisfaction, l'objectif est toujours de trouver un appariement m qui induit une distance nulle (c.-à-d. tel que $\delta_m(G, G') = 0$).

Isomorphisme de graphe

Définition du problème (Rappel). Deux graphes étiquetés $G = (V, E, L_V, \alpha, L_E, \beta)$ et $G' = (V', E', L_V, \alpha', L_E, \beta')$ tels que $|V| = |V'|$ sont isomorphes si et seulement s'il existe un appariement bijectif $m \subseteq V \times V'$ tel que $\forall (u, v) \in V^2, (u, v) \in E \Leftrightarrow (m(u), m(v)) \in E'$ et $\forall u \in V, \alpha(u) = \alpha'(m(u))$ et $\forall (u, v) \in E, \beta((u, v)) = \beta'((m(u), m(v)))$.

Définition de notre mesure. Pour résoudre le problème de l'isomorphisme de deux graphes en utilisant notre mesure de distance, il faut définir les fonctions de distance de sommets et d'arcs de telle façon qu'elles retournent 0 si le sommet où l'arc est apparié à exactement un élément ayant la même étiquette et $+\infty$ sinon (afin d'éviter les appariements non bijectifs). Plus

formellement, nous définissons la distance $\delta^{iso} =< \delta_v^{iso}, \delta_e^{iso}, \otimes_{\sum} >$:

$$\begin{aligned}
\forall v \in V, \forall s_v \subseteq V', \delta_v^{iso}(v, s_v) &= 0 \text{ si } s_v = \{v'\} \wedge \alpha(v) = \alpha'(v') \\
&= +\infty \text{ sinon} \\
\forall (u,v) \in E, \forall s_e \subseteq E', \delta_e^{iso}(u, v, s_e) &= 0 \text{ si } s_e = \{(u',v')\} \wedge \beta((u,v)) = \beta'((u',v')) \\
&= +\infty \text{ sinon} \\
\forall v' \in V', \forall s_v \subseteq V, \delta_v^{iso}(v', s_v) &= 0 \text{ si } s_v = \{v\} \wedge \alpha(v) = \alpha'(v') \\
&= +\infty \text{ sinon} \\
\forall (u',v') \in E', \forall s_e \subseteq E, \delta_e^{iso}(u', v', s_e) &= 0 \text{ si } s_e = \{(u,v)\} \wedge \beta((u,v)) = \beta'((u',v')) \\
&= +\infty \text{ sinon}
\end{aligned}$$

Théorème 1. *Étant donnés deux graphes étiquetés G et G', les deux propriétés suivantes sont équivalentes :*

1. *G et G' sont des graphes isomorphes ;*
2. *la distance $\delta^{iso} =< \delta_v^{iso}, \delta_e^{iso}, \otimes_{\sum} >$ entre G et G' est égale à 0.*

Démonstration. (1) $\Rightarrow$ (2). Par définition de l'isomorphisme de deux graphes, si les deux graphes sont isomorphes, il existe un appariement bijectif $m \subseteq V \times V'$ tel que $(u,v) \in E \Leftrightarrow (m(u), m(v)) \in E'$ et $\forall u \in V, \alpha(u) = \alpha'(m(u))$ et $\forall (u,v) \in E, \beta((u,v)) = \beta'((m(u), m(v)))$. Par conséquent, $\forall v \in V, m(v) = \{v'\} \wedge \alpha(v) = \alpha'(v')$, $\forall v' \in V', m(v') = \{v\} \wedge \alpha(v) = \alpha'(v')$. En outre, $\forall (u,v) \in E, m(u,v) = \{(u',v')\} \wedge \beta((u,v)) = \beta'((u',v'))$ et $\forall (u',v') \in E', m(u',v') = \{(u,v)\} \wedge \beta((u,v)) = \beta'((u',v'))$ (car $\forall (u,v) \in V \times V, (u,v) \in E \Leftrightarrow (m(u), m(v)) \in E'$ et que $\forall (u,v) \in E, \beta((u,v)) = \beta'((m(u), m(v)))$). Donc, étant données les définitions de δ_v^{iso} et de δ_e^{iso}, la distance entre G et G' par rapport à m est égale à 0 et la distance entre G et G' est donc égale à 0.

(2) $\Rightarrow$ (1). Si la distance entre G et G' est égale à 0, alors, étant donnée la définition de δ_v^{iso}, il existe un appariement m tel que $\forall v \in V \cup V', |m(v)| = 1$. Par conséquent, l'appariement m est un appariement bijectif. En outre, si m implique une distance égale à 0, alors, $\forall (u,v) \in E \cup E', |m(u,v)| = 1$. Par conséquent, chaque arc des deux graphes est apparié à exactement un arc de l'autre graphe et $(u,v) \in E \Leftrightarrow (m(u), m(v)) \in E'$. En outre, $\forall v \in V, \alpha(v) = \alpha'(m(v))$ et $\forall (u,v) \in E, \beta((u,v)) = \beta'((u',v'))$. m définit donc un appariement d'isomorphisme entre les deux graphes et G et G' sont isomorphes. □

Isomorphisme de sous-graphe partiel

Définition du problème (Rappel). Étant donnés deux graphes étiquetés $G = (V, E, L_V, \alpha, L_E, \beta)$ et $G' = (V', E', L_V, \alpha, L_E, \beta)$ tels que $|V| \leq |V'|$, le graphe G est un sous-graphe partiel du graphe G' si et seulement s'il existe un appariement injectif $m \subseteq V \times V'$ tel que $\forall (u,v) \in E, (m(u), m(v)) \in E', \forall v \in V, \alpha(v) = \alpha(m(v))$ et $\forall (u,v) \in E, \beta((u,v)) = \beta'((m(u), m(v)))$.

Définition de notre mesure. Pour résoudre le problème de l'isomorphisme de sous-graphe partiel en utilisant notre mesure de distance, il faut définir les fonctions de distance de sommets et d'arcs de telle façon qu'elles retournent 0 si un sommet où un arc de G est apparié à exactement un élément ayant la même étiquette et $+\infty$ sinon (afin d'éviter les appariements non bijectifs et de préserver les sommets et les arcs de G). Les fonctions de distance des sommets et des arcs de G' doivent uniquement éviter les appariements non univoques. Plus formellement, nous définissons la distance $\delta^{psub} =< \delta_v^{psub}, \delta_e^{psub}, \otimes_{\sum} >$:

$$
G \begin{cases} \forall v \in V, \forall s_v \subseteq V', \delta_v^{psub}(v, s_v) &= 0 \text{ si } s_v = \{v'\} \wedge \alpha(v) = \alpha'(v') \\ &= +\infty \text{ sinon} \\ \forall (u,v) \in E, \forall s_e \subseteq E', \delta_e^{psub}(u, v, s_e) &= 0 \text{ si } s_e = \{(u', v')\} \wedge \beta((u,v)) = \beta'((u', v')) \\ &= +\infty \text{ sinon} \end{cases}
$$

$$
G' \begin{cases} \forall v' \in V', \forall s_v \subseteq V, \delta_v^{psub}(v', s_v) &= 0 \text{ si } |s_v| \leq 1 \\ &= +\infty \text{ sinon} \\ \forall (u', v') \in E', \forall s_e \subseteq E, \delta_e^{psub}(u', v', s_e) &= 0 \text{ si } |s_e| \leq 1 \\ &= +\infty \text{ sinon} \end{cases}
$$

Théorème 2. *Étant donnés deux graphes étiquetés G et G', les deux propriétés suivantes sont équivalentes :*

1. *le graphe G est un sous-graphe partiel du graphe G' ;*
2. *la distance $\delta^{psub} =< \delta_v^{psub}, \delta_e^{psub}, \otimes_{\sum} >$ entre G et G' est égale à 0.*

Démonstration. (1) $\Rightarrow$ (2). Par définition, si G est un sous-graphe partiel de G', il existe un appariement injectif $m \subseteq V \times V'$ tel que $\forall (u,v) \in V \times V, (u,v) \in E \Rightarrow (m(u), m(v)) \in E' \wedge \beta((u,v)) = \beta'((m(u), m(v)))$ et $\forall v \in V, \alpha(v) = \alpha'(m(v))$. Par conséquent, $\forall v \in V, m(v) = \{v'\} \wedge \alpha(v) = \alpha'(v')$, $\forall v' \in V', |m(v')| \leq 1$ et $\forall (u', v') \in E', |m(u', v')| \leq 1$ (car m est un appariement injectif). En outre, $\forall (u,v) \in E, m(u,v) = \{(u', v')\} \wedge \beta((u,v)) = \beta'((u', v'))$ (car $(u,v) \in E \Rightarrow (m(u), m(v)) \in E' \wedge \beta((u,v)) = \beta'((m(u), m(v)))$). Donc, étant données les définitions de δ_v^{psub} et δ_e^{psub}, la distance de G et G' par rapport à m est égale à 0 et la distance entre G et G' est égale à 0.

(2) $\Rightarrow$ (1). Si la distance entre G et G' est égale à 0, alors, étant donnée la définition de δ_v^{psub}, il existe un appariement m tel que $\forall v \in V, m(v) = \{v'\} \wedge \alpha(v) = \alpha'(v')$ et $\forall v \in V', |m(v)| \leq 1$. Par conséquent, m est un appariement injectif qui conserve les étiquettes de sommets du graphe G. En outre, si m induit une distance égale à 0, alors, $\forall (u,v) \in E, m(u,v) = \{(u', v')\} \wedge \beta((u,v)) = \beta'((u', v'))$. Par conséquent, chaque arc de G est apparié à exactement un arc de G' ayant la même étiquette et $(u,v) \in E \Rightarrow (m(u), m(v)) \in E' \wedge \beta((u,v)) = \beta'((m(u), m(v)))$. Donc, il existe un appariement injectif $m \subseteq V \times V'$ qui préserve tous les arcs de G et, par définition, G est un sous-graphe partiel de G'. □

Isomorphisme de sous-graphe induit

Définition du problème (Rappel). Étant donnés deux graphes étiquetés $G = (V, E, L_V, \alpha, L_E, \beta)$ et $G' = (V', E', L'_V, \alpha', L'_E, \beta')$ tels que $|V| \leq |V'|$, le graphe étiqueté G est isomorphe à un sous-graphe (induit) du graphe étiqueté G' si et seulement s'il existe un appariement injectif $m \subseteq V \times V'$ tel que $\forall (u,v) \in V^2, (u,v) \in E \Leftrightarrow (m(u), m(v)) \in E'$ et $\forall u \in V, \alpha(u) = \alpha'(m(u))$ et $\forall (u,v) \in E, \beta((u,v)) = \beta'((m(u), m(v)))$.

Définition de notre mesure. Le problème de l'isomorphisme de sous-graphe induit entre G et G' ajoute une contrainte sur chaque couple de sommets de V (être ou ne pas être apparié à un arc de G'). Afin de vérifier cette contrainte, la fonction de distance des arcs de G doit être définie pour chaque couple de sommets $(u,v) \in V \times V$ de G (et non pas seulement pour chaque arc de G) et pour chaque sous-ensemble $s_e \subseteq E'$ d'arcs de G'. Par conséquent, il est nécessaire de comparer le graphe complet G'' de G ($G'' = (V, V \times V)$) au graphe $G' = (V', E')$. La fonction de distance de sommets doit retourner $+\infty$ si l'appariement n'est pas injectif (1, 3 et 4) ou qu'un sommet de G n'est pas apparié à un sommet de G' ayant la même étiquette (1) et 0 sinon. La

fonction de distance d'arcs doit retourner $+\infty$ si un arc de G n'est pas apparié à un arc de G' ayant la même étiquette (2.1) ou si un couple (u, v) de sommets de G qui ne forme pas un arc est apparié à un arc de G' (2.2) et 0 sinon. Plus formellement, étant donnés un graphe $G = (V, E)$ et un graphe $G' = (V', E')$, il faut comparer les graphes $G'' = (V, V \times V)$ et G' avec la distance $\delta_G^{sub} =< \delta_v^{sub}, \delta_{eG}^{sub}, \otimes_{\sum} >$ définie par :

$$
G'' \begin{cases}
1 \quad \forall v \in V, \forall s_v \subseteq V', \delta_v^{sub}(v, s_v) = 0 \text{ si } s_v = \{v'\} \wedge \alpha(v) = \alpha'(v') \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad = +\infty \text{ sinon} \\
2.1 \quad \forall (u, v) \in V^2, \forall s_e \subseteq E', \delta_{eG}^{sub}(u, v, s_e) = 0 \text{ si } (u, v) \in E \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \wedge s_e = \{(u', v')\} \wedge \beta((u, v)) = \beta'((u', v')) \\
2.2 \qquad\qquad\qquad\qquad\qquad\quad = 0 \text{ si } (u, v) \notin E \wedge s_e = \emptyset \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad = +\infty \text{ sinon}
\end{cases}
$$

$$
G' \begin{cases}
3 \quad \forall v' \in V', \forall s_v \subseteq V, \delta_v^{sub}(v', s_v) = 0 \text{ si } |s_v| \leq 1 \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad = +\infty \text{ sinon} \\
4 \quad \forall (u', v') \in E', \forall s_e \subseteq E, \delta_{eG}^{sub}(u', v', s_e) = 0 \text{ si } |s_e| \leq 1 \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad = +\infty \text{ sinon}
\end{cases}
$$

Théorème 3. *Étant donnés deux graphes étiquetés $G = (V, E, L_V, \alpha, L_E, \beta)$ et $G' = (V', E', L'_V, \alpha', L'_E, \beta')$, les deux propriétés suivantes sont équivalentes :*

1. *le graphe G est un sous-graphe induit de G' ;*
2. *la distance $\delta_G^{sub} =< \delta_v^{sub}, \delta_{eG}^{sub}, \otimes_{\sum} >$ entre $G'' = (V, V \times V)$ et G' est égale à 0 .*

Démonstration. (1) $\Rightarrow$ (2). Par définition, si G est un sous-graphe induit de G', il existe un appariement injectif $m \subseteq V \times V'$ tel que $\forall (u, v) \in V \times V, (u, v) \in E \Leftrightarrow (m(u), m(v)) \in E'$ et $\forall u \in V, \alpha(u) = \alpha'(m(u))$ et $\forall (u, v) \in E, \beta((u, v)) = \beta'((m(u), m(v)))$. Par conséquent, $\forall v \in V, m(v) = \{v'\} \wedge \alpha(v) = \alpha'(v')$, $\forall v' \in V', |m(v')| \leq 1$ et $\forall (u', v') \in E', |m(u', v')| \leq 1$ (car m est un appariement injectif). En outre, $\forall (u, v) \in E, m(u, v) = \{(u', v')\} \wedge \beta((u, v)) = \beta'((u', v'))$ (car $\forall (u, v) \in V \times V, (u, v) \in E \Rightarrow (m(u), m(v)) \in E'$ et que $\forall (u, v) \in E, \beta((u, v)) = \beta'((m(u), m(v)))$) et $\forall (u, v) \in (V \times V) - E, m(u, v) = \emptyset$ (car $\forall (u, v) \in V \times V, (u, v) \notin E \Rightarrow (m(u), m(v)) \notin E'$). Donc, étant données les définitions des δ_v^{sub} et δ_{eG}^{sub}, la distance entre G'' et G' par rapport à m est égale à 0 et la distance entre G'' et G' est égale à 0.

(2) $\Rightarrow$ (1). Si la distance entre G'' et G' est égale à 0, alors, étant donnée la définition de δ_v^{sub}, il existe un appariement m tel que $\forall v \in V, m(v) = \{v'\} \wedge \alpha(v) = \alpha'(v')$ et $\forall v' \in V', |m(v')| \leq 1$. Par conséquent, m est un appariement injectif tel que $\forall v \in V, \alpha(v) = \alpha'(m(v))$. En outre, si m induit une distance égale à 0, alors, $\forall (u, v) \in E, m(u, v) = \{(u', v')\} \wedge \beta((u, v) = \beta'((u', v'))$. Par conséquent, chaque arc de G est apparié à exactement un arc de G' ayant la même étiquette. De plus, $\forall (u, v) \in (V \times V) - E, m(u, v) = \emptyset$, et chaque couple de sommets de G qui ne forme pas un arc de G est apparié à un couple de sommets de G' qui ne forme pas non plus un arc de G'. Par conséquent, m est un appariement injectif tel que $\forall (u, v) \in V \times V, (u, v) \in E \Leftrightarrow (m(u), m(v)) \in E'$, $\forall u \in V, \alpha(u) = \alpha'(m(u))$ et $\forall (u, v) \in E, \beta((u, v)) = \beta'((m(u), m(v)))$ et G est donc un sous-graphe induit de G'. □

Généralisation de l'isomorphisme de sous-graphe

Définition du problème (Rappel). Étant donnés un graphe étiqueté modèle $G_p = (V_p, E_p, L_V, \alpha_p, L_E, \beta_p)$, un ensemble de sommets optionnels $O_p \subseteq V_p$, un ensemble d'arcs interdits $F_p \subseteq (V_p \times V_p) - E_p$ et un graphe étiqueté cible $G_t = (V_t, E_t, L_V, \alpha_t, L_E, \beta_t)$ le problème de

l'isomorphisme de sous-graphe généralisé consiste à trouver un appariement entre le graphe modèle et le graphe cible qui respecte toutes les contraintes du problème, c.-à-d. un appariement $m \subseteq V_p \times V_t$ tel que :

1. m est univalent ;
2. $\forall u \in (V_p \backslash O_p), |m(u)| = 1$;
3. $\forall u \in V_p, m(u) \neq \emptyset \Rightarrow \alpha_p(u) = \alpha_t(m(u))$;
4. $\forall (u,v) \in E_p, m(u) \neq \emptyset \wedge m(v) \neq \emptyset \Rightarrow (m(u), m(v)) \in E_t \wedge \beta_p((u,v)) = \beta_t((m(u), m(v)))$;
5. $\forall (u,v) \in F_p, m(u) \neq \emptyset \wedge m(v) \neq \emptyset \Rightarrow (m(u), m(v)) \notin E_t$.

Définition de notre mesure. Résoudre un problème d'appariement approximatif de sous-graphe consiste à trouver un appariement univalent m entre le graphe G_p et le graphe complet (issu de G_t) $G' = (V_t, V_t \times V_t)$ tel que chaque sommet obligatoire est apparié à exactement un sommet ayant la même étiquette (1), chaque sommet facultatif est apparié à au plus un sommet ayant la même étiquette (2.1 et 2.2), chaque arc (u,v) est apparié à un couple de sommets (u',v') de G' qui forme un arc de G_t ayant la même étiquette (3.2) ou, si une de ses extrémités est facultative, n'est apparié à aucun couple de sommets (3.1), chaque arc interdit n'est pas apparié (4). Finalement, l'appariement recherché doit être univoque (1, 5 et 6). Plus formellement, il faut calculer la distance $\delta^{agm}_{G_t} =< \delta^{agm}_v, \delta^{agm}_{eG_t}, \otimes_{\sum} >$ entre les graphes $G = (V_p, E_p \cup F_p)$ et $G' = (V_t, E' = V_t \times V_t)$ définie par :

$$
\begin{array}{lll}
G & 1 & \forall v \in V_p - O_p, \forall s_v \subseteq V_t, \delta^{agm}_v(v, s_v) = 0 \text{ si } s_v = \{v'\} \wedge \alpha_p(v) = \alpha_t(v') \\
 & & \qquad = +\infty \text{ sinon} \\
 & 2.1 & \forall v \in O_p, \forall s_v \subseteq V_t, \delta^{agm}_v(v, s_v) = 0 \text{ si } s_v = \{v'\} \wedge \alpha_p(v) = \alpha_t(v') \\
 & 2.2 & \qquad = 0 \text{ si } s_v = \emptyset \\
 & & \qquad = +\infty \text{ sinon} \\
 & 3.1 & \forall (u,v) \in E_p, \forall s_e \subseteq V_t \times V_t, \delta^{agm}_{eG_t}(u,v,s_e) = 0 \text{ si } s_e = \emptyset \\
 & 3.2 & \qquad = 0 \text{ si } s_e = \{(u',v')\} \wedge (u',v') \in E_t \\
 & & \qquad \wedge \beta_p((u,v)) = \beta_t((u',v')) \\
 & & \qquad = +\infty \text{ sinon} \\
 & 4 & \forall (u,v) \in F_p, \forall s_e \subseteq E', \delta^{agm}_{eG_t}(u,v,s_e) = 0 \text{ si } s_e = \{(u',v')\} \wedge (u',v') \notin E_t \\
 & & \qquad = +\infty \text{ sinon} \\
G' & 5 & \forall v' \in V_t, \forall s_v \subseteq V_p, \delta^{agm}_v(v', s_v) = 0 \text{ si } |s_v| \leq 1 \\
 & & \qquad = +\infty \text{ sinon} \\
 & 6 & \forall (u',v') \in E', \forall s_e \subseteq E_p \cup F_p, \delta^{agm}_{eG_t}(u',v',s_e) = 0 \text{ si } |s_e| \leq 1 \\
 & & \qquad = +\infty \text{ sinon}
\end{array}
$$

Théorème 4. *Étant donnés un graphe étiqueté modèle $G_p = (V_p, E_p, L_V, \alpha_p, L_E, \beta_p)$, un ensemble de sommets optionnels $O_p \subseteq V_p$, un ensemble d'arcs interdits $F_p \subseteq (V_p \times V_p) - E_p$, un graphe étiqueté cible $G_t = (V_t, E_t, L_V, \alpha_t, L_E, \beta_t)$ et un appariement $m \subseteq V \times V'$, les deux propriétés suivantes sont équivalentes :*

1. *m est un solution du problème de l'appariement approximatif de sous-graphe entre le graphe modèle G_p et le graphe cible G_t ;*
2. *la distance $\delta^{agm}_{G_t} =< \delta^{agm}_v, \delta^{agm}_{eG_t}, \otimes_{\sum} >$ induite par l'appariement m entre les graphes $G = (V_p, E_p \cup F_p)$ et $G' = (V_t, V_t \times V_t)$ est égale à 0.*

Démonstration. (1) $\Rightarrow$ (2). Si m est une solution du problème de l'appariement approximatif de sous-graphe alors $\forall v \in V_p - O_p, m(v) = \{v'\} \wedge \alpha_p(v) = \alpha_t(v')$ (condition 2 et 3). En outre, $\forall v \in$

$O_p, (m(v) = \{v'\} \wedge \alpha_p(v) = \alpha_t(v')) \vee m(v) = \emptyset$ (condition 1 et 3). De plus, $\forall (u,v) \in E_p, m(u,v) = \emptyset \vee (m(u,v) = \{(u',v')\} \wedge \beta_p((u,v)) = \beta_t((u',v')))$ (condition 1 et 4). En outre, $\forall (u,v) \in F_p, (m(u), m(v)) \notin E_t$ (condition 5). Finalement, comme m est univalent, $\forall v' \in V_t, |m(v')| \leq 1$ et $\forall (u',v') \in V_t \times V_t, |m(u',v')| \leq 1$. Par conséquent, étant données les définitions des distances de sommets et d'arcs, la distance $\delta^{agm}_{G_t} =< \delta^{agm}_v, \delta^{agm}_{eG_t}, \otimes_{\sum} >$ induite par l'appariement m entre les graphes $G = (V_p, E_p \cup F_p)$ et $G' = (V_t, V_t \times V_t)$ est égale à 0.

(2) $\Rightarrow$ (1). Si la distance $\delta^{agm}_{eG_t} =< \delta^{agm}_v, \delta^{agm}_{eG_t}, \otimes_{\sum} >$ induite par l'appariement m entre les graphes $G = (V_p, E_p \cup F_p)$ et $G' = (V_t, V_t \times V_t)$ est égale à 0, alors, l'appariement m est univoque (condition 1) car, étant données les fonctions de distance de sommets et d'arcs, tous les appariements non-univoques de sommets engendrent une distance infinie (1,2 et 5). En outre, si la distance induite par m est égale à 0, alors $\forall v \in V_p - O_p, |m(v)| = 1$ et m respecte la condition 2. De plus si la distance est nulle, $\forall u \in V_p, m(v) \neq \emptyset \Rightarrow \alpha_p(v) = \alpha_t(m(v))$ (1, 2.1 et 2.2) et m respecte la condition 3. Si la distance des arcs est nulle, $\forall (u,v) \in E_p, m(u,v) \neq \emptyset \Rightarrow \beta_p((u,v)) = \beta_t((m(u), m(v)))$ (3.2) et m respecte donc la condition 4. Finalement, pour que la distance soit nulle, m est tel que $\forall (u,v) \in F_p, (m(u), m(v)) \notin E_t$ (4) et m respecte donc la condition 5. m est donc une solution du problème de l'appariement approximatif de sous-graphe.

□

3.2.2 Appariements de graphes à tolérance d'erreurs

Dans cette section, nous montrons comment modéliser des problèmes d'appariement de graphes à tolérance d'erreurs en mesure de distance de graphes. Pour chacun de ces problèmes, nous recherchons un appariement univoque des sommets de deux graphes. Par conséquent, les fonctions de distance de sommets et d'arcs sont définies de telle façon à ce qu'un appariement non-univoque des sommets engendre toujours une distance infinie. En outre, comme ces problèmes sont des problèmes d'optimisation, l'objectif est toujours de trouver un meilleur appariement, c.-à-d. celui engendrant la distance la plus faible.

Plus grand sous-graphe partiel commun

Définition du problème (Rappel). Étant donnés deux graphes G et G', le problème du plus grand sous-graphe partiel commun à deux graphes consiste à trouver la taille du plus grand sous-graphe partiel G'' de G isomorphe à un graphe partiel de G'. Pour ce problème, la taille d'un graphe $G = (V, E)$ est définie comme la somme de son nombre de sommets et de son nombre d'arcs, c.-à-d. $|G| = |V| + |E|$.

Définition de notre mesure. Les fonctions de distance de sommets et d'arcs à utiliser doivent interdire les appariements non-univoques tout en "encourageant" les sommets et les arcs de G et de G' ayant le même étiquette a être appariés. Par conséquent, les fonctions de distance de sommets et d'arcs doivent retourner $+\infty$ si l'élément est apparié à plus d'un élément, 1 s'il n'est apparié à aucun élément et 0 s'il est apparié à exactement un élément ayant la même étiquette.

Plus formellement, la distance $\delta^{mcps} =< \delta_v^{mcps}, \delta_e^{mcps}, \otimes_{\sum} >$ à utiliser est définie par :

$$\begin{aligned}
\forall v \in V, \forall s_v \subseteq V', \delta_v^{mcps}(v, s_v) &= 1 \text{ si } s_v = \emptyset \\
&= 0 \text{ si } s_v = \{v'\} \wedge \alpha(v) = \alpha(v') \\
&= +\infty \text{ sinon} \\
\forall (u, v) \in E, \forall s_e \subseteq E', \delta_e^{mcps}(u, v, s_e) &= 1 \text{ si } s_e = \emptyset \\
&= 0 \text{ si } s_e = \{(u', v')\} \wedge \beta((u, v)) = \beta((u', v')) \\
&= +\infty \text{ sinon} \\
\forall v' \in V', \forall s_v \subseteq V, \delta_v^{mcps}(v', s_v) &= 1 \text{ si } s_v = \emptyset \\
&= 0 \text{ si } s_v = \{v\} \wedge \alpha(v) = \alpha(v') \\
&= +\infty \text{ sinon} \\
\forall (u', v') \in E', \forall s_e \subseteq E, \delta_e^{mcps}(u', v', s_e) &= 1 \text{ si } s_e = \emptyset \\
&= 0 \text{ si } s_e = \{(u, v)\} \wedge \beta((u, v)) = \beta((u', v')) \\
&= +\infty \text{ sinon}
\end{aligned}$$

Théorème 5. *Étant donnés deux graphes étiquetés G et G' et un appariement $m \subseteq V \times V'$, les deux propriétés suivantes sont équivalentes :*

1. *m est un appariement qui minimise la distance $\delta^{mcps} =< \delta_v^{mcps}, \delta_e^{mcps}, \otimes_{\sum} >$;*
2. *le sous-graphe $G_{\downarrow m}$ de G induit par l'appariement m est un plus grand sous-graphe partiel commun à G et G'.*

Démonstration. La preuve est décomposée en deux étapes. Nous montrons d'abord que pour chaque appariement $m \subseteq V \times V'$ tel que $\delta_m^{mcps}(G, G') = d \neq +\infty$, le sous-graphe $G_{\downarrow m}$ de G induit par m est un sous-graphe partiel commun de G et G' et que $|G_{\downarrow m}| = (|G| + |G'| - d)/2$. Dans une seconde étape, nous montrons que, s'il existe un sous-graphe partiel G'' de G isomorphe à un sous-graphe partiel de G', alors, il est possible de trouver un appariement m engendrant une distance d égale à $|G| + |G'| - 2 * |G''|$ et telle que $G'' = G_{\downarrow m}$, le sous-graphe de G induit par l'appariement m. Dès lors, comme nous montrons que chaque sous-graphe partiel commun G'' correspond à un appariement induisant une distance non-infinie inversement proportionnelle à la taille de G'' (et inversement), la propriété est valide.

$\delta_m^{mcps}(G, G') = d < +\infty \Rightarrow G_m$ est un sous-graphe partiel commun de G et de G' tel que $|G_{\downarrow m}| = (|G| + |G'| - d)/2$. Étant données les définitions des fonctions de distances de sommets et d'arcs, si $\delta_m^{mcps}(G, G') < +\infty$, alors m est univoque (car tous les appariements non-univoques engendrent une distance infinie). Par définition, le sous-graphe $G_{\downarrow m} = (V_{\downarrow m}, E_{\downarrow m}, L_V, \alpha, L_E, \beta)$ de G induit par l'appariement m est un sous-graphe partiel de G et le sous-graphe $G'_{\downarrow m} = (V'_{\downarrow m}, E'_{\downarrow m}, L_V, \alpha', L_E, \beta')$ de G' induit par l'appariement m est un sous-graphe partiel de G'. Étant donnée la définition d'un sous-graphe induit par un appariement et sachant que l'appariement m est univoque, l'appariement m est un appariement bijectif entre les sommets des graphes $G_{\downarrow m}$ et $G'_{\downarrow m}$ tel que $(u, v) \in E_{\downarrow m} \Leftrightarrow (m(u), m(v)) \in E'_{\downarrow m}$. Par conséquent, et comme m préserve les étiquettes de sommets et d'arcs, $G_{\downarrow m}$ et $G'_{\downarrow m}$ sont isomorphes et $G_{\downarrow m}$ est un sous-graphe partiel commun à G et de G'. Étant données les définitions des distances de sommets et d'arcs, si $\delta_m^{mcps}(G, G') = d < +\infty$ alors $d = |G| + |G'| - |G_{\downarrow m}| - |G'_{\downarrow m}|$. Comme $G_{\downarrow m}$ et $G'_{\downarrow m}$ sont isomorphes, alors $|G_{\downarrow m}| = |G'_{\downarrow m}|$. Par conséquent, $|G_{\downarrow m}| = (|G| + |G'| - d)/2$ et la propriété est valide.

G'' est un sous-graphe partiel commun à G et $G' \Rightarrow$ il existe un appariement m tel que $\delta_m^{mcps}(G, G') = |G| + |G'| - 2 * |G''|$ et $G'' = G_{\downarrow m}$. S'il existe un sous-graphe partiel $G'' =$

$(V'', E'', L''_V, \alpha'', L''_E, \beta'')$ commun à $G = (V, E, L_V, \alpha, L_E, \beta)$ et $G' = (V', E', L'_V, \alpha, L'_E, \beta)$, alors, par définition d'un sous-graphe partiel commun, il existe au moins un graphe $G''' = (V''' \subseteq V', E''' \subseteq E', L'''_V, \alpha''', L'''_E, \beta''')$ et un appariement bijectif $m \subseteq V'' \times V'''$ tel que $(u, v) \in E'' \Leftrightarrow (m(u), m(v)) \in E'''$ et qui préserve les étiquettes de sommets et d'arcs. Par conséquent, l'appariement m est tel que $\forall v \in V'' \cup V''', |m(v)| = 1$ (car m est un appariement bijectif) et m préserve les étiquettes de sommets, $\forall (u, v) \in E'' \cup E''', |m(u, v)| = 1$ (car m est tel que $(u, v) \in E'' \Leftrightarrow (m(u), m(v)) \in E'''$) et m préserve les étiquettes d'arcs. En outre, par définition, m est tel que $\forall v \in V - V'', m(v) = \emptyset$, $\forall v \in V' - V''', m(v) = \emptyset$, $\forall (u, v) \in E - E'', m(u, v) = \emptyset$ et $\forall (u, v) \in E' - E''', m(u, v) = \emptyset$. Par conséquent, $\delta_m^{mcps}(G, G') = |G| + |G'| - |G''| - |G'''|$. G'' et G''' étant isomorphes, $|G''| = |G'''|$ et donc $\delta_m^{mcps}(G, G') = |G| + |G'| - 2 * |G''|$. La propriété est valide. □

Plus grand sous-graphe induit commun

Définition du problème (Rappel). Étant donnés deux graphes G et G', le problème du plus grand sous-graphe induit commun consiste à trouver le plus grand sous-graphe induit G'' de G qui est isomorphe à un sous-graphe induit de G'. Pour ce problème, la taille d'un graphe $G = (V, E)$ est définie par son nombre de sommets, c.-à-d. $|G| = |V|$.

Définition de notre mesure. Afin de résoudre le problème du plus grand sous-graphe induit commun à deux graphes, il faut utiliser des fonctions de distance de sommets et d'arcs encourageant les sommets de G à être appariés tout en interdisant les appariements qui ne correspondent pas à des sous-graphes induits communs. Dès lors, tout comme pour le problème de l'isomorphisme de sous-graphe induit, la fonction de distances d'arcs doit vérifier une contrainte entre (et donc être définie pour) chaque couple de sommets des deux graphes. Par conséquent, des graphes complets doivent être comparés. La fonction de distances de sommets encourage les sommets de G à être appariés à des sommets ayant la même étiquette (1) et la fonction de distances d'arcs retourne $+\infty$ quand un couple de sommets (u, v) de G (resp. (u', v') de G') est apparié à un couple de sommets (u', v') de G' (resp. (u, v) de G) tel que $(u, v) \in E \not\Leftrightarrow (u', v') \in E'$ ou que (u, v) et (u', v') n'ont pas la même étiquette (2). Finalement, l'appariement doit être univoque (1 et 3). Plus formellement, il faut calculer la distance entre les graphes $G_2 = (V, V \times V, L_V, \alpha, L_E \cup \{l_\emptyset\}, \beta_2)$ et $G'_2 = (V', V' \times V', L'_V, \alpha', L'_E \cup \{l_\emptyset\}, \beta'_2)$[5] où $\forall (u, v) \in V \times V, ((u, v) \in E \Rightarrow \beta_2((u, v)) = \beta((u, v))) \wedge ((u, v) \notin E \Rightarrow \beta_2((u, v)) = l_\emptyset$ et $\forall (u', v') \in V' \times V', ((u', v') \in E' \Rightarrow \beta'_2((u', v')) = \beta'((u', v'))) \wedge ((u', v') \notin E' \Rightarrow \beta'_2((u', v')) = l_\emptyset$ en utilisant la distance $\delta^{mcs} =< \delta_v^{mcs}, \delta_e^{mcs}, \otimes_\sum >$ définie par :

$$
G_2 \begin{cases}
1 & \forall v \in V, \forall s_v \subseteq V', \delta_v^{mcs}(v, s_v) = 1 \text{ si } s_v = \emptyset \\
& \qquad = 0 \text{ si } s_v = \{v'\} \wedge \alpha(v) = \alpha'(v') \\
& \qquad = +\infty \text{ sinon} \\
2 & \forall (u, v) \in V^2, \forall s_e \subseteq V'^2, \delta_{eGG'}^{mcs}(u, v, s_e) = 0 \text{ si } s_e = \emptyset \\
& \qquad = 0 \text{ si } s_e = \{(u', v')\} \wedge \beta_2((u, v)) = \beta'_2((u', v')) \\
& \qquad = +\infty \text{ sinon}
\end{cases}
$$

$$
G'_2 \begin{cases}
3 & \forall v' \in V', \forall s_v \subseteq V, \delta_v^{mcs}(v', s_v) = 0 \text{ si } |s_v| \leq 1 \\
& \qquad = +\infty \text{ sinon} \\
4 & \forall (u', v') \in V'^2, \forall s_e \subseteq V^2, \delta_{eGG'}^{mcs}(u', v', s_e) = 0 \text{ si } |s_e| \leq 1 \\
& \qquad = +\infty \text{ sinon}
\end{cases}
$$

[5] Nous supposerons que l'étiquette $l_\emptyset$ n'appartient pas aux ensembles L_E et L'_E.

Théorème 6. *Étant donnés deux graphes étiquetés* $G = (V, E, L_V, \alpha, L_E, \beta)$ *et* $G' = (V', E', L_V, \alpha', L'_E, \beta')$*, et un appariement* $m \subseteq V \times V'$*, les deux propriétés suivantes sont équivalentes :*

1. *m est un appariement qui minimise la distance* $\delta^{mcs} =< \delta_v^{mcs}, \delta_e^{mcs}, \otimes_{\sum} >$ *entre* $G_2 = (V, V \times V, L_V, \alpha, L_E \cup \{l_\emptyset\}, \beta_2)$ *et* $G'_2 = (V', V' \times V', L'_V, \alpha', L'_E \cup \{l_\emptyset\}, \beta'_2))$ *;*
2. *le sous-graphe $G_{\downarrow m}$ de G induit par l'appariement m est un plus grand sous-graphe induit commun aux deux graphes G et G'.*

Démonstration. La preuve de correction est décomposée en deux étapes. Nous montrons d'abord que, pour chaque appariement $m \subseteq V \times V'$ tel que $\delta^{mcs}_{mGG'} = d \neq +\infty$, le sous-graphe $G_{\downarrow m}$ de G induit par l'appariement m est un sous-graphe induit commun à G et G' tel que $|G_{\downarrow m}| = |G| - d$. Dans une seconde étape, nous montrons que, s'il existe un sous-graphe induit G'' de G isomorphe à un sous-graphe induit de G', alors, il est possible de trouver un appariement m induisant une distance d égale à $|G| - |G''|$ et telle que $G'' = G_{\downarrow m}$, le sous-graphe de G induit par l'appariement m. Nous aurons alors prouvé que chaque sous-graphe induit commun G'' correspond à un appariement induisant une distance non-infinie inversement proportionnelle à la taille de G'' (et inversement) et que donc, la propriété est valide.

$\delta^{mcs}_{mGG'}(G_2, G'_2) = d < +\infty \Rightarrow G_{\downarrow m}$ est un sous-graphe induit commun à G et G' tel que $|G_{\downarrow m}| = |G| - d$. Étant données les définitions des fonctions de distances de sommets et d'arcs, si $\delta_{mGG'}(G_2, G'_2) < +\infty$, alors m est un appariement univoque (car tous les appariements non-univoques engendrent une distance égale à $+\infty$). Par définition, le sous-graphe $G_{2\downarrow m} = (V_{2\downarrow m}, E_{2\downarrow m})$ de G_2 induit par l'appariement m est un sous-graphe partiel de G_2 et de G. En outre, étant donnée la définition de la fonction de distances d'arcs et l'étiquetage des arcs de G_2 et G'_2, $(u, v) \in E_{2\downarrow m} \Rightarrow (u, v) \in E$ et $(u, v) \notin E_{2\downarrow m} \Rightarrow (u, v) \notin E$. Par conséquent et comme m préserve les étiquettes de sommets et d'arcs, $G_{2\downarrow m}$ est un sous-graphe induit (c.-à-d. non partiel) de G et $G_{2\downarrow m} = G_{\downarrow m}$. De la même façon, il est possible de montrer que le sous-graphe $G'_{2\downarrow m} = (V'_{2\downarrow m}, E'_{2\downarrow m})$ de G'_2 induit par l'appariement m est un sous-graphe induit de G' et que $G'_{2\downarrow m} = G'_m$. Finalement, m est un appariement univoque et, étant données les définitions des fonctions de distances de sommets et d'arcs, m est tel que $(u, v) \in E_{\downarrow m} \Leftrightarrow (m(u), m(v)) \in E'_{\downarrow m}$ et m est un appariement montrant que les graphes $G_{\downarrow m}$ et $G'_{\downarrow m}$ sont isomorphes car m préserve les étiquettes. Par conséquent, $G_{\downarrow m}$ est un sous-graphe induit de G et de G'. Finalement, comme seul le nombre de sommets non-appariés influe (positivement) la distance, $|G_{\downarrow m}| = |G| - d$.

G'' est un sous-graphe induit de G et $G' \Rightarrow \exists$ un appariement m tel que $\delta_{mGG'}(G_2, G'_2) = |G| - |G''|$ et tel que $G_{\downarrow m} = G''$. S'il existe un sous-graphe induit $G'' = (V'', E'', L''_V, \alpha'', L''_E, \beta'')$ commun à G et G', alors, par définition d'un sous-graphe induit commun, il existe au moins un sous-graphe induit $G''' = (V''', E''', L'''_V, \alpha''', L'''_E, \beta''')$ de G' et un appariement bijectif $m \subseteq V'' \times V'''$ tel que $(u, v) \in E'' \Leftrightarrow (m(u), m(v)) \in E'''$ et qui préserve les étiquettes de sommets et d'arcs. Étant données les définitions des fonctions de distances de sommets et d'arcs, nous pouvons voir que la distance $\delta_{mGG'}(G_2, G'_2)$ est alors égale à $|G| - |G''|$ et que $G_{\downarrow m} = G''$. □

Distance d'édition de graphe (*ged*)

Définition du problème (Rappel). Étant donnés deux graphes étiquetés $G_1 = (V_1, E_1, L_1, \alpha_1, \beta_1)$ et $G_2 = (V_2, E_2, L_2, \alpha_2, \beta_2)$, un *appariement à tolérance d'erreurs* est un appariement univoque $m \subseteq V_1 \times V_2$. Un sommet $u \in V_1$ est substitué par le sommet v si $m(u) = v$. Si $\alpha_1(u) = \alpha_2(m(u))$, la substitution est dite *identique* sinon, elle est dite *non-identique*. Tous les sommets $v \in V_1$ tels que $m(v) = \emptyset$ sont dits *supprimés* par m et tous les sommets $v' \in V_2$ tels que $m(v') = \emptyset$ sont dits *insérés* par m. Les mêmes termes sont utilisés pour les arcs insérés, substitués et supprimés. Un coût c_{vs} (resp. c_{vi} et c_{vd}) est associé aux substitutions de sommets

non-identiques (resp. insertions et suppressions) et un coût c_{es} (resp. c_{ei} et c_{ed}) est associé aux substitutions d'arcs non-identiques (resp. insertions et suppressions). Le *coût d'un appariement à tolérance d'erreurs* m est la somme des coûts de chacune des opérations induites par m. La *distance d'édition de graphe* est le coût de l'appariement à tolérance d'erreurs le moins coûteux.

Définition de notre mesure. Chaque appariement univoque de notre modèle correspond à un appariement de graphes à tolérance d'erreur du modèle de Bunke et Jiang [BJ00]. Par conséquent, si les fonctions de distance des sommets et des arcs sont définies de façon à reproduire le coût de chacune des opérations d'éditions tout en interdisant les appariements non-univoques, la distance entre les graphes $G'_1 = (V_1, E_1)$ et $G'_2 = (V_2, E_2)$ par rapport à un appariement m correspond au coût de l'appariement à tolérance d'erreur défini par m. Plus formellement, pour calculer la distance d'édition de graphe entre les deux graphes étiquetés $G_1 = (V_1, E_1, L_1, \alpha_1, \beta_1)$ et $G_2 = (V_2, E_2, L_2, \alpha_2, \beta_2)$, il faut comparer les graphes $G'_1 = (V_1, E_1)$ et $G'_2 = (V_2, E_2)$ avec la distance $\delta^{ged}_{G_1G_2} =< \delta^{ged}_{vG_1G_2}, \delta^{ged}_{eG_1G_2}, \otimes_{\sum} >$ définie par :

$$
G'_1 \begin{cases}
\forall v \in V_1, \forall s_v \subseteq V_2, \delta^{ged}_{vG_1G_2}(v, s_v) & = & c_{vd} \text{ si } s_v = \emptyset \\
& = & 0 \text{ si } s_v = \{v'\} \wedge \alpha_1(v) = \alpha_2(v') \\
& = & c_{vs} \text{ si } s_v = \{v'\} \wedge \alpha_1(v) \neq \alpha_2(v') \\
& = & +\infty \text{ si } |s_v| > 1 \\
\forall (u, v) \in E_1, \forall s_e \subseteq E_2, \delta^{ged}_{eG_1G_2}(u, v, s_e) & = & c_{ed} \text{ si } s_e = \emptyset \\
& = & 0 \text{ si } s_e = \{(u', v')\} \wedge \beta_1((u, v)) = \beta_2((u', v')) \\
& = & c_{es} \text{ si } s_v = \{(u', v')\} \wedge \beta_1((u, v)) \neq \beta_2((u', v')) \\
& = & +\infty \text{ si } |s_e| > 1
\end{cases}
$$

$$
G'_2 \begin{cases}
\forall v \in V_2, \forall s_v \subseteq V_1, \delta^{ged}_{vG_1G_2}(v, s_v) & = & c_{vi} \text{ si } s_v = \emptyset \\
& = & 0 \text{ si } |s_v| = 1 \\
& = & +\infty \text{ si } |s_v| > 1 \\
\forall (u, v) \in E_2, \forall s_e \subseteq E_1, \delta^{ged}_{eG_1G_2}(u, v, s_e) & = & c_{ei} \text{ si } s_e = \emptyset \\
& = & 0 \text{ si } |s_e| = 1 \\
& = & +\infty \text{ si } |s_e| > 1
\end{cases}
$$

Théorème 7. *Étant donnés deux graphes étiquetés* $G_1 = (V_1, E_1, L_1, \alpha_1, \beta_1)$ *et* $G_2 = (V_2, E_2, L_2, \alpha_2, \beta_2)$*, la distance d'édition de graphe de Bunke et Jiang [BJ00] est égale à la distance* $\delta^{ged}_{G_1G_2} =< \delta^{ged}_{vG_1G_2}, \delta^{ged}_{eG_1G_2}, \otimes_{\sum} >$ *entre le graphe* $G'_1 = (V_1, E_1)$ *et le graphe* $G'_2 = (V_2, E_2)$*.*

Démonstration. La preuve de correction est triviale : il suffit de montrer l'équivalence entre l'ensemble des appariements de graphes à tolérance d'erreurs et l'ensemble des appariements univoques puis de montrer que, étant donné un appariement m, la distance par rapport à m est égale au coût de l'appariement à tolérance d'erreurs m. □

3.2.3 Appariements de graphes multivoques

Dans cette section, nous montrons comment modéliser des problèmes d'appariement multivoque de graphes comme un calcul de notre mesure de distance de graphes. Ces problèmes étant des problèmes d'optimisation, l'objectif est toujours de trouver l'appariement induisant la distance la plus faible.

Distance d'édition de graphes étendue

Définition du problème (Rappel). Par rapport à la distance d'édition "classique", deux nouvelles opérations d'éditions sont ajoutées à la distance d'édition étendue : l'éclatement de sommets et la fusion de sommets. Chacune de ces deux opérations est pondérée par un coût c_{split} et c_{merge}. Une contrainte de cohérence sur les opérations de fusion et d'éclatement de sommets est ajoutée : quand deux sommets u et v d'un graphe sont appariés à un même sommet u', les sommet u et v sont fusionnés et il devient impossible d'apparier u à un sommet v' sans apparier également v à v'.

Définition de notre mesure. Le problème du calcul de la distance d'édition de graphes étendue peut être modélisé en un calcul de notre distance de graphes de la même façon que la distance d'édition de graphe (non-étendue). Les fonctions de distances de sommets et d'arcs sont similaires mais ne doivent pas retourner une distance $+\infty$ quand un appariement multivoque est considéré et la fonction de distances de sommets δ_v doit prendre en compte le coût des opérations d'éclatements et de fusions de sommets. Cependant, cette modélisation ne prend pas en compte la contrainte de cohérence des fusions et des éclatements de sommets.

Afin de prendre en compte cette contrainte, il faut que l'appariement recherché représente les opérations d'édition et non pas le résultat de ces opérations. Nous introduisons donc un "graphe d'opération" G_O : chaque sommet de ce graphe correspond à une opération d'édition possible entre les sommets des deux graphes à comparer. Les sommets "opérations" du graphe d'opérations doivent être appariés aux sommets des deux graphes G et G' à comparer. Lorsqu'un sommet du graphe G_O n'est apparié qu'à un sommet v du graphe G, cela correspond à une opération de suppression du sommet v. Lorsqu'un sommet du graphe G_O n'est apparié qu'à un sommet v' du graphe G', cela correspond à une opération d'insertion du sommet v'. Lorsqu'un sommet de G_O est apparié à un sommet v du graphe G et à un sommet v' du graphe G', cela correspond à une opération de substitution du sommet v par le sommet v'. Les arcs appariés aux arcs de G_0 représentent de la même façon les opérations de suppression, d'insertion et de substitution d'arcs. Lorsqu'un sommet de G_O est apparié à plusieurs sommets de G (resp. de G'), cela correspond à une opération de fusion (resp. d'éclatement) de sommets. Notons que si les sommets de G et de G' doivent être appariés à un et un seul sommet de G_O, tous les appariements respectant cette contrainte correspondent à une chaîne d'édition étendue de graphe qui respecte la contrainte de cohérence des fusions et des éclatements de sommets.

De façon plus formelle, afin de modéliser le problème de la distance d'édition de graphe étendue entre un graphe $G_1 = (V_1, E_1, L_1, \alpha_1, \beta_1)$ et $G_2 = (V_2, E_2, L_2, \alpha_2, \beta_2)$, il faut comparer le graphe $G'' = (V'' = V_1 \cup V_2, E'' = E_1 \cup E_2)$ (les sommets des graphes G_1 et G_2 sont disjoints) et le graphe complet $G_O = (V_O, E_O = V_O \times V_O)$ tel que $|V_O| = |V_1| + |V_2|$ (car il y a au plus une opération pour chaque sommet des graphes G_1 et G_2). Les fonctions δ_v^{eged} et δ_e^{eged} doivent contraindre les sommets de G_1 et de G_2 a être appariés à exactement un sommet de G_O. Le coût des suppressions, des insertions et des substitutions de sommets et d'arcs ainsi que le coût des fusions et des éclatements de sommets est calculé sur les sommets "opérations" du graphe G_O. Plus formellement, nous définissons la distance $\delta^{eged} =< \delta_v^{eged}, \delta_e^{eged}, \otimes_{\sum} >$ suivante :

$$G_2 \begin{cases} 1 & \forall v \in V'', \forall s_v \subseteq V_O, \delta_v^{eged}(v, s_v) = 0 \text{ si } |s_v| = 1 \\ & \qquad\qquad\qquad\qquad\qquad\quad = +\infty \text{ sinon} \\ 2 & \forall (u,v) \in E'', \forall s_e \subseteq E_O, \delta_e^{eged}((u,v), s_e) = 0 \\ 3.1 & \forall v_o \in V_O, \forall s_v \subseteq V'', \delta_v^{eged}(v_o, s_v) = 0 \text{ si } s_v = \emptyset \\ 3.2 & \qquad\qquad\qquad\qquad\qquad\quad = app_v(s_v \cap V_1, s_v \cap V_2) \text{ sinon} \\ 4.1 & \forall (u_o, v_o) \in E_O, \forall s_e \subseteq E'', \delta_e^{eged}((u_o, v_o), s_e) = 0 \text{ si } s_e = \emptyset \\ 4.2 & \qquad\qquad\qquad\qquad\qquad\quad = app_e(s_e \cap E_1, s_e \cap E_2) \text{ sinon} \end{cases}$$

où $app_v(s_v, s'_v)$ (resp. $app_e(s_e, s'_e)$) est le coût nécessaire pour apparier l'ensemble éventuellement vide de sommets de s_v (resp. d'arcs de s_e) à l'ensemble éventuellement vide de sommets de s'_v (resp. d'arcs de s'_e). De façon plus formelle, les fonctions $app_v : \wp(V_1) \times \wp(V_2) \to [0, +\infty[$ et $app_e : \wp(E_1) \times \wp(E_2) \to [0, +\infty[$ sont définies par :

$$\begin{array}{lll} a & \forall s_v \subseteq V, \forall s'_v \subseteq V', app_v(s_v, s'_v) & = merge(s_v) + merge(s'_v) \\ & & \quad + subst_v(s_v, s'_v) \qquad \text{si } s_v \neq \emptyset \wedge s'_v \neq \emptyset \\ b & & = merge(s_v) + supp_v(s_v) \qquad \text{si } s_v \neq \emptyset \wedge s'_v = \emptyset \\ c & & = merge(s'_v) + ins_v(s'_v) \qquad \text{si } s_v = \emptyset \wedge s'_v \neq \emptyset \\ d & \forall s_e \subseteq V, \forall s'_e \subseteq V', app_e(s_e, s'_e) & = subst(s_e, s'_e) \qquad \text{si } s_e \neq \emptyset \wedge s'_e \neq \emptyset \\ e & & = supp_e(s_e) \qquad \text{si } s_e \neq \emptyset \wedge s'_e = \emptyset \\ f & & = ins_e(s'_e) \qquad \text{si } s_e = \emptyset \wedge s'_e \neq \emptyset \end{array}$$

La fonction $merge(s_v)$ correspond au coût de la fusion des sommets s_v, la fonction $subst_v(s_v, s'_v)$ (resp. $subst_e(s_e, s'_e)$) correspond au coût de la substitution de l'ensemble des sommets de s_v (resp. des arcs de s_e) par l'ensemble des sommets de s'_v (resp. des arcs de s'_e). $ins_v(s_v)$ (resp. $ins_e(s_e)$) correspond au coût de l'insertion des sommets (resp. arcs) de s_v (resp. s_e) et $supp_v(s_v)$ (resp. $supp_e(s_e)$) à leur suppression.

Théorème 8. *Étant donnés deux graphes étiquetés $G_1 = (V_1, E_1, L_1, \alpha_1, \beta_1)$ et $G_2 = (V_2, E_2, L_2, \alpha_2, \beta_2)$, la distance d'édition de graphe étendue est égale à la distance $\delta^{eged} =< \delta_v^{eged}, \delta_e^{eged}, \otimes_{\sum} >$ entre le graphe $G'' = (V_1 \cup V_2, E_1 \cup E_2)$ et un graphe $G_O = (V_O, V_O \times V_O)$ tel que $|V_O| = |V_1| + |V_2|$.*

Démonstration. Tout appariement menant à une distance différente de $+\infty$ correspond à une chaîne d'édition de graphe étendue (et inversement). En outre, les fonctions de distance des sommets et des arcs sont telles que le coût de cette chaîne est égale à la distance induite par l'appariement. □

Problème de l'appariement non-bijectif de graphes

Définition du problème (Rappel). La similarité de Boeres *et al.* [BRB04] entre une image et son schéma par rapport à un appariement est calculée à partir de matrices de similarité de sommets et d'arcs. Le problème consiste à trouver le meilleur appariement qui respecte des contraintes propres au problème. Deux graphes sont utilisés pour ce problème : le graphe modèle $G_1 = (V_1, E_1)$ et le graphe image $G_2 = (V_2, E_2)$ (avec $|V_1| \leq |V_2|$). Une solution de ce problème est un appariement $m \subseteq V_1 \times V_2$ entre les graphes G_1 et G_2 tel que chaque sommet de G_1 est apparié à un ensemble connecté non-vide de sommets de G_2 (c.-à-d. $\forall v \in V_1, |m(v)| \geq 1$ et les sommets de $m(v)$ sont connectés par des arcs de E_2), et tel que chaque sommet de G_2 est apparié à exactement un sommet de G_1 (c.-à-d. $\forall v \in V_2, |m(v)| = 1$). Finalement, certains couples de sommets sont "interdits". Étant donné un appariement qui respecte ces contraintes, une mesure de similarité $sim[Boeres]_m$ est calculée par rapport à des matrices de similarité de sommets et

d'arcs $sm_v : V_1 \times V_2 \rightarrow [0,1]$ et $sm_e : E_1 \times E_2 \rightarrow [0,1]$ comme ceci :

$$sim[Boeres]_m = \frac{\sum_{(u,v)\in m} sm_v(u,v)}{|V_1|.|V_2|} + \frac{\sum_{(u,v)\in(V_1\times V_2)-m} 1 - sm_v(u,v)}{|V_1|.|V_2|} + \frac{\sum_{((u,u'),(v,v'))\in E_1\times E_2 | \{(u,v),(u',v')\}\subseteq m} sm_e((u,u'),(v,v'))}{|E_1|.|E_2|} + \frac{\sum_{((u,u'),(v,v'))\in E_1\times E_2 | \{(u,v),(u',v')\}\not\subseteq m} 1 - sm_e((u,u'),(v,v'))}{|E_1|.|E_2|}$$

Définition de la mesure. En choisissant correctement les fonctions de distance des sommets et des arcs δ_v et δ_e, il est possible de modéliser le problème de l'appariement non-bijectif de graphes de Boeres *et al.* en un problème de calcul de notre distance de graphes. La fonction de distances de sommets retourne $+\infty$ quand l'appariement viole une contrainte et les deux fonctions de distances de sommets et d'arcs doivent reproduire les matrices de similarité sm_v et sm_e. De façon plus formelle, nous définissons la distance $\delta^{nbgm} =< \delta_v^{nbgm}, \delta_e^{nbgm}, \otimes_{\sum} >$ suivante :

$$G_1 \begin{cases} \forall v \in V_1, \forall s_v \subseteq V_2, \delta_v^{nbgm}(v,s_v) = \sum_{v'\in s_v} 1 - sm_v(v,v') \\ \qquad\qquad\qquad\qquad\quad + \sum_{v'\in V_2 - s_v} sm_v(v,v') \\ \qquad\qquad\qquad\qquad\quad \text{si } connecte(v,s_v) \\ \qquad\qquad\qquad\qquad\quad +\infty \text{ sinon} \\ \forall (u,v) \in E_1, \forall s_e \subseteq E_2, \delta_e^{nbgm}((u,v),s_e) = \sum_{(u',v')\in s_e} 1 - sm_e((u,v),(u',v')) \\ \qquad\qquad\qquad\qquad\quad + \sum_{(u',v')\in E_2 - s_e} sm_e((u,v),(u',v')) \end{cases}$$

$$G_2 \begin{cases} \forall v \in V_2, \forall s_v \subseteq V_1, \delta_v^{nbgm}(v,s_v) = 0 \text{ si } autorise(v,s_v) \\ \qquad\qquad\qquad\qquad\quad +\infty \text{ sinon} \\ \forall (u',v') \in E_2, \forall s_e \subseteq E_1, \delta_e^{nbgm}((u',v'),s_e) = 0 \end{cases}$$

où *connecte* et *autorise* sont deux fonctions introduites pour vérifier les contraintes. *connecte* est une fonction qui retourne $faux$ quand un sommet du modèle n'est pas apparié où quand il est apparié à un ensemble de sommets non-connectés et *vrai* sinon. *autorise* est une fonction qui retourne $faux$ quand un sommet du graphe image n'est pas apparié à un et un seul sommet autorisé du graphe modèle et *vrai* sinon :

$$\begin{aligned} \forall v \in V_1, \forall s_v \subseteq V_2, connecte(v,s_v) &= \text{vrai si } s_v \text{ est un ensemble non-vide} \\ &\quad \text{de sommets connectés} \\ &\quad \text{faux sinon} \\ \forall v \in V_2, \forall s_v \subseteq V_1, autorise(v,s_v) &= \text{vrai si } s_v = \{v'\} \wedge (v,v') \text{ est autorise} \\ &\quad \text{faux sinon} \end{aligned}$$

Théorème 9. *Si l'appariement m minimisant la distance $\delta^{nbgm} =< \delta_v^{nbgm}, \delta_e^{nbgm}, \otimes_{\sum} >$ induit une distance non-infinie, alors m est l'appariement qui maximise la similarité de Boeres et al. sinon, il n'existe pas d'appariement satisfaisant les contraintes du problème de la similarité de Boeres et al.*

Démonstration. Il est simple de montrer que, grâce aux fonctions *connecte* et *autorise*, la distance entre G_1 et G_2 par rapport à un appariement m est égale à $+\infty$ si et seulement si m est un appariement qui viole au moins une contrainte. Finalement, en décomposant les fonctions de distance des sommets et des arcs, nous pouvons montrer que la distance δ^{nbgm} est inversement proportionnelle à la similarité de Boeres *et al.* [BRB04] et que par conséquent, l'appariement qui minimise la distance δ^{nbgm} est l'appariement qui maximise la similarité de Boeres *et al.* □

3.2.4 Similarité de Champin et Solnon [CS03]

Dans [SS05a, SS05b], nous avons montré que la similarité de Champin et Solnon [CS03] est générique : en instanciant correctement les deux fonctions f et g, cette mesure peut elle aussi être utilisée pour résoudre les problèmes d'appariement de graphes cités plus haut dans ce chapitre. Dans cette section, nous rappelons la définition de la mesure de Champin et Solnon et nous montrons que cette mesure et notre distance sont équivalentes dans le sens où elles permettent d'exprimer les même problèmes d'appariement.

Définition de la mesure de similarité de Champin et Solnon [CS03] (Rappel).

Afin de mesurer la similarité de deux graphes multi-étiquetés $G = \langle V, r_V, r_E\rangle$ et $G' = \langle V', r_{V'}, r_{E'}\rangle$ définis sur les mêmes ensembles d'étiquettes de sommets et d'arcs L_V et L_E, une première étape consiste à mettre en correspondance leurs sommets et à identifier l'ensemble de caractéristiques $G \sqcap_m G'$ communes aux deux graphes par rapport à l'appariement m :

$$\begin{aligned} G \sqcap_m G' \doteq\ & \{(v,l) \in r_V | \exists v' \in m(v), (v',l) \in r_{V'}\} \\ \cup\ & \{(v',l) \in r_{V'} | \exists v \in m(v'), (v,l) \in r_V\} \\ \cup\ & \{(v_i,v_j,l) \in r_E | \exists (v'_i,v'_j) \in m(v_i,v_j), (v'_i,v'_j,l) \in r_{E'}\} \\ \cup\ & \{(v'_i,v'_j,l) \in r_{E'} | \exists (v_i,v_j) \in m(v'_i,v'_j), (v_i,v_j,l) \in r_E\} \end{aligned}$$

Il est également nécessaire d'identifier l'ensemble des sommets "éclatés" ou "split" :

$$splits(m) = \{(v, m(v)) | v \in V \cup V', |m(v)| \geq 2\}$$

La **similarité** de G et de G' relativement à l'appariement m est alors définie par :

$$sim_m(G,G') = \frac{f(G \sqcap_m G') - g(splits(m))}{f(r_V \cup r_E \cup r_{V'} \cup r_{E'})}$$

où f et g sont deux fonctions introduites pour valuer les caractéristiques et les éclatements de sommets et qui dépendent de l'application considérée.

Finalement, la **similarité maximale** $sim(G,G')$ de deux graphes G et G' est la similarité induite par le meilleur appariement, c.-à-d. celui induisant la similarité la plus élevée.

$$sim(G,G') = \max_{m \subseteq V \times V'} \frac{f(G \sqcap_m G') - g(splits(m))}{f(r_V \cup r_E \cup r_{V'} \cup r_{E'})}$$

Notre mesure de distance de graphe et la similarité de Champin et Solnon

Notre distance de graphe et la similarité de graphe de Champin et Solnon ont toutes les deux été montrées génériques. Elles peuvent toutes les deux être utilisées pour modéliser les mesures de similarité ou de distance de graphes de la littérature. Nous montrons dans cette sous section que ces deux mesures sont équivalentes dans le sens où elles ont la même capacité à représenter des problèmes d'appariement.

Dans un premier temps, nous montrons que quelle que soit la façon dont la similarité de Champin et Solnon est définie, il existe une distance de graphe équivalente, c.-à-d. permettant de trouver les même appariements optimaux.

Théorème 10. *Étant donnés deux graphes multi-étiquetés* $G_1 = \langle V_1, r_{V1}, r_{E1}\rangle$ *et* $G_2 = \langle V_2, r_{V2}, r_{E2}\rangle$ *définis sur les mêmes ensemble d'étiquettes de sommets et d'arcs* L_V *et* L_E *et deux fonctions* f *et* g *instanciant la mesure de similarité de Champin et Solnon pour* G_1 *et* G_2*, il existe une définition de la distance* $\delta =< \delta_v, \delta_e, \otimes >$ *entre deux graphes* $G'_1 = (V_1, E_1)$ *et* $G'_2 = (V_2, E_2)$ *telle que l'appariement* m *qui minimise la distance entre* G'_1 *et* G'_2 *soit l'appariement qui maximise la similarité de Champin et Solnon entre les graphes* G_1 *et* G_2*.*

Démonstration. Afin de faire la preuve de ce théorème, nous montrons dans un premier temps qu'un graphe multi-étiqueté $G = \langle V, r_V, r_E\rangle$ peut être modélisé par un graphe $G' = (V, E)$ et par deux fonctions l_v et l_e étiquetant les sommets et les arcs de G'. Nous montrons ensuite qu'il est possible de définir les fonctions de distance des sommets et des arcs δ_v et δ_e de façon à ce que les arguments passés à la fonction $\otimes$ contiennent toute l'information nécessaire pour retrouver l'appariement effectué, c.-à-d. toute l'information nécessaire au calcul des arguments des fonctions f et g. Par conséquent, la fonction $\otimes$ peut être définie de façon à calculer les valeurs des fonctions f et g et donc la similarité de Champin et Solnon.

Quel que soit le graphe multi-étiqueté $G = \langle V, r_V, r_E\rangle$ défini sur les ensembles L_V et L_E d'étiquettes de sommets et d'arcs, il est possible de définir un graphe $G' = (V, E)$ et des fonctions d'étiquetage des sommets et des arcs contenant la même information :

- $E = \{(u, v) | \exists (u, v, l) \in r_E\}$;
- $l_v : V \rightarrow \wp(L_V), \forall v \in V, l_v(v) = \{l | (v, l) \in r_V\}$;
- $l_e : E \rightarrow \wp(L_E), \forall (u, v) \in E, l_e(u, v) = \{l | (u, v, l) \in r_E\}$.

Une fois les graphes à comparer définis, il est nécessaire de définir les fonctions de distance de sommets et des arcs de ces graphes. Ces fonctions doivent retourner des valeurs permettant de retrouver l'appariement réalisé. De façon plus formelle, les ensembles des sommets des graphes étant finis, il est toujours possible de définir une fonction bijective $num : \wp(V'_2) \rightarrow N$ associant un identifiant (entier) à chaque sous-ensemble de sommets de G'_2 avec lesquels un sommet de G'_1 est susceptible d'être apparié. Cette fonction bijective num est alors utilisée par la fonction de distance des sommets δ_v pour retourner l'ensemble des sommets de G'_2 apparié à chaque sommet de G'_1 :

$$\begin{aligned}
\forall v \in V_1, \forall s_v \subseteq V_2, \delta_v(v, s_v) &= num(s_v) \\
\forall v \in V_2, \forall s_v \subseteq V_1, \delta_v(v, s_v) &= 0 \\
\forall (u, v) \in E_1, \forall s_e \subseteq E_2, \delta_e((u, v), s_e) &= 0 \\
\forall (u', v') \in E_2, \forall s_e \subseteq E_1, \delta_e((u', v'), s_e) &= 0
\end{aligned}$$

Avec une telle fonction δ_v, l'ensemble $S = \{(u, \delta_v(u, m(u))) | u \in V\}$ permet de reconstituer l'appariement initialement effectué :

$$m \quad = \quad \{(u,u')|\exists(u,d) \in S \wedge u' \in num^{-1}(d)\}$$

L'ensemble S de ces tuples est un sous-ensemble des arguments passés à la fonction $\otimes$. La fonction $\otimes$ peut donc être définie de telle façon à retrouver l'appariement m réalisé et peut donc calculer les ensembles $G \sqcap_m G'$ et $splits(m)$ de la mesure de similarité de Champin et Solnon. Par conséquent, la fonction $\otimes$ peut reconstituer les arguments des fonctions f et g et peut donc calculer la mesure de similarité de Champin et Solnon[6]. □

De la même façon, la preuve peut-être réalisée dans l'autre sens : pour toute définition de notre distance de graphe, il existe une définition des fonctions f et g de la mesure de Champin et Solnon permettant de découvrir les même appariements.

Théorème 11. *Étant donnée une définition $\delta =< \delta_v, \delta_e, \otimes >$ de la distance entre deux graphes $G_1 = (V_1, E_1)$ et $G_2 = (V_2, E_2)$, il existe une définition des fonctions f et g de la mesure de la similarité de Champin et Solnon de deux graphes multi-étiquetés $G'_1 = \langle V_1, r_{V1}, r_{E1}\rangle$ et $G'_2 = \langle V_2, r_{V2}, r_{E2}\rangle$ telle que l'appariement m qui maximise la mesure de similarité de Champin et Solnon entre les graphes G'_1 et G'_2 soit l'appariement qui minimise la distance $\delta =< \delta_v, \delta_e, \otimes >$ entre les graphes G_1 et G_2.*

Démonstration. Afin de faire la preuve de ce théorème, nous montrons que, en choisissant correctement les graphes multi-étiquetés G_1 et G_2 à comparer, l'ensemble $G_1 \sqcap_m G_2$ des caractéristiques communes par rapport à un appariement m peut contenir toute l'information nécessaire pour retrouver l'appariement m réalisé. Par conséquent, la fonction f qui reçoit en argument cet ensemble peut être définie de façon à calculer les fonctions δ_v, δ_e et $\otimes$ et donc la distance $\delta =< \delta_v, \delta_e, \otimes >$.

Étant donnés deux graphes $G_1 = (V_1, E_1)$ et $G_2 = (V_2, E_2)$, nous définissons les ensembles L_V et L_E d'étiquettes de sommets et d'arcs et les deux graphes multi-étiquetés $G'_1 = \langle V_1, r_{V1}, r_{V2}\rangle$ et $G'_2 = \langle V_2, r_{V2}, r_{E2}\rangle$ suivants :

$$\begin{aligned} L_V = \{l_{(u,v)}|u \in V_1, v \in V_2\} \quad &, \quad L_E = \{l_e\} \\ r_{V1} = \{(u, l_{(u,v)})|u \in V_1, v \in V_2\} \quad &, \quad r_{E1} = \{(u, v, l_e)|(u,v) \in E_1\} \\ r_{V2} = \{(v, l_{(u,v)})|u \in V_1, v \in V_2\} \quad &, \quad r_{E2} = \{(u, v, l_e)|(u,v) \in E_2\} \end{aligned}$$

Avec de tels graphes multi-étiquetés, l'ensemble $G'_1 \sqcap_m G'_2$ de caractéristiques communes relativement à l'appariement m contient toute l'information nécessaire pour reconstituer l'appariement m effectué :

$$m \quad = \quad \{(u,v)|\exists(u, l_{(u,v)}) \in G'_1 \sqcap_m G'_2\}$$

L'ensemble $G'_1 \sqcap_m G'_2$ étant son argument, la fonction f peut donc être définie de façon à reconstituer l'appariement m et calculer les valeurs des fonctions δ_v, δ_e et $\otimes$. Elle permet donc de calculer la distance $\delta =< \delta_v, \delta_e, \otimes >$. □

[6] Notons cependant que dans un cas l'objectif est de minimiser la distance et que dans l'autre cas, le problème est de maximiser la similarité. La fonction $\otimes$ doit donc être définie en conséquence, par exemple, en essayant de minimiser l'inverse de la similarité de Champin et Solnon.

3.3 Discussion

Nous avons proposé dans ce chapitre une mesure générique de la distance de deux graphes. Cette mesure a trois caractéristiques principales :

- elle est basée sur la recherche d'un meilleur appariement multivoque des sommets des graphes à comparer ;
- elle permet d'exprimer des contraintes dures et des préférences (individuelles) entre les arcs et les sommets appariés, la distance entre deux graphes étant alors une agrégation de ces préférences ;
- elle permet de comparer des graphes de tout type : étiquetés, valués, multi-étiquetés...

Nous montrons que notre mesure de la distance de deux graphes est générique. En effet, cette mesure est paramétrable et permet de modéliser les mesures de distance ou de similarité de graphes existantes. Les contraintes et les préférences imposées sur l'appariement recherché par ces mesures sont exprimées par la définition de deux fonctions de distance de sommets et d'arcs et par une fonction permettant d'agréger ces contraintes et ces préférences.

Nous avons montré comment définir ces fonctions pour modéliser le problème de l'isomorphisme de (sous-) graphe, celui du calcul de la distance d'édition de graphe (étendue ou non), celui de la recherche du plus grand sous-graphe commun (partiel ou induit) entre deux graphes, la mesure de similarité de Boeres *et al.* [BRB04] et la mesure de Champin et Solnon [CS03].

À l'exception de la mesure de Champin et Solnon, toutes les mesures de distance de graphes existantes sont modélisées par notre mesure générique comme une simple somme des distances entre les sommets et les arcs appariés entre eux.

Les mesures de distance ou de similarité de graphes existantes utilisent des formalismes très différents les uns des autres ce qui rend la comparaison de ces mesures difficiles. Avoir une mesure de distance générique facilite la comparaison de ces mesures. Elle offre également un cadre uniformisé permettant une définition plus aisée de nouvelles mesures de distance de graphes. Enfin, un même algorithme, capable de calculer cette distance générique, peut être utilisé pour calculer des mesures de similarité ou de distance de graphes très différentes.

Nous avons montré dans [SS05a, SS05b] que, comme notre mesure de la distance de deux graphes, la mesure de Champin et Solnon est générique. Nous avons également montré en section 3.2.4 que ces deux mesures étaient équivalentes dans le sens où elles permettaient l'expression des mêmes problèmes. Néanmoins, notre mesure de distance de graphes comporte deux avantages par rapport à la mesure de similarité de Champin et Solnon :

- Il est généralement plus simple d'exprimer un problème de comparaison de graphes dans notre formalisme que dans celui de Champin et Solnon. L'exemple donné en section 2.3.3 montre que la formulation de certains problèmes avec la mesure de Champin et Solnon est difficile. La similarité de Champin et Solnon évalue la qualité d'un appariement par rapport aux étiquettes de sommets et d'arcs que cet appariement permet de retrouver. Pour utiliser cette mesure de similarité, il est donc souvent nécessaire de créer de nombreuses étiquettes "artificielles" de sommets et d'arcs pour rendre possible l'expression de contraintes et de préférences sur la façon dont les sommets et les arcs doivent être appariés. *A contrario*, notre mesure de la distance de deux graphes exprime directement ces contraintes et ces préférences. Il est donc plus facile de formuler un problème de comparaison de graphes avec notre mesure ;
- Le modèle que nous proposons permet une résolution plus efficace des problèmes. Dans le formalisme de Champin et Solnon, certaines contraintes sémantiquement simples doivent parfois être exprimées de façon complexe en termes d'étiquettes à retrouver ou à ne pas retrouver. Notre formalisme permet d'exprimer directement ces contraintes et il n'est donc

pas nécessaire de comparer de grands ensembles d'étiquettes pour vérifier qu'un appariement respecte ces contraintes.

Deuxième partie

Algorithmes pour mesurer la distance/similarité de graphes

4

Complexité du problème et approches existantes pour sa résolution

Dans ce chapitre, nous faisons un état de l'art des méthodes proposées dans la littérature pour calculer les distances du chapitre précédent. Dans une première section, nous présentons la complexité théorique de chacun des problèmes. Nous présentons ensuite quelques méthodes complètes de résolution de ces problèmes (c.-à-d. des algorithmes qui garantissent l'obtention de la solution mais qui ont une complexité en temps exponentielle dans le pire des cas). Nous discutons ensuite de l'utilisation d'une méthode complète pour le calcul de notre distance générique. La dernière section est dédiée aux méthodes de recherches incomplètes (c.-à-d. de complexité polynomiale mais ne garantissant pas de trouver la solution optimale).

4.1 Complexité des problèmes d'appariement de graphes

Calculer les mesures de distance (ou de similarité) de graphes présentées au chapitre 2 repose sur un problème de recherche d'un meilleur appariement entre deux graphes. Généralement, il existe un nombre exponentiel d'appariements possibles des sommets de deux graphes, ce problème est donc un problème d'optimisation combinatoire.

De façon générale, calculer la distance entre deux graphes est plus difficile lorsque des préférences (et non des contraintes dures) sont exprimées sur l'appariement recherché. Les distances basées sur des appariements ne devant respecter que des contraintes dures sont généralement plus simples à calculer que les distances basées sur des appariements univoques devant respecter des contraintes souples (ou des préférences) qui sont elles mêmes plus simples à calculer que les distances basées sur des appariements multivoques de sommets.

4.1.1 Appariements sous contraintes dures

Isomorphisme de graphes

Dans le cas général, la complexité théorique du problème de l'isomorphisme de deux graphes n'est pas précisément établie : le problème est clairement dans NP mais on ne sait pas s'il est dans P ou s'il est NP-complet [For96] ; la classe des problèmes *isomorphe-complets* a donc été définie. Cependant, il a été montré que pour certains types de graphes (les graphes planaires [HW74], les arbres [AHU74], les graphes à degré borné [Luk82]. ..), le problème de l'isomorphisme de graphe a une complexité polynomiale.

Isomorphisme de sous-graphe, de sous-graphe partiel et généralisation de ces problèmes

Si la complexité théorique du problème de l'isomorphisme de graphes n'est pas parfaitement déterminée, le problème de l'isomorphisme de sous-graphe (induit ou partiel) est clairement un problème NP-complet [GJ79]. Il peut être vu comme une généralisation du problème de la recherche d'une clique de taille donnée. En pratique, ce problème est un problème plus difficile que l'isomorphisme de graphes et de nombreuses instances ne peuvent être résolues en un temps raisonnable. Régin montre dans sa thèse [Rég95] que le problème de l'isomorphisme de sous-graphe partiel est plus général et plus difficile que le problème de l'isomorphisme de sous-graphe induit.

La généralisation du problème de l'isomorphisme de sous-graphe (ou "approximate subgraph matching" [ZDD05]) est également un problème NP-complet : il permet de modéliser n'importe quel problème d'isomorphisme de sous-graphe (voir la section 2.2.1).

Comme pour l'isomorphisme de graphes, en restreignant le problème à certaines classes de graphes, ces problèmes peuvent devenir polynomiaux. Ainsi, dans le cas des arbres ou des graphes planaires [Epp99], le problème de l'isomorphisme de sous-graphe est polynomial.

4.1.2 Appariements univoques avec préférences

Les distances basées sur des appariements univoques et permettant d'approximer la structure des graphes (comme la distance d'édition de graphe ou la distance basée sur la recherche du plus grand sous-graphe commun) peuvent toutes être utilisées pour résoudre le problème de la recherche de la clique maximale dans un graphe. Par conséquent, tous ces problèmes sont des problèmes d'optimisation NP-difficiles.

Notons cependant qu'il existe des exceptions pour lesquelles ces problèmes ont une complexité (faiblement) polynomiale. Citons par exemple le cas où l'on cherche à comparer des graphes étiquetés tels que chaque étiquette est utilisée au plus une fois dans chacun des deux graphes. Si l'appariement recherché impose de n'apparier que des sommets et des arcs ayant la même étiquette, le problème devient trivial. Cette restriction à ces graphes très particuliers a été utilisée par Schenker *et al.* [SLBK04] pour calculer le plus grand sous-graphe commun à deux graphes représentant les mots présents dans une page HTML.

4.1.3 Appariements multivoques

Les distances de graphes basées sur des appariements multivoques comme la distance d'édition de graphes étendue [AFB03] ou la mesure de similarité de [CS03] sont plus générales que celles basées sur des appariements univoques avec préférences et donc également NP-difficiles. Cependant, l'espace de recherche de ces problèmes est beaucoup plus grand. Étant donnés deux graphes ayant respectivement n et m sommets, il existe $2^{n \times m}$ appariements multivoques possibles entre ces deux graphes alors que le nombre d'appariements univoques n'est "que" de $min(n, m)$!.

4.2 Approches complètes pour les problèmes d'appariement de graphes

Dans cette section, nous présentons quelques approches complètes pour la résolution de problèmes d'appariement de graphes. Ces algorithmes évaluent de façon exhaustive toutes les combinaisons de l'espace de recherche d'un problème. L'évaluation des combinaisons se fait en com-

binant énumération (la combinaison est construite et évaluée) et raisonnement (montrant qu'un ensemble de combinaisons possède certaines caractéristiques ne lui permettant pas de contenir de combinaison solution).

L'exploration des combinaisons de l'espace de recherche étant exhaustive, ces algorithmes garantissent de trouver la solution (si elle existe) des problèmes de satisfaction ou la solution optimale des problèmes d'optimisation. Comme les problèmes résolus ne sont généralement pas dans la classe P des problèmes de complexité polynomiale, la complexité en temps (et parfois en espace) de ces algorithmes est exponentielle dans le pire des cas.

Nous avons choisi de découper cette section en deux sous-sections : d'une part les méthodes de résolution dédiées à un problème en particulier et d'autre part les méthodes issues de la programmation par contraintes (plus génériques). Nous comparons ensuite ces deux approches.

4.2.1 Approches dédiées

Recherches arborescentes

De nombreux algorithmes *ad-hoc* ont été proposés pour résoudre des problèmes d'appariement de graphes. La majorité de ces algorithmes sont basés sur une structuration en arbre de l'espace de recherche du problème couplée à des méthodes d'élagage de cet arbre de recherche.

Les problèmes à résoudre n'étant pas polynomiaux, la taille de l'arbre de recherche est potentiellement exponentielle par rapport à la taille des graphes à apparier. Par conséquent, cette technique n'est efficace que si elle est couplée d'une part à des méthodes de filtrage efficaces (des techniques prouvant le plus tôt possible qu'un nœud ne doit pas être développé) et d'autre part à un choix efficace de l'ordre dans lequel les nœuds doivent être développés (pour trouver les bonnes solutions le plus tôt possible). Cette méthode de structuration de l'espace de recherche est connue sous le nom de A^* ou "séparation/évaluation" ("branch and bound" en anglais).

L'algorithme proposé par Ullmann *et al.* [Ull76] (pour l'isomorphisme de sous-graphe), celui proposé par Cordella *et al.* [CFSV99, CFSV00, CFCS01] (pour les problèmes d'isomorphisme de (sous-)graphe(s)) ou celui proposé par Bunke *et al.* [NRB06] (pour la recherche du plus grand sous-graphe commun à deux graphes ou le calcul de la distance d'édition de graphes) sont tous basés sur ce type d'approche. Ils diffèrent sur la façon dont les nœuds de l'arbre sont filtrés et sur l'ordre dans lequel les sommets sont appariés (ainsi que sur les structures de données utilisées). Pour chaque problème, des méthodes de filtrage différentes des appariements sont utilisées.

Un algorithme de ce type est également possible pour la recherche d'un meilleur appariement multivoque. Ambauen *et al.* [AFB03] proposent une recherche de type A^* pour le calcul de la distance d'édition de graphes étendue. Les auteurs structurent l'espace de recherche en arbre où chaque nœud de l'arbre correspond à une opération d'édition du graphe. Chaque branche de l'arbre correspond donc à une chaîne d'édition de graphe. L'arbre de recherche est alors élagué en fonction des coûts des opérations d'édition réalisées et sur une estimation du coût des opérations devant encore être réalisées pour rendre les graphes isomorphes.

Recherche basée sur une représentation canonique des graphes

L'algorithme le plus efficace pour le problème de l'isomorphisme de deux graphes est Nauty [McK81]. Nauty (No AUTomorphism, Yes ?) est un algorithme permettant de représenter un graphe de façon canonique (c.-à-d. que deux graphes auront la même représentation si et seulement s'ils sont isomorphes). Par conséquent, Nauty permet de résoudre le problème de l'isomorphisme de deux graphes : il suffit de calculer la représentation canonique de chacun des deux graphes et de comparer (en un temps linéaire en la taille des graphes) ces représentations.

Le principe de Nauty consiste à trouver une partition ordonnée des sommets d'un graphe G dite "équitable", c.-à-d. telle que toute permutation des sommets de ce graphe respectant cette partition engendre un graphe G' isomorphe à G. Une telle partition ordonnée équitable permet de décrire les graphes indépendemment du nom de leurs sommets et constitue donc une représentation des graphes canonique par rapport à l'isomorphisme de graphes : il faut pour cela décrire le nombre de parties de la partition, leur taille et les relations de connexité entre les sommets des différentes parties.

Afin de trouver une partition ordonnée équitable, Nauty utilise un arbre de recherche explorant l'ensemble de toutes les partitions ordonnées possibles des sommets d'un graphe. Chaque nœud de l'arbre représente une partition ordonnée possible des sommets du graphe et chaque fils n' d'un nœud n représente une partition plus forte que n, c.-à-d. que n' utilise une partie de plus que n et que les sommets appartenant à une partie différente dans n appartiennent également à une partie différente dans n'. Des méthodes d'élagage de cet arbre permettant de ne pas explorer des partitions similaires à une partition déjà visitée sont implémentées afin de réduire le nombre de nœuds à développer. Finalement, comme il peut exister plusieurs partitions ordonnées équitables, seule la première partition trouvée est utilisée pour la représentation canonique du graphe. Il est donc nécessaire, pour tester l'isomorphisme de deux graphes, d'utiliser une méthode de développement de l'arbre de recherche totalement indépendante du nom des sommets, garantissant ainsi d'obtenir la même partition ordonnée équitable pour deux graphes isomorphes.

Nauty a une complexité exponentielle dans le pire des cas. Cependant, en pratique, il est extrêmement rapide et permet de comparer des graphes ayant plusieurs milliers de sommets en quelques secondes seulement. Darga *et al.* [DLSM04] proposent une adaptation de Nauty dédiée aux graphes creux (c.-à-d. peu denses) appelée Saucy. Puget [Pug05] propose un algorithme similaire nommé Autom et montre qu'il est bien plus rapide que Saucy (et Nauty).

Efficacité des approches complètes

De façon générale, les algorithmes de recherche complète *ad-hoc* permettent de trouver la solution des problèmes de satisfaction en un temps raisonnable. Par exemple, les problèmes d'isomorphisme de graphes ou de sous-graphe sont très efficacement résolus par des recherches arborescentes combinées à des méthodes de filtrage sophistiquées. Néanmoins, ces algorithmes montrent leur limites sur les problèmes d'optimisation (tels que la recherche du plus grand sous-graphe commun à deux graphes) et sont quasiment inutilisables pour les problèmes basés sur des appariements multivoques (voir [AFB03] où il est montré que ces méthodes sont limitées à de toutes petites instances, c.-à-d. moins de 10 sommets).

Le principal avantage des algorithmes de recherche complète *ad-hoc* est le fait qu'ils garantissent l'optimalité de la solution trouvée. En outre, ils ont été conçus et optimisés pour la résolution d'un type de problème en particulier et par conséquent, tirent au maximum partie des spécificités du problème pour filtrer l'espace de recherche et utilisent des structures de données parfaitement adaptées au problème.

Le principal défaut des méthodes de recherche complète *ad-hoc* est leur manque de généricité : leur efficacité dépend fortement des techniques de filtrage utilisées. Ces techniques dépendent du problème considéré et ne sont généralement pas adaptables à d'autres problèmes ou à un problème similaire auquel de nouvelles contraintes ont été ajoutées. L'efficacité de ces recherches peut également dépendre du type de graphe, il n'est pas rare de voir dans la littérature des algorithmes dédiés aux graphes très denses et d'autre dédiés aux graphes peu denses. Par exemple, Nauty [McK81] comprend de nombreux paramètres et l'efficacité de cet algorithme dépend très

fortement de ce paramétrage : selon le type de graphes considéré et le paramétrage choisi, le temps d'exécution de Nauty peut varier de plusieurs ordres de grandeur. En particulier, Puget [Pug05] montre que Nauty est efficace sur les graphes denses mais que sur les graphes creux il convient d'utiliser un algorithme dédié aux graphes creux tels que Autom [Pug05] ou Saucy [DLSM04].

4.2.2 Programmation par contraintes

Une alternative intéressante à ces algorithmes dédiés à un problème en particulier est l'utilisation de la programmation par contrainte (PPC) qui fournit un cadre générique pour la résolution des problèmes combinatoires.

En programmation par contraintes, le problème est exprimé de façon déclarative en termes de variables et de contraintes (dures ou souples) que les valeurs de ces variables doivent satisfaire. Un solveur générique de contraintes (c.-à-d. capable de résoudre n'importe quel problème représenté dans ce formalisme) tel que CHOCO [Lt00], Ilog solver [ILO00] ou CHIP [AB93] est alors utilisé pour trouver une affectation des variables permettant de satisfaire toutes les contraintes ou de minimiser un critère d'optimisation relatif aux contraintes souples violées. Les solveurs de contraintes sont basés sur une structuration en arbre de l'espace de recherche et sur des méthodes de filtrage de cet arbre basées sur les contraintes du problème (et donc indépendantes du problème à résoudre lui-même, contrairement aux approches dédiées).

Les problèmes d'appariement de graphes peuvent être aisément transformés en problèmes de satisfaction de contraintes et il est donc possible d'utiliser un solveur générique de contraintes pour les résoudre.

Problèmes de satisfaction de contraintes

Définition. Dans le cas des problèmes de satisfaction (c.-à-d. visant à satisfaire toutes les contraintes du problème), un CSP (Constraint Satisfaction Problem) est défini par un triplet (X, D, C) où :

- X est un ensemble fini de variables ;
- D est une fonction associant à chaque variable $x_i \in X$ son domaine $D(x_i)$, c.-à-d. l'ensemble des valeurs qui peuvent lui être associées ;
- C est un ensemble de contraintes, c.-à-d. de relations entre des variables restreignant l'ensemble des valeurs que les variables peuvent prendre simultanément.

Résoudre un CSP (X, D, C) consiste à trouver une affectation complète A (c.-à-d. affectant une valeur $v_i \in D(x_i)$ à chacune des variables $x_i \in X$) telle que toutes les contraintes de C sont satisfaites.

Les contraintes de l'ensemble C peuvent être de différentes formes. Par exemple, si on prend l'exemple de deux variables x_1 et x_2 ayant pour domaine l'ensemble d'entiers $\{1, 2, 3\}$, la contrainte $C(x_1, x_2) = x_1 < x_2$ imposera, en intention, que la valeur de la variable x_1 soit inférieure à celle de x_2. Cette même contrainte peut être définie en extension, c.-à-d. en listant l'ensemble des couples de valeurs autorisés pour le couple de variables (x_1, x_2) : $C(x_1, x_2) = \{(1, 2), (1, 3), (2, 3)\}$. L'arité d'une contrainte est le nombre de variables sur lesquelles la contrainte s'applique.

Exemples. Les problèmes d'appariement de graphes basés sur des appariements univoques sous contraintes dures peuvent être aisément représentés par des problèmes de satisfaction de contraintes.

Le problème de l'isomorphisme de sous-graphe partiel est représenté sous forme d'un CSP par Larrosa et Valiente [LV97]. Étant donnés deux graphes $G_1 = (V_1, E_1)$ et $G_2 = (V_2, E_2)$, le graphe G_1 est un sous-graphe partiel du graphe G_2 si et seulement si le CSP (X, D, C) suivant admet une solution :

- X, associe une variable à chaque sommet du graphe G_1, c.-à-d. que $X = \{x_u | u \in V_1\}$;
- $D(x_u)$, le domaine d'une variable x_u correspondant à un sommet u du graphe G_1 contient l'ensemble des sommets de G_2, c.-à-d. que $\forall u \in V_1, D(x_u) = V_2$;
- il existe une contrainte binaire $C_{edge}(x_u, x_v)$ entre chaque paire de variables (x_u, x_v) telle que $(u, v) \in E_1$ exprimant le fait que les sommets de G' affectés à x_u et x_v doivent être connectés par un arc, c.-à-d. que :

$$\forall (u, v) \in E_1, \quad C_{edge}(x_u, x_v) = E_2$$

Régin [Rég95] propose une modélisation du problème de l'isomorphisme de sous-graphe induit sous forme de CSP. Cette modélisation est la même que celle de l'isomorphisme de sous-graphe partiel à laquelle quelques contraintes sont ajoutées. Ces contraintes permettent de garantir que la solution trouvée ne correspond pas à un sous-graphe partiel, c.-à-d. que tous les arcs de G_2 dont les extrémités sont affectées à une variable du problème sont retrouvés dans G_1. Plus formellement, étant donnés deux graphes $G_1 = (V_1, E_1)$ et $G_2 = (V_2, E_2)$, le graphe G_1 est un sous-graphe induit du graphe G_2 si et seulement si le CSP (X, D, C) suivant admet une solution :

- X contient une variable pour chaque sommet du graphe G_1, c.-à-d. que $X = \{x_u | u \in V_1\}$;
- $D(x_u)$, le domaine d'une variable x_u correspondant à un sommet u du graphe G_1, contient l'ensemble des sommets de G_2, c.-à-d. que $\forall u \in V_1, D(x_u) = V_2$;
- Il existe une contrainte binaire $C_{edge}(x_u, x_v)$ entre chaque paire de variables (x_u, x_v) exprimant le fait que les sommets de G' affectés à x_u et x_v doivent être connectés par un arc si et seulement si $(u, v) \in E_1$, c.-à-d. que :

$$\begin{array}{lll} \forall (u, v) \in V_1^2, & \text{si } (u, v) \in E_1, & C_{edge}(x_u, x_v) = E_2 \\ & \text{sinon} & C_{edge}(x_u, x_v) = \{(u', v') \in V_2^2 \mid u' \neq v' \text{ et } (u', v') \notin E_2\} \end{array}$$

Problèmes d'optimisation sous contraintes

La programmation par contraintes a été initialement définie pour les problèmes de satisfaction. Elle peut néanmoins être utilisée pour des problèmes d'optimisation et permet donc de résoudre des problèmes d'appariement sous contraintes souples.

Une première solution pour résoudre des problèmes d'optimisation en programmation par contraintes est de résoudre le problème de satisfaction associé. L'objectif à optimiser est représenté par une variable du problème et des contraintes sont ajoutées afin de garantir la cohérence entre la valeur de cette variable et la satisfaction des contraintes du problème. Le problème est alors résolu en définissant le domaine de la variable objectif aux bornes minimum et maximum de la valeur de l'objectif (dans le cas d'un calcul de distance de graphe, le minimum est 0 et le maximum $+\infty$). Une fois une solution de ce CSP trouvée, la valeur v de la variable objectif x va être utilisée pour changer le domaine de x et le fixer à $[min, v[$ (dans le cas d'un problème de minimisation de l'objectif). La résolution de ce nouveau CSP est ensuite relancée jusqu'à ce que le CSP obtenu n'admette pas de solution. La valeur optimale est alors la valeur de la variable objectif dans la solution du dernier CSP résolu.

Pour résoudre des problèmes d'optimisation par programmation par contraintes, il est également possible d'utiliser le formalisme des CSP valués [BFM+96]. Ce formalisme permet de modéliser et de résoudre des problèmes d'optimisation : chaque contrainte (resp. chaque tuple

autorisé par une contrainte) se voit attribuer une valeur. L'objectif est alors de minimiser une fonction d'agrégation des valeurs des contraintes violées (resp. des tuples utilisés).

Définition. De façon plus formelle, un CSP valué valuant les contraintes est défini par un tuple $P = (X, D, C, S, \varphi)$ où :

- (X, D, C) est un CSP classique ;
- $S = (E, \otimes, \prec)$ est une structure de valuation, c.-à-d. que E est un ensemble de valeurs contenant un élément maximum $\top$ et un élément minimum $\bot$, que $\prec$ est une relation d'ordre totale sur E et que $\otimes$ est un opérateur commutatif, associatif et fermé dans E vérifiant les propriétés suivantes :
  - Identité : $\forall a \in E, a \otimes \bot = a$;
  - Monotonie : $\forall a, b, c \in E, (a \preceq b) \Rightarrow ((a \otimes c) \preceq (b \otimes c))$;
  - Élément absorbant : $\forall a \in E, (a \otimes \top) = \top$
- $\varphi : C \rightarrow E$ est une application associant une valeur de E à chaque contrainte.

La valeur $V_P(A)$ d'une affectation A des variables X d'un CSP valué P est définie par :

$$V_P(A) = \bigotimes_{c \in C, A \text{ viole c}} (\varphi(c))$$

Exemple. Le problème de la recherche du plus grand sous-graphe partiel commun à deux graphes peut être défini comme un CSP valué. Il faut pour cela utiliser une structure de valuation permettant la valuation des contraintes dures du problème (l'appariement doit nécessairement être univoque), et les contraintes souples (retrouver les arcs des deux graphes). De façon plus formelle, le CSP valué $P = (X, D, C, S, \varphi)$ correspondant au problème de la recherche du plus grand sous-graphe partiel commun à deux graphes $G_1 = (V_1, E_1)$ et $G_2 = (V_2, E_2)$ peut être défini par :

- $X = \{x_u, u \in V_1\}$;
- $\forall u \in V_1, D(x_u) = V_2 \cup \{none\}$ $x_v = none$ si v n'est pas apparié ;
- $C = HardC \cup SoftC$;
- $HardC = \{\forall (u, v) \in V_1^2, u \neq v \Rightarrow (x_u \neq x_v) \vee (x_u = x_v = none)\}$, l'ensemble des contraintes dures garantissant que l'appariement est univoque ;
- $SoftC = \{\forall (u, v) \in E_1, (x_u, x_v) \in E_2\}$, l'ensemble des contraintes souples permettant de maximiser le nombre d'arcs retrouvés dans les deux graphes ;
- $S = ([0, +\infty[, +, \leq)$;
- $\forall c \in HardC, \varphi(c) = +\infty$;
- $\forall c \in SoftC, \varphi(c) = 1$.

L'affectation des variables minimisant ce CSP valué correspond à l'appariement permettant d'identifier le plus grand sous-graphe partiel commun (en terme de nombre d'arcs) aux deux graphes.

De façon générale, le formalisme des CSP valués permet de modéliser les problèmes du calcul de la distance entre deux graphes. Les fonctions de distance des sommets et des arcs peuvent s'exprimer par des contraintes valuées dans l'ensemble $[0, +\infty[$ ordonné par $\leq$. Si la fonction d'agrégation $\otimes$ utilisée dans la définition de la distance respecte les conditions de monotonie, d'identité et possède un élément absorbant, les calculs de distance de graphes peuvent alors être modélisés par des CSP valués. Si la fonction $\otimes$ ne respecte pas ces propriétés, le calcul de la distance ne peut généralement pas être exprimé dans ce formalisme. Néanmoins, nous avons vu dans la première partie de cette thèse que toutes les distances/similarité que nous avons défini (à

l'exception de celle modélisant la similarité de Champin et Solnon [CS03]) utilisent l'opérateur d'agrégation $\otimes_{\sum}$ et respectent donc ces conditions.

Résolution des CSP

Les solveurs de problèmes de satisfaction de contraintes sont basés sur l'exploration complète de l'espace de recherche par construction d'un arbre de recherche et sur la réduction de cet espace par des techniques de filtrage basées sur la propagation des contraintes du problème.

Des méthodes de consistances locales permettent de filtrer le domaine des variables. Par exemple, la consistance d'arc (AC pour Arc Consistency ou 2-consistance [Mac77]) est définie pour les contraintes binaires (mais se généralise à des contraintes d'arité supérieure). Une contrainte $C(x_1, x_2)$ est arc-consistante si pour chaque valeur v_1 du domaine de x_1 (resp. v_2 de x_2), il existe au moins une valeur v_2 dans le domaine de x_2 (v_1 dans x_1) tel que le tuple (v_1, v_2) est autorisé par la contrainte C. Ainsi, si la variable x_1 a pour domaine l'ensemble $\{1, 2, 3\}$, que la variable x_2 a pour domaine l'ensemble fini $\{1, 3\}$ et que la contrainte $x_1 = x_2$ appartient au réseau de contraintes, la consistance d'arc permet de retirer la valeur 2 du domaine de x_1 puisque cette valeur n'admet aucun "support" dans le domaine de x_2. On parle alors de propagation de contraintes.

Un réseau de contraintes est arc-consistant si chaque contrainte de ce réseau est arc-consistante. De nombreux algorithmes ont été développés afin de maintenir l'arc-consistance d'un réseau de contraintes par exemple AC1 [Mac77, MF85], AC4 [MH86], AC7 [BFR99].

Notons que les méthodes de consistance utilisées sont locales : elles ne considèrent que quelques variables (2 pour l'arc consistance) et les contraintes s'y rattachant. Par conséquent, elles ne permettent pas, dans le cas général, d'obtenir une consistance globale (c.-à-d. un filtrage permettant de retirer des domaines toutes les valeurs qui n'appartiennent à aucune solution du problème). Il est toutefois possible de considérer des consistances de niveau plus élevé que la 2-consistance (considérant alors plus de deux variables en même temps). Néanmoins, garantir de telles consistances est plus coûteux en temps et le meilleur compromis efficacité / coût est généralement obtenu par l'arc consistance.

Importance de la modélisation et contraintes globales

La résolution des problèmes d'isomorphisme de sous-graphe peut être facilitée en introduisant de nouvelles contraintes explicitant les propriétés particulières du problème modélisé. Par exemple, il est facile de démontrer qu'un sommet de $G_1 = (V_1, E_1)$ ne peut être apparié qu'à un sommet du graphe $G_2 = (V_2, E_2)$ de degré inférieur ou égal dans un appariement d'isomorphisme de sous-graphe entre G_1 et G_2. Il est possible de réduire en conséquence le domaine des variables de ce CSP :

$$\forall u \in V_1, D(x_u) = \{u' \in V_2 \mid \begin{array}{l} |\{(u,v) \in E_1\}| \leq |\{(u',v') \in E_2\}| \text{ et} \\ |\{(v,u) \in E_1\}| \leq |\{(v',u') \in E_2\}|\} \end{array}$$

Cette réduction permet d'accélérer la recherche de façon significative.

Bien que les modélisations proposées ne l'explicitent pas, les solutions de ces CSP affecteront des valeurs différentes à chacune des variables du problème (l'appariement recherché est univoque). Par conséquent, Régin [Rég95] propose d'ajouter une contrainte "globale" (la contrainte *allDiff(X)*) exprimant le fait que toutes les variables du problème doivent avoir une valeur différentes deux à deux.

De façon informelle, "une contrainte C est souvent appelée "globale" quand "traiter" C comme un tout donne de meilleurs résultats que "traiter" n'importe qu'elle conjonction de contraintes

"sémantiquement équivalente" à C" [BH03a]. La définition formelle d'une contrainte globale est sujette à controverse [BH03b]. Néanmoins, comme le soulignent Bessière et Van Hentenryck, une contrainte peut être globale de 3 façons différentes : sémantiquement, opérationnellement et algorithmiquement. Une contrainte est sémantiquement globale lorsqu'il n'existe pas de décomposition de cette variable en un ensemble de contraintes d'arités inférieures : ce n'est pas le cas de la contrainte *allDiff(S)* sémantiquement équivalente à une clique de contraintes binaires d'inégalité des variables de S. Une contrainte est opérationnellement globale lorsqu'elle permet un filtrage plus important que n'importe laquelle de ses décompositions. Enfin, une contrainte est algorithmiquement globale quand appliquer un algorithmique de filtrage sur cette contrainte est algorithmiquement moins coûteux que de l'appliquer sur sa décomposition.

Utiliser des contraintes opérationnellement ou algorithmiquement globales permet aux solveurs de contraintes d'être plus efficaces. Dans sa thèse, Régin [Rég95] propose un algorithme de filtrage basé sur un calcul de flot dans un graphe bipartie pour la contrainte *allDiff* et montre que les solveurs de contraintes sont plus efficaces en utilisant ce filtrage qu'en utilisant un ensemble de contraintes binaires spécifiant que les variables doivent être différentes deux à deux.

Pour que la programmation par contraintes soit compétitive, il est nécessaire de modéliser intelligemment le problème et d'utiliser au maximum les contraintes globales existantes : un problème mal modélisé peut ne pas être résolu en un temps raisonnable par la PPC alors qu'il l'aurait été avec une bonne modélisation.

Efficacité de cette approche pour les problèmes d'appariement de graphes

La programmation par contraintes permet d'exprimer très simplement un problème : le problème à résoudre est exprimé sous la forme d'un réseau de contraintes et, quel que soit le CSP obtenu, les solveurs de contraintes disposent de méthodes de résolution génériques permettant de le résoudre. La programmation par contraintes offre donc une souplesse d'utilisation que n'ont pas les approches dédiées.

En contrepartie de cette généricité et de cette simplicité d'utilisation, la programmation par contraintes est souvent moins efficace que des méthodes dédiées. Par exemple, de nombreuses instances du problème de l'isomorphisme de deux graphes sont résolues en un temps polynomial par des algorithmes *ad-hoc* alors que les solveurs de contraintes ont un comportement exponentiel sur ces instances. Il en est de même pour les problèmes plus difficiles (et dont on est sûr de la difficulté en théorie). En outre, si trouver une modélisation d'un problème en programmation par contraintes est simple, il est parfois difficile d'obtenir une modélisation efficace, c.-à-d. une modélisation qui permet à un solveur de contraintes de résoudre efficacement le problème.

Les problèmes basés sur des appariements à tolérance d'erreurs qui sont des problèmes d'optimisation ne peuvent pas être résolus en un temps raisonnable par des solveurs de contraintes. En effet, si la programmation par contraintes s'avère parfois aussi efficace que des méthodes dédiées sur des problèmes de satisfaction de contraintes, sur les problèmes d'optimisation, elle n'est généralement pas compétitive par rapport aux méthodes dédiées qui savent utilisées au mieux les spécificités d'un problème pour filtrer son espace de recherche. En outre, dans le cas de la recherche d'un meilleur appariement multivoque, la taille du problème d'optimisation de contraintes explose par rapport à la taille des graphes comparés. Pour comparer deux graphes $G_1 = (V_1, E_1)$ et $G_2 = (V_2, E_2)$, il faut $|V_1 \times V_2|$ variables binaires (une par couple de sommets possible). Pour les instances que l'on souhaite résoudre (des graphes ayant une centaine de sommets), le problème correspondant se compose donc d'une dizaine de milliers de variables ce qui dépasse largement les capacités d'un solveur de contraintes.

4.2.3 Approche complète de calcul de notre distance

Lors de mon stage de maîtrise [SCS03], nous avons proposé un algorithme complet pour le calcul de la similarité de Champin et Solnon. Néanmoins, cet algorithme est aisément adaptable au calcul de notre distance générique de deux graphes.

Principe de base

Cet algorithme est de type "séparation et évaluation" ("branch and bound"). L'espace de recherche (c.-à-d. l'ensemble de tous les appariements possibles entre deux graphes) est structuré en treillis et exploré de façon complète. Afin de limiter l'explosion combinatoire de la recherche, des techniques de filtrage sont mises en œuvre afin de "couper" les sous-ensembles du treillis dont il a été démontré qu'ils ne pouvaient pas contenir la solution.

L'espace de recherche du problème de la similarité maximum entre deux graphes est composé de tous les appariements multivoques possibles des deux graphes à comparer. Celui-ci peut être structuré en treillis par rapport à la relation d'inclusion : on note n_m un nœud du treillis associé à un appariement m ; chaque nœud n_m a pour fils l'ensemble des nœuds $n_{m'}$ tels que m' est un appariement contenant un couple de sommets de plus que m.

Les fonctions f et g de la similarité de Champin et Solnon sont des fonctions monotones croissantes par rapport à l'inclusion. Cela nous a permis de proposer deux différentes façons d'élaguer le treillis de recherche. La première identifie des couples de sommets "inutiles" (c.-à-d. des couples dont l'ajout ne peut que dégrader l'appariement courant) afin de ne pas développer les appariements contenant un de ces couples inutiles. La seconde méthode identifie les étiquettes pour lesquelles il n'existe plus de couple de sommets (non inutile) permettant de les retrouver. Identifier ces étiquettes permet un calcul plus juste de la borne maximum de la qualité des appariements et, par conséquent, permet d'élaguer plus rapidement le treillis.

Résultats obtenus

Les résultats obtenus par cet algorithme complet ont montré [SCS03] qu'une exploration par "séparation et évaluation" n'est envisageable que pour de tous petits graphes (une dizaine de sommets maximum dans le cas général) et que son efficacité dépend énormément de l'influence du poids des éclatements de sommets par rapport aux poids des caractéristiques des sommets et des arcs des graphes.

L'inefficacité de cette approche vient du fait que, dans le cas général, la fonction de similarité à optimiser n'est pas monotone par rapport à l'inclusion des appariements. Il n'est donc pas possible de trouver des méthodes de filtrage efficaces de cet espace de recherche.

Dès lors, utiliser des algorithmes de recherche incomplète, c.-à-d. qui ne garantissent pas l'optimalité de la solution trouvée mais de complexité faiblement polynomiale semble être une bonne alternative.

4.3 Approches incomplètes pour les problèmes d'appariement de graphes

La complexité d'un problème est définie par rapport à la complexité du meilleur algorithme complet et correct (c.-à-d. permettant de trouver pour chaque instance possible du problème sa solution si elle existe ou de montrer l'inexistence de cette solution). Les problèmes d'appariement de graphes étant NP-complets ou NP-difficiles, tous les algorithmes garantissant la complétude

de la solution ont une complexité en temps exponentielle dans le pire des cas (sauf si la conjecture $P \neq NP$ n'est pas vérifiée). Par conséquent, comme nous l'avons vu dans la section précédente, sur les problèmes difficiles, les algorithmes complets ne peuvent être exécutés que sur de petites instances.

Un autre type d'approche pour la résolution de ce type de problèmes consiste à explorer l'espace de recherche de façon incomplète. Les algorithmes de ce type ne garantissent pas de trouver la solution si elle existe mais ont une complexité plus faible que leur pendant complet (généralement la complexité est polynomiale). Par conséquent, en un temps polynomial par rapport à la taille du problème, ce type d'algorithme peut parfois trouver la solution d'un problème de satisfaction de contraintes. En revanche, si aucune solution n'est trouvée, cela ne signifie pas que le problème n'admet pas de solution. Pour les problèmes d'optimisation, la solution retournée par ce type d'algorithme est peut-être la solution optimale mais rien ne le garantit : l'objectif de ce type d'approche est de trouver une "bonne" solution en un temps raisonnable.

La perte de la complétude peut paraître gênante. Néanmoins, dans de nombreux cas, les approches incomplètes trouvent les solutions optimales des instances également résolues par les approches complètes et obtiennent une solution approchant l'optimum en un temps raisonnable là où les approches complètes ne permettent pas d'obtenir de solution en un temps raisonnable.

Dans cette section nous présentons (quelques unes) des méthodes incomplètes de calcul de la distance de deux graphes. Chacune de ces méthodes est remise dans son contexte d'utilisation afin de montrer pourquoi, dans ce contexte, l'optimalité de la solution retournée n'est pas indispensable.

4.3.1 Principe général

Les approches incomplètes considèrent des problèmes d'optimisation combinatoire : dans le cas des problèmes de satisfaction, elles vont chercher à maximiser la satisfaction du problème. L'espace de recherche d'un problème d'optimisation combinatoire est un couple (S, f) où S est l'ensemble des combinaisons possibles pour ce problème (par exemple tous les appariements possibles entre deux graphes) et f une fonction de coût associant un nombre réel à chaque combinaison de S. Une solution du problème est une "meilleure" combinaison de S par rapport à f (celle qui maximise ou minimise f).

Exploration partielle de l'espace de recherche

Contrairement aux recherches complètes qui élaguent des parties de l'espace de recherche (en *démontrant* qu'elles ne peuvent pas contenir de solutions) et explorent intégralement les parties restantes, les recherches incomplètes n'explorent que partiellement l'espace de recherche en se concentrant sur les parties de l'espace qui leur semblent plus prometteuses.

Deux types différents de recherches incomplètes existent : les recherches basées sur la construction itérée de combinaisons et les recherches basées sur la modification itérée de combinaisons. Les recherches basées sur la construction (algorithmes gloutons, algorithmes à base de colonies de fourmis, EDA...) construisent des combinaisons en ajoutant incrémentalement à une combinaison courante des composants de combinaison (par exemple en instanciant une à une les variables d'un CSP ou en ajoutant des couples de sommets à un appariement). Les composants ajoutés sont sélectionnés par rapport à une heuristique évaluant localement l'intérêt des composants. L'ajout de composant se fait jusqu'à l'obtention d'une combinaison complète, (par exemple lorsque toutes les variables d'un CSP sont instanciées ou lorsqu'aucun ajout d'un couple de sommets ne permet d'améliorer l'appariement courant).

A contrario, les recherches basées sur la modification itérée de combinaisons (recherches locales, algorithmes génétiques...), explorent l'espace de recherche en se déplaçant d'un point à l'autre de cet espace. Chaque mouvement dans cet espace se fait en modifiant partiellement la combinaison courante (par exemple en modifiant la valeur d'une variable d'un CSP) et les mouvements effectués tentent de se diriger vers les points de l'espace de recherche prometteurs (toujours vis-à-vis d'une heuristique).

Paysage de recherche, optimum local

Les méthodes de recherche basées sur la modification itérée de combinaisons structurent l'espace de recherche (S, f) en terme de voisinage : deux combinaisons sont voisines lorsqu'il est possible d'aller de l'une à l'autre en une modification. L'espace de recherche ainsi structuré est appelé *paysage de recherche*.

Un optimum local dans un paysage de recherche est une combinaison optimale par rapport à la notion de voisinage, c.-à-d. une combinaison meilleure que toutes ses combinaisons voisines. Notons qu'un optimum local n'est pas nécessairement globalement optimal. La solution d'un problème est appelée "optimum global".

Intensification vs. diversification

L'efficacité des recherches incomplètes dépend de l'équilibre entre l'intensification et la diversification de la recherche. Intensifier la recherche consiste à explorer de façon minutieuse certains sous-espaces de l'espace de recherche qui semblent plus prometteurs vis-à-vis de l'heuristique utilisée. *A contrario*, diversifier la recherche consiste à aller vers les points de l'espace de recherche qui semblent a priori moins prometteurs par rapport à l'heuristique utilisée et ainsi explorer de façon plus large l'espace de recherche d'un problème.

Lorsqu'une recherche est trop intensifiée, l'algorithme est incapable de sortir des optima locaux. En effet, quel que soit l'algorithme incomplet utilisé, si la recherche est trop intensifiée, une fois qu'un optimum local est trouvé, l'algorithme n'explorera que des combinaisons proches de cet optimum local (c.-à-d. que des combinaisons appartenant au même sous-espace de recherche) et ne peut généralement pas trouver l'optimum global. Au contraire, lorsque la recherche est trop diversifiée, l'algorithme ne se dirige pas du tout vers les combinaisons prometteuses et ne trouve pas les bonnes combinaisons.

4.3.2 Approches gloutonnes

Un algorithme glouton est un algorithme qui construit incrémentalement une combinaison en ajoutant itérativement à la combinaison courante des composants de solution choisis en fonction de l'intérêt immédiat de l'ajout de ce composant à la solution courante. En cherchant à optimiser ce critère local à chaque étape de la construction, un algorithme glouton espère obtenir l'optimum global du problème. Les constructions gloutonnes sont itérées et il s'agit de diversifier la recherche afin ne pas obtenir les mêmes combinaisons d'une construction à une autre.

Les méthodes gloutonnes régulent la diversification de la recherche en fonction de la façon dont les composants de solution sont choisis à chaque étape. Si à chaque étape le composant ajouté est choisi parmi ceux qui maximisent le critère à optimiser, la recherche est très intensifiée. Si au contraire le composant est choisi selon une probabilité qui dépend de la qualité du composant, la recherche est plus diversifiée. Si les composants de solutions sont choisis de façon purement aléatoire, la recherche est extrêmement diversifiée.

Un algorithme glouton pour le calcul de la similarité de Champin et Solnon est proposé dans [CS03, SCS03]. Cet algorithme démarre d'un appariement vide (c.-à-d. ne contenant aucun couple de sommets) et ajoute à chaque étape un couple de sommets parmi ceux dont l'ajout maximise la similarité. La construction de l'appariement est interrompue quand aucun ajout de couple de sommets ne permet d'augmenter la similarité. Cet algorithme permet de calculer très rapidement une borne minimum de la similarité de deux graphes. Cette borne est utilisée par un algorithme de recherche de type "séparation/évaluation" pour élaguer un arbre de recherche.

Un algorithme similaire est proposé par Boeres *et al.* [BRB04] pour le problème de l'appariement non-bijectif de deux graphes. Cet algorithme a pour objectif de construire un appariement valide (c.-à-d. respectant les contraintes dures du problème) servant de point de départ à une recherche locale. À chaque itération, un nouveau couple de sommets est choisi de façon aléatoire parmi ceux qui permettent de satisfaire le plus grand nombre de contraintes dures du problèmes. L'algorithme est itéré jusqu'à l'obtention d'un appariement valide, cet appariement est alors utilisé comme le point de départ d'une recherche locale.

Notons que ces deux algorithmes gloutons introduisent de l'aléatoire dans le choix des couples de sommets à ajouter. Par conséquent, d'une exécution à l'autre de ces algorithmes, l'appariement retourné n'est (dans le cas général) pas toujours le même et ces algorithmes peuvent être exécutés plusieurs fois afin de conserver le meilleur appariement trouvé.

4.3.3 Approches inspirées des recherches complètes

Dans [NRB06], les auteurs soulignent le fait que les algorithmes de résolution de problèmes d'optimisation basés sur l'exploration d'un arbre de recherche trouvent généralement la combinaison optimale du problème rapidement puis passent ensuite un temps très long à développer l'arbre de recherche pour montrer que le problème n'admet pas de meilleure combinaison.

Les auteurs de [NRB06] proposent donc deux algorithmes permettant d'exploiter ce phénomène pour le calcul de la distance d'édition entre deux graphes. Chacun de ces deux algorithmes repose sur l'exploration d'un arbre de recherche où chaque nœud correspond à la substitution (identique ou non-identique) d'un sommet d'un graphe par un sommet de l'autre graphe. Notons que dans ce cas précis, chaque chemin de la racine à un nœud de l'arbre correspond à une chaîne d'édition : il suffit de compléter (en un temps polynomial) les opérations de substitution déjà effectuées par des suppressions et des insertions d'arcs et de sommets pour obtenir la chaîne d'édition rendant les graphes isomorphes.

Le premier algorithme consiste à dérouler l'arbre de recherche de façon partielle : à chaque étape de l'algorithme, les k meilleurs nœuds de l'arbre sont mémorisés. Seuls ces k meilleurs nœuds sont développés[7]. La recherche est donc incomplète mais converge généralement vers de bonnes solutions.

Le second algorithme élague plus rapidement l'arbre de recherche que l'algorithme complet en surévaluant la qualité des nœuds des "grandes" solutions (celles contenant le plus de substitutions, les plus profondes dans l'arbre de recherche). Cette surévaluation à deux effets : (1) l'algorithme a plus souvent tendance à développer les nœuds les plus bas dans l'arbre (c.-à-d. ceux qui font le plus de substitutions et donc le moins de suppressions et d'insertions) et converge plus rapidement vers les bonnes solutions ; (2) les branches les moins prometteuses sont plus rapidement élaguées.

Bien que l'algorithme proposé surévalue souvent la distance (exacte) entre deux graphes, les auteurs montrent que généralement, la distance entre deux graphes très similaires n'est que peu

[7] Notons qu'en pratique, le nombre de nœuds développés est largement supérieur à k car au fur et à mesure que l'arbre est développé, tous les nouveaux nœuds ayant une qualité meilleure que des nœuds déjà développés sont découverts et insérés dans la liste des k meilleurs nœuds.

perturbée par cette surévaluation. Par conséquent, comme la distance entre deux graphes est utilisée pour faire de la classification, l'erreur de calcul sur la distance entre deux graphes de la même classe est faible et la classification n'est pas perturbée par le fait que le calcul de la distance ne soit pas exact.

Ces algorithmes sont bien plus rapides qu'une recherche arborescente complète, mais, contrairement aux autres méthodes citées dans cette section, ils gardent une complexité exponentielle dans le pire des cas. Néanmoins, le temps d'exécution de ces méthodes peut être contrôlé en choisissant un nombre de meilleurs nœuds à développer faible ou en surévaluant de façon importante la qualité des appariements.

Le nombre de meilleurs nœuds à explorer et la façon dont la qualité des nœuds est surestimée permettent de réguler intensification et diversification de la recherche. En choisissant de surestimer fortement la qualité des nœuds ou en limitant fortement le nombre de nœuds explorés, la recherche est très intensifiée. *A contrario*, en surestimant faiblement la qualité des nœuds ou en augmentant le nombre de nœuds à explorer, la recherche est plus diversifiée.

4.3.4 Algorithmes à estimation de distribution (EDA)

Les algorithmes à estimation de distribution (EDA) sont des algorithmes évolutifs basés sur l'amélioration d'une population d'individus (c.-à-d. un ensemble de combinaisons) de génération en génération. Contrairement aux algorithmes génétiques où l'évolution d'une population se fait par croisement des meilleurs individus et mutation, les EDA utilisent un modèle probabiliste pour la génération des nouveaux individus. À chaque étape, un modèle probabiliste analysant l'influence des variables les unes par rapport aux autres est créé à partir des meilleurs individus de la population, puis cette population est remplacée par des nouveaux individus générés à partir de ce modèle probabiliste.

Bengoetxea *et al.* [Ben02, BLB+02] proposent un algorithme à estimation de distribution (EDA) pour la résolution du problème de l'appariement non bijectif de graphes.

Il existe plusieurs façons de réguler diversification et intensification d'une recherche de type EDA. Le modèle probabiliste choisi pour la génération des individus et le nombre de meilleurs individus conservés à chaque génération permet d'intensifier plus ou moins fortement la recherche selon si la génération et la sélection est plus ou moins élitiste en fonction du critère à optimiser.

4.3.5 Recherche locale

Un nombre important de recherches incomplètes pour la résolution de problèmes combinatoires concerne des recherches locales. Contrairement aux méthodes citées ci-dessus, les recherches locales ne sont pas des approches qui construisent à chaque étape des combinaisons : l'espace de recherche est exploré à partir d'une combinaison initiale en modifiant légèrement à chaque itération la combinaison courante.

Principe de la recherche locale

La recherche locale [Glo89, KGV83] est une technique d'exploration opportuniste de l'espace de recherche d'un problème combinatoire. L'espace de recherche est structuré en termes de voisinage des différentes combinaisons possibles et l'algorithme explore l'espace de recherche, à partir d'une combinaison initiale (généralement calculée par un algorithme glouton), en se "déplaçant" à chaque itération sur un voisin de la combinaison courante. Une méta-heuristique de recherche locale est utilisée pour déterminer, à chaque itération, le prochain voisin à sélectionner.

Les algorithmes de recherche locale sont généralement incomplets (pour des versions complètes, voir [JL02, PV04]). À chaque itération, l'algorithme tente d'améliorer la qualité de la meilleure combinaison trouvée jusqu'alors.

Espace de recherche et voisinage

Une première étape lors de la conception d'un algorithme de recherche locale est de structurer l'espace de recherche en terme de voisinage des combinaisons, c.-à-d. de définir, étant donnée une combinaison, l'ensemble des combinaisons voisines. Cette structuration doit avoir certaines propriétés qui ne sont généralement pas indépendantes de l'heuristique du choix du prochain voisin à explorer :

1. La structuration choisie doit offrir un "chemin" (c.-à-d. une liste de combinaisons voisines) entre toutes les paires de combinaisons (c, c') de l'espace de recherche telles que c est une combinaison initiale (c.-à-d. un point de départ possible de la recherche locale) et c' est une solution. Cela permet de garantir que, indépendemment de l'heuristique du choix du prochain voisin, la structuration de l'espace en terme de voisinage permet à l'algorithme de converger vers les solutions du problème ;
2. Il est également souhaitable que le paysage de recherche soit tel que deux combinaisons voisines soient de qualité proches [BH99]. Cette propriété est importante car, comme leur nom l'indique, les recherches locales ont une vision très réduite de l'espace de recherche (généralement que les voisins directs de la combinaison courante). Cette propriété leur permet donc de se diriger vers les parties de l'espace de recherche qui semblent prometteuses tout en gardant une vision locale du problème ;
3. Il est souhaitable d'avoir une structuration telle que la fonction objectif soit calculable de façon incrémentale relativement à la définition du voisinage d'une solution. Cela permet de limiter le temps de calcul nécessaire à chaque itération.

Méta-heuristiques de recherche locale

À partir d'un appariement calculé par un algorithme glouton, la recherche locale explore l'espace de recherche de voisin en voisin. Une heuristique permettant de choisir à chaque itération le prochain voisin à visiter doit donc être définie. Il existe de nombreuses heuristiques de sélection du prochain voisin.

Hill Climbing ou montée de gradient. La plus simple est le "Hill climbing" qui consiste à choisir à chaque itération le meilleur voisin et à arrêter la recherche lorsque tous les voisins sont moins bons que la solution courante ou que le nombre d'itérations autorisé est atteint. L'heuristique de choix du prochain voisin est donc la même que celle d'un algorithme glouton. Selman *et al.* utilisent cette heuristique [SLM92, SKC93] dans l'algorithme GSAT de résolution de problèmes SAT (un problème de satisfaction d'un ensemble de formules booléennes). Le problème d'une telle recherche est qu'elle est extrêmement intensifiée et est donc incapable de sortir d'un optimum local. Par conséquent, afin d'être efficace, cet algorithme doit être relancé de nombreuses fois à partir de différents points de l'espace de recherche pour diversifier la recherche de façon plus importante. Pour plus de précisions, nous vous invitons à consulter l'article de Gent et Walsh à ce propos [GW93].

Afin qu'il puisse quitter les optima locaux, de l'aléatoire a été ajouté à GSAT (permettant ainsi de diversifier la recherche) [LMS02]. Lorsque l'algorithme est piégé dans un optimum local, la

combinaison courante est partiellement modifiée de façon aléatoire afin de relancer l'algorithme vers la recherche d'un autre optimum. La condition d'arrêt de l'algorithme est alors fixée en fonction du nombre de combinaisons visitées ou du nombre de fois où l'algorithme a été relancé en modifiant aléatoirement la combinaison courante.

Recuit simulé. Kirkpatrick *et al.* [KGV83] et Cerny [Cer85] proposent une heuristique appelée "recuit simulé" (Simulated annealing) en référence au comportement des atomes lors du processus de refroidissement des métaux visant à les solidifier. À chaque itération, un voisin est sélectionné aléatoirement. Celui-ci est systématiquement choisi lorsqu'il est meilleur que la combinaison courante. Si en revanche celui-ci dégrade la combinaison courante, il est choisi en fonction de sa qualité (la probabilité est d'autant plus élevée que la qualité est élevée) et de la température courante (plus la température est élevée, plus la probabilité est élevée et la température diminue au cours de la recherche). La recherche est interrompue lorsque le nombre maximum de mouvements autorisé est atteint. Intensification et diversification de la recherche sont régulées en fonction du choix de la température initiale et de la vitesse de refroidissement de la température au cours de la recherche.

Tabou. La recherche *Taboue* [Glo89, PKY02] est une des meilleures méta-heuristiques de recherche locale connues. À chaque étape, le "meilleur" voisin (au sens de la fonction objectif) est choisi. Notons que ce meilleur voisin peut-être moins bon que la solution courante. Par conséquent, pour éviter de rester autour d'un optimum local en se déplaçant sans cesse sur les mêmes solutions, une liste de mouvements tabous est utilisée. Cette liste, de longueur k, mémorise les k derniers mouvements réalisés (c.-à-d. les k dernières modifications faites pour passer d'une solution à une solution voisine) et il est interdit de réaliser un mouvement inverse à un mouvement de la liste taboue (c.-à-d. remettre en question une modification présente dans la liste Taboue). Une exception nommée "aspiration" est ajoutée : l'interdiction ne s'applique pas aux mouvements permettant d'atteindre une solution de meilleure qualité que la meilleure solution rencontrée depuis le début de la recherche.

Petrovic *et al.* [PKY02] proposent un algorithme de recherche locale pour le calcul d'un cas particulier de la distance d'édition entre deux graphes[8]. Le problème consiste à trouver un meilleur appariement univoque entre deux graphes appariant un maximum de sommets des deux graphes. Les auteurs justifient l'utilisation d'une méthode de recherche incomplète par le fait que la NP-complétude du problème limite les recherches complètes à de très petits graphes.

L'algorithme proposé est une recherche locale taboue. L'espace de recherche (c.-à-d. l'ensemble de tous les appariements univoques possibles entre les deux graphes) est structuré en terme de voisinage : les voisins d'un appariement m entre deux graphes G et G' sont les appariements qui peuvent être obtenus en inversant les sommets appariés à un couple de sommets de G ou en inversant l'appariement d'un sommet de G avec un sommet de G non apparié. À partir d'un appariement initial (généré de façon gloutonne), l'espace de recherche est exploré de voisin en voisin jusqu'à ce que le meilleur appariement rencontré ne semble plus améliorable (c.-à-d. si pendant les 100 dernières itérations aucune amélioration n'a pu être réalisée).

La longueur k de la liste taboue permet de réguler l'intensification de la recherche par rapport à sa diversification. Une liste taboue courte entraîne une forte intensification alors qu'une liste taboue longue entraîne une forte diversification.

[8] Les auteurs ne présentent pas le problème de cette façon. Néanmoins, leur problème peut être vu comme un problème de calcul d'une distance d'édition de graphes avec des poids particuliers.

Tabou réactif. À l'exception de la définition du voisinage d'une solution dans l'espace de recherche, le seul paramètre de la recherche locale taboue est la longueur de la liste taboue. Ce paramètre permet de réguler l'intensification de la recherche (c.-à-d. sa faculté à explorer minutieusement une partie de l'espace de recherche) par rapport à la diversification de la recherche (c.-à-d. sa faculté à se déplacer d'une partie à l'autre de l'espace de recherche). Les expérimentations sur les recherches taboues ont montré que la longueur de la liste taboue est un paramètre critique [GH97]. Un mauvais choix de la longueur dégrade considérablement les performances de la recherche locale et la longueur optimale de la liste taboue peut varier d'une instance d'un problème à une autre. Par conséquent, bien que la recherche locale taboue ne nécessite le réglage que d'un seul paramètre, elle n'est pas robuste dans le sens où un bon paramétrage doit être trouvé pour chaque instance à résoudre.

Pour pallier ce problème de paramétrage, Battiti et Protasi [BT94, BP01] proposent un algorithme de recherche locale taboue où la longueur de la liste est dynamiquement adaptée au cours de la recherche en fonction des besoins de diversification ou d'intensification de la recherche. Lorsque la recherche visite un état de l'espace de recherche qu'il a déjà visité, la liste est allongée afin d'augmenter l'effort de diversification. Si au contraire l'algorithme n'a pas revisité d'état précédemment visité depuis un certain nombre d'itérations (signe que la recherche est suffisamment diversifiée), la liste est raccourcie afin d'augmenter l'intensification de la recherche. Battiti et Protasi proposent cet algorithme pour la résolution du problème de la clique maximale. À notre connaissance, il n'existe pas d'algorithme de recherche locale taboue réactive appliquée aux problèmes d'appariement de graphes.

4.3.6 Optimisation par colonies de fourmis

Sammoud *et al.* proposent [SSSG06a][9] un algorithme d'optimisation par colonies de fourmis appelé `ANT-GM'06` (ANT-Graph Matching) pour le calcul de la similarité de graphes basée sur un appariement multivoque de Champin et Solnon [CS03].

Principe de fonctionnement

La méta-heuristique d'optimisation par colonies de fourmis (ACO, Ant Colony Optimization) est une approche bio-inspirée [DS05, DD99] fréquemment utilisée pour la résolution de problèmes combinatoires difficiles. Le paradigme consiste à modéliser le problème à résoudre en une recherche d'un meilleur chemin dans un graphe (appelé graphe de construction) et à utiliser des fourmis artificielles pour rechercher des "bons" chemins dans ce graphe. Le comportement de ces fourmis artificielles est inspiré des fourmis réelles : *(i)* les fourmis déposent de la phéromone pour marquer les chemins prometteurs du graphe de construction, *(ii)* elles se déplacent sur le graphe de construction en choisissant leur chemin avec une probabilité dépendant des traces de phéromones précédemment déposées, et *(iii)* la quantité de phéromone déposée sur les chemins décroît à chaque cycle de l'algorithme afin de simuler le phénomène d'évaporation de la phéromone observé dans la nature.

Algorithme

Principe général

[9] Je figure parmi les auteurs car cet article est une comparaison de l'algorithme à base de colonies de fourmis proposé par Olfa Sammoud et de la recherche locale taboue réactive que je propose d'utiliser.

À chaque cycle de l'algorithme, chaque fourmi de la colonie construit un appariement en partant d'un appariement initialement vide et en ajoutant itérativement de nouveaux couples de sommets. Le couple de sommets inséré à chaque étape est choisi selon une probabilité qui dépend des traces de phéromone précédemment déposées et d'une heuristique propre au problème considéré. Une fois que chaque fourmi a construit son appariement, une procédure de recherche locale est lancée afin d'essayer d'améliorer la qualité du meilleur appariement trouvé lors de ce cycle. Les traces de phéromone sont par la suite mises à jour en fonction de cet appariement amélioré. Ce processus est réitéré jusqu'à ce qu'une fourmi ait trouvé l'appariement optimal ou que le nombre maximum de cycles autorisé soit atteint.

Graphe de construction

Le graphe de construction est le graphe sur lequel les fourmis déposent de la phéromone. Les sommets de ce graphe sont les composants sélectionnés par les fourmis pour progressivement construire des solutions. Dans le cas de la recherche d'un meilleur appariement multivoque entre deux graphes, les fourmis construisent des appariements en sélectionnant un à un les couples de sommets à apparier. Étant donnés deux graphes multi-étiquetés $G = \langle V, r_V, r_E \rangle$ et $G' = \langle V', r_{V'}, r_{E'} \rangle$ à apparier, le graphe de construction est donc le graphe complet non-orienté contenant un sommet pour chaque couple de sommets $(u, u') \in V \times V'$.

Construction des appariements À chaque cycle, chacune des fourmis construit un appariement. En partant d'un appariement m vide ($m = \emptyset$), les fourmis ajoutent à chaque itération un couple de sommets à m choisi parmi l'ensemble des couples non encore sélectionnés $cand = \{(u, u') \in (V \times V') - m\}$. Comme de coutume dans les algorithmes ACO, le couple de sommets à ajouter à m est choisi selon une probabilité qui dépend des traces de phéromone et d'un facteur heuristique propre au problème considéré. Plus formellement, étant donné un appariement courant m et un ensemble de candidats $cand$, la probabilité $p_m(u, u')$ d'ajouter le couple de sommets (u, u') à m est définie comme suit :

$$p_m(u, u') = \frac{[\tau(u, u')]^\alpha \cdot [\eta_m(u, u')]^\beta}{\sum_{(v,v') \in cand} [\tau(v, v')]^\alpha \cdot [\eta_m(v, v')]^\beta} \tag{4.1}$$

où

- $\tau(u, u')$ est le facteur phéromonal. Lors du choix du premier couple, $\tau(u, u') = 1$, la probabilité ne dépend alors que du facteur heuristique ;
- $\eta_m(u, u')$ est le facteur heuristique, introduit pour favoriser les couples de sommets qui maximisent la fonction *score*, c.-à-d. que $\eta_m(u, u') = 1 + score(m \cup \{(u, u')\}) - score(m)$ si l'ajout de (u, u') ne fait pas décroître le *score*, et que $\eta_m(u, u') = 1$ sinon ;
- α et β sont deux paramètres qui déterminent l'importance relative des deux facteurs.

Les fourmis arrêtent leur construction quand tous les couples de sommets candidats font décroître le *score* de l'appariement ou quand les trois derniers ajouts n'ont pas permis d'accroître ce *score*.

Procédure de recherche locale. Pour de nombreux problèmes combinatoires, les algorithmes ACO les plus performants sont ceux qui combinent la construction de solutions par des fourmis avec de la recherche locale. `ANT-GM'06` a été amélioré en ajoutant une procédure de recherche locale simple offrant un bon compromis entre la qualité des solutions trouvées et son temps d'exécution. Cette recherche locale est appliquée sur le meilleur appariement trouvé lors de chaque cycle. Le voisinage d'un appariement m est l'ensemble des appariements obtenus en ajoutant ou en supprimant un seul couple de sommets à m. L'heuristique utilisée est gloutonne : à chaque itération, le meilleur voisin (au sens de la fonction *score*) est sélectionné. À l'obtention d'un

maximum local m, la recherche est diversifiée en supprimant les trois plus mauvais couples de sommets à m (au sens de la fonction *score*) et en interdisant leur rajout lors des itérations suivantes. Ce processus d'intensification et de diversification est réitéré tant qu'il permet d'améliorer l'appariement.

Mise-à-jour de la phéromone. Une fois que chaque fourmis a construit un appariement et que le meilleur appariement trouvé a été amélioré par recherche locale, les traces de phéromone sont mises à jour selon la stratégie *Max-Min Ant System*[SH00]. Toutes les traces de phéromone sont diminuées par un coefficient d'évaporation ρ ($0 \leq \rho \leq 1$). La meilleure fourmi du cycle dépose alors de la phéromone. Plus formellement, soit m_k le meilleur appariement (par rapport à la fonction *score*) trouvé durant le cycle et amélioré par la recherche locale et m_{best} le meilleur appariement trouvé depuis le début de la recherche (y compris le cycle courant), la quantité de phéromone déposée est inversement proportionelle à la différence entre le *score* de m_k et celui de m_{best} : elle est égale à $1/(1 + score(m_{best}) - score(m_k))$. Comme nous avons choisi de déposer la phéromone sur les sommets du graphe de construction, cette quantité de phéromone est ajoutée sur chaque sommet $< u, u' >$ tel que $(u, u') \in m_k$. Finalement, une borne minimale τ_{min} et une borne maximale τ_{max} sont imposées aux traces de phéromone (avec $0 < \tau_{min} < \tau_{max}$). Cela permet de minimiser les différences relatives entre les traces de phéromone et d'éviter une convergence prématurée de la recherche vers une région probablement sous-optimale.

Intensification vs. diversification

Chacun des paramètres de l'algorithme permet de réguler intensification et diversification de la recherche :

- plus le nombre de fourmis de la colonie est élevé et meilleure sera la diversification ;
- la façon de gérer la phéromone (taux d'évaporation, taux maximal, taux minimal, quantité de phéromone ajoutée à chaque cycle...) influent sur la diversification de la recherche car lorsque les différences entre les taux de phéromones sont élevées, la recherche est plus intensifiée ;
- une valeur de β élevée intensifie la recherche car elle augmente l'influence du facteur heuristique ;
- une valeur de α élevée diversifie la recherche (si les écarts de taux de phéromone sont faibles) car elle augmente la probabilité d'un couple moins intéressant au regard du facteur heuristique d'être sélectionné.

4.4 Discussion

Calculer la distance entre deux graphes est un problème difficile en théorie comme en pratique. La complexité en pratique du problème dépend néanmoins de la façon dont la distance est paramétrée : les problèmes basés sur des appariements univoques devant respecter des contraintes dures sont généralement plus faciles que les appariements univoques devant respecter des contraintes souples eux-mêmes plus simples que les problèmes basés sur des appariements multivoques.

Les approches complètes peuvent calculer efficacement les mesures de distance de graphes basées sur des contraintes dures. Néanmoins, ces performances ne peuvent être obtenues que grâce à la mise en œuvre de techniques *ad-hoc* performantes. Il est donc nécessaire d'utiliser des contraintes globales et leur algorithme de filtrage associé pour permettre à la programmation par contraintes d'être compétitive sur ce type de problèmes.

Les distances basées sur des appariements univoques avec des contraintes souples (recherche du plus grand sous-graphe commun à deux graphes, distance d'édition de graphe...) et sur des

appariements multivoques (distance d'édition de graphe étendue, similarité de Champin et Solnon...) ne peuvent être calculées par des approches complètes que pour des graphes de petites tailles (quelques dizaines de sommets seulement voire quelques sommets pour les appariements multivoques).

Pour résoudre des instances de ces problèmes de tailles plus importantes, il est possible d'utiliser des approches incomplètes. Ce type de méthodes ne garantit pas de trouver la solution d'un problème de satisfaction de contraintes ou de trouver la solution optimale d'un problème d'optimisation. En contrepartie de cette perte d'optimalité, ces algorithmes ont généralement une complexité faiblement polynomiale en la taille des graphes à comparer et permettent généralement de trouver en un temps raisonnable des "bonnes" solutions aux problèmes qu'une approche complète ne sait pas résoudre en un temps raisonnable.

Dans de nombreux domaines d'application, les graphes manipulés possèdent bien plus qu'une dizaine de sommets : il nous a donc été nécessaire d'explorer l'utilisation de méthodes de recherche incomplètes. Ce type de méthodes ne garantissant pas l'optimalité de la solution retournée, la distance de deux graphes sera parfois surévaluée par ce type d'algorithmes. Ce biais n'est pas rédhibitoire dans de nombreuses applications. En effet, la valeur exacte de la distance importe parfois peu : il suffit d'avoir une bonne approximation de la distance pour faire de la classification ou de la recherche de documents similaires. En outre, lors de la modélisation des documents en graphes ou lors de la définition de la distance entre deux documents, un biais est souvent introduit : la transformation en graphe n'est bien souvent pas réversible (il y a une perte d'information) et la définition "idéale" de la distance entre deux objets n'est parfois pas formellement exprimable. Par conséquent, dans de nombreux domaines, la notion d'optimalité n'a pas beaucoup de sens et avoir une approximation de la solution est suffisant.

Nous avons choisi de proposer un algorithme de recherche locale taboue réactive pour le calcul de notre distance générique de deux graphes. En effet, pour de nombreux problèmes, la recherche locale taboue est une méta-heuristique qui donne de très bons résultats et est relativement simple à implémenter. Comme nos expérimentations nous ont montré que la recherche taboue est parfois difficile à paramétrer, nous proposons une version réactive de cette algorithme capable de s'auto-paramétrer au cours de la recherche.

5

Recherche taboue réactive

Nous proposons dans ce chapitre un algorithme de recherche locale taboue réactive pour le calcul de notre distance générique entre deux graphes. Cet algorithme est incomplet, c.-à-d. qu'il ne garantit pas l'optimalité de l'appariement trouvé. Par conséquent, la distance entre les deux graphes peut être surévaluée. En contrepartie de cette perte d'optimalité, nous verrons qu'il permet de résoudre (parfois de façon optimale) des instances beaucoup plus grosses et beaucoup plus difficiles des problèmes d'appariement de graphes que les méthodes complètes le permettent.

Dans un premier temps, nous proposons un algorithme de recherche gloutonne du meilleur appariement entre deux graphes. Nous présentons cet algorithme, initialement proposé dans [CS03], car il est utilisé comme point de départ de notre algorithme de recherche locale. Après avoir vu comment une recherche locale peut être mise en œuvre pour la recherche d'un appariement multivoque optimal, nous proposons ensuite un algorithme de recherche locale taboue et une version réactive de cette recherche. Des résultats expérimentaux de ces algorithmes sur différentes classes de problèmes sont présentés.

5.1 Algorithme glouton

Dans [CS03], Champin et Solnon proposent un algorithme glouton utilisant une heuristique simple pour construire rapidement un appariement de "bonne" qualité pour le calcul de la similarité de deux graphes. Nous avons adapté cet algorithme au calcul de la distance de deux graphes.

L'algorithme glouton est décrit en figure 5.1. L'algorithme démarre avec un appariement m vide (ligne 1) et ajoute, à chaque itération (ligne 6), un couple de sommets dans m selon un principe glouton. On choisit le couple (u, u') à insérer parmi l'ensemble *cand* des couples qui minimisent la distance induite par $m \cup \{(u, u')\}$ (ligne 3). Le couple de sommets à ajouter est alors choisi aléatoirement parmi ceux de l'ensemble *cand* (ligne 5). Le processus d'ajout des couples de sommets est itéré tant que cet ajout permet de diminuer la distance entre les deux graphes (ligne 4).

Notons que l'algorithme glouton initialement proposé par Champin et Solnon proposait de départager les couples de sommets candidats en évaluant leur "potentiel" (c.-à-d. le nombre d'étiquettes d'arcs que le couple de sommets permet de retrouver). Nous avons choisi de retirer cette procédure d'évaluation fine de la qualité d'un couple de sommets car nos expérimentations ont montré qu'elle avait, dans le cas général, tendance à dégrader l'efficacité de l'algorithme glouton (en temps mais aussi en terme de qualité des résultats).

L'algorithme glouton a une complexité en $\mathcal{O}(c \times (|V| \times |V'|)^2)$, où c est la complexité de la

Algorithme Glouton
Entrée :2 graphes $G = (V, E)$ et $G' = (V', E')$
une distance δ telle que définie au chapitre 3
Sortie : un appariement $m \subseteq V \times V'$

```
1  m ← ∅
2  itérer
3      cand ← {(u1, u2) ∈ V×V' − m | δ_{m∪{(u1,u2)}}(G, G') est minimal}
4  sortir si ∀(u1, u2) ∈ cand, δ_{m∪{(u1,u2)}}(G, G') ≥ δ_m(G, G')
5      choisir aléatoirement un couple (u1, u2) dans cand
6      m ← m ∪ {(u1, u2)}
7  fin itérer
8  retourner m
```

FIG. 5.1 – Recherche gloutonne d'un appariement

mise à jour de la distance lors de l'ajout d'un couple de sommets à l'appariement. Notons que pour la majorité des définitions des distances que nous proposons dans la première partie de cette thèse, le coût c de cette mise à jour est constant si de bonnes structures de données sont utilisées. Par conséquent, dans la majorité des cas, la complexité de l'algorithme glouton est quadratique par rapport au nombre de couples de sommets possibles entre les deux graphes.

L'algorithme glouton n'est pas déterministe (les couples de sommets à ajouter sont choisis aléatoirement parmi l'ensemble des meilleurs). Par conséquent, cet algorithme peut être exécuté plusieurs fois afin de conserver le meilleur appariement trouvé.

Malgré sa simplicité, nous verrons dans la section résultats expérimentaux que l'algorithme glouton permet de résoudre efficacement des problèmes simples. Par exemple, les petites instances du problème de l'isomorphisme de sous-graphe peuvent être résolues assez rapidement par cet algorithme. L'algorithme glouton est capable de construire un grand nombre d'appariements différents en un très court temps et, sur les instances de petites tailles, a donc de bonnes chances de trouver l'appariement optimal en un temps relativement court.

Néanmoins, cet algorithme ne permet pas de résoudre efficacement les instances des problèmes d'appariement de graphes difficiles. En outre, comme cet algorithme ne fait jamais de retour arrière et est élitiste (à chaque itération, il ajoute un couple de sommets parmi ceux qui minimisent la distance), l'algorithme est très facilement piégé dans des appariements sous-optimaux. Par exemple, à la première itération, si le couple de sommets le plus prometteur n'est pas un couple de sommets appartenant à l'appariement optimal, celui-ci est tout de même systématiquement sélectionné et l'algorithme glouton ne trouve jamais l'appariement optimal.

Afin de pallier ce problème d'efficacité de l'algorithme glouton, nous proposons d'utiliser la recherche locale pour améliorer la qualité des appariements générés par Glouton.

5.2 Recherche locale taboue réactive

5.2.1 Voisinage des problèmes d'appariement

L'espace de recherche d'un problème de calcul de la distance de deux graphes $G = (V, E)$ et $G' = (V', E')$ se compose de tous les $2^{|V|\times|V'|}$ appariements multivoques possibles entre G et G'. Nous proposons la définition du voisinage suivant : les voisins $voisins(m)$ d'un appariement m sont les appariements obtenus en ajoutant ou en retirant un couple de sommets à m.

$$\forall m \subseteq V \times V', voisins(m) = \{m \cup \{(v,v')\} | (v,v') \in (V \times V') - m\} \\ \cup \{m - \{(v,v')\} | (v,v') \in m\}$$

Étant donné cette définition du voisinage d'un appariement, nous notons $diff(m,m')$ le couple de sommets ajouté ou supprimé à m pour obtenir son voisin m'. Plus formellement, étant donné un appariement $m \subseteq V \times V'$ et un appariement m' voisin de m, c.-à-d. tel que $m' \in voisins(m)$:

$$diff(m,m') = (m \cup m') - (m \cap m')$$

Nous avons vu au chapitre précédent que la structuration en voisinage d'un espace de recherche doit respecter 3 conditions (voir section 4.3.5). Quelle que soit la définition de la distance à calculer, le voisinage choisi permet à une recherche locale d'explorer l'intégralité de l'espace de recherche à partir de n'importe quel appariement initial. Il respecte donc la condition (1). Selon la définition de la distance entre deux graphes, les conditions (2) et (3) sont ou pas respectées. Notons cependant que dans tous les cas, la condition (3) est partiellement respectée dans le sens où les arguments des fonctions de distance des sommets et des arcs sont calculables de façon incrémentale par rapport à ce voisinage. En outre, pour toutes les définitions de distances de graphes correspondant aux problèmes d'appariement de graphes que nous avons présentés, les conditions (2) et (3) sont respectées. Finalement, le nombre de voisins d'un appariement n'excède pas $|V| \times |V'|$ (le nombre de couples de sommets possibles entre les deux graphes à comparer). Cette propriété est importante car l'algorithme tabou examine à chaque itération toutes les combinaisons voisines de la combinaison courante.

Notons que le voisinage peut être adapté en fonction du problème. En effet, si l'appariement recherché est univoque, il est possible de ne garder que les voisins qui sont des appariements univoques. Plus formellement, pour la recherche d'un appariement univoque, le voisinage $voisins_{uni}$ suivant peut être utilisé :

$$\forall m \subseteq V \times V', voisins_{uni}(m) = \{m \cup \{(v,v')\} | m(v) = 0 \wedge m(v') = 0\} \\ \cup \{m - \{(v,v')\} | (v,v') \in m\}$$

Ce voisinage respecte la condition (1) : il existe un chemin dans ce voisinage entre tout couple d'appariements univoques. En outre, le nombre de voisins moyen des appariements est fortement réduit ce qui accélère fortement le temps d'exécution d'une itération de tabou.

5.2.2 Algorithme tabou

À partir d'un appariement calculé par l'algorithme glouton, la recherche locale explore l'espace de recherche de voisin en voisin. Le prochain voisin à visiter est choisi selon le principe tabou.

La figure 5.2 présente l'algorithme tabou en détail. Cet algorithme admet deux paramètres : le nombre maximum de mouvements $nbMaxMouv$ à réaliser et la longueur k de la liste taboue. La recherche commence à partir d'un appariement calculé de façon gloutonne (ligne 1). À chaque itération, l'ensemble des appariements candidats est construit. Cet ensemble est composé de tous les appariements voisins de l'appariement courant $mCourant$ induisant une distance plus faible que le meilleur appariement connu $meilleurM$ (ligne 6) : on parle alors d'aspiration. Si cet ensemble est vide, les candidats sont les meilleurs appariements voisins atteignables par un

Algorithme Tabou
Entrée : 2 graphes $G = (V, E)$ et $G' = (V', E')$
une distance δ telle que définie au chapitre 3
une longueur de liste k et un nombre $nbMaxMouv$ de mouvements
Sortie : un appariement $m \subseteq V \times V'$

1 $mCourant \leftarrow Glouton(G, G', \delta)$
2 $meilleurM \leftarrow mCourant$
3 $nbMouv \leftarrow 0$
4 **tant que** $nbMouv < nbMaxMouv$ **faire**
5 $nbMouv \leftarrow nbMouv + 1$
6 $aspiration \leftarrow \{m' | m' \in voisins(mCourant) \wedge \delta_{m'}(G, G') < \delta_{meilleurM}(G, G')\}$
7 **si** $aspiration \neq \emptyset$ **alors**
8 $cand \leftarrow aspiration$
9 **sinon**
10 $cand \leftarrow \{m' | m' \in voisins(mCourant) \wedge nonTabou(k, diff(m, m'))$
$\wedge \delta_{m'}(G, G')$ est minimal$\}$
11 **fin si**
12 $ms \leftarrow$ sélection aléatoire d'un appariement de $cand$
13 rendre tabou le mouvement $diff(mCourant, ms)$
14 $mCourant \leftarrow ms$
15 **si** $\delta_{mCourant}(G, G') < \delta_{meilleurM}(G, G')$
16 $meilleurM \leftarrow mCourant$
17 **fin si**
18 **fin tant que**
19 **retourner** $meilleurM$

FIG. 5.2 – Recherche taboue d'un appariement

mouvement non tabou (ligne 10). Cet ensemble de candidat n'est pas forcément réduit à un singleton : l'appariement suivant est donc choisi de façon aléatoire parmi tous les candidats possibles (ligne 12). Le mouvement permettant d'aller de l'appariement courant à l'appariement suivant devient alors tabou (ligne 13). Le meilleur appariement connu $meilleurM$ est alors mis à jour si nécessaire (ligne 16). Une nouvelle itération est lancée tant que le nombre maximum de mouvements n'est pas réalisé (ligne 4-18).

Notons que la liste Taboue se comporte en fait comme une file : les premiers mouvements à entrer dans cette file sont également les premiers à en sortir. Finalement, afin que la recherche d'un couple de sommets dans la liste Taboue se fasse en temps constant, pour chaque couple de sommets, nous mémorisons la date (c.-à-d. le nombre de mouvements effectués depuis le début de la recherche) à laquelle ce couple de sommets a été le plus récemment ajouté ou supprimé. Ainsi, la présence ou pas d'un couple de sommets dans la liste Taboue se fait en comparant la date courante, la dernière date à laquelle le couple de sommets a été ajouté ou supprimé et la longueur de la liste Taboue.

5.2.3 Recherche tabou réactive

Intérêt de la réactivité

À l'exception de la définition du voisinage d'une solution dans l'espace de recherche, le seul paramètre de la recherche locale taboue est la longueur de la liste taboue. Nos expérimentations ont montré que sur les instances difficiles, ce paramètre était d'une part critique et d'autre part "fragile" (voir la section 5.3). En effet, nos expérimentations montrent que sur certaines instances de problèmes d'appariement de graphes, un mauvais paramétrage donne de mauvais résultats alors qu'un bon paramétrage permet de résoudre l'instance de façon optimale. Le comportement de l'algorithme est également très sensible aux variations de la longueur de la liste. Allonger la liste de une ou deux unités peut donner des résultats très différents sur une instance donnée et la longueur optimale de la liste varie de façon significative sur deux instances de même taille d'un même problème. Par conséquent, bien que l'algorithme ne nécessite le réglage que d'un seul paramètre, il n'est pas suffisamment robuste dans le sens où un bon paramétrage doit être trouvé pour chaque instance à résoudre.

Battiti et Protasi [BT94, BP01] proposent un algorithme de recherche locale taboue où la longueur de la liste est dynamiquement adaptée au cours de la recherche en fonction des besoins de diversification ou d'intensification de la recherche. Lorsque la recherche visite un état de l'espace de recherche qu'il a déjà visité, la liste est allongée afin d'augmenter l'effort de diversification. Si au contraire l'algorithme n'a pas revisité d'état précédemment visité depuis un certain nombre d'itérations (signe que la recherche est suffisamment diversifiée), la liste est raccourcie afin d'intensifier la recherche.

Détecter les redondances

Le processus de réactivité nécessite de savoir à chaque mouvement de la recherche taboue si l'appariement courant a déjà été visité auparavant. Mémoriser l'ensemble des appariements visités depuis le début de la recherche est très coûteux en mémoire. En outre, chercher à chaque étape de la recherche si l'appariement courant appartient à l'ensemble des appariements visités est coûteux en temps. Par conséquent, afin de réduire la complexité en temps et en espace de cette phase de détection des redondances dans la recherche, une table de hachage est utilisée. La table de hachage mémorise la clé de chaque appariement visité. Notons que seule la clé des appariements est mémorisée et que l'appariement lui-même n'est pas mémorisé. Si une collision intervient dans la table de hachage, et en supposant que notre fonction de hachage est bonne, il est raisonnable de penser que l'appariement courant a déjà été visité.

La fonction de hachage utilisée est très simple : chaque couple de sommets se voit attribuer au début de la recherche une puissance de 2 différente modulo la taille de la table de hachage. La clé de hachage d'un appariement est égale à la somme (modulo la taille de la table de hachage) des clés des couples de sommets qu'il contient. Notons que cette clé de hachage est calculable en un temps constant par rapport à la définition du voisinage d'un appariement : il suffit d'additionner (resp. de soustraire) la clé du couple de sommets ajouté (resp. enlevé) à la clé de l'appariement courant (modulo la taille de la table de hachage).

Finalement, afin de limiter les "fausses" collisions dues à une table de hachage trop petite tout en limitant l'espace mémoire nécessaire, plusieurs tables de hachages sont créées (en attribuant différentes puissances de 2 aux couples de sommets), et une redondance dans la recherche est détectée lorsqu'une collision est détectée dans chacune des tables. Nos expérimentations ont montré que lorsque 4 tables de hachages différentes sont utilisées et que leur taille est égale à 10 fois le nombre maximum de mouvements effectués, notre fonction de hachage ne semble détecter

aucune "fausse" collision.

Paramètres

En plus de la définition du voisinage, l'algorithme Tabou réactif nécessite plusieurs paramètres :

- la longueur minimale $kmin$ de la liste taboue, cette longueur minimale est également utilisée comme longueur initiale de la liste taboue ;
- la longueur maximale $kmax$ de la liste taboue ;
- la longueur $kstep$ de rallongement ou de raccourcissement de la liste taboue en cas de détection d'une redondance ou au contraire quand la recherche est suffisamment diversifiée ;
- le nombre de mouvements $kfreq$ au dessus duquel, s'il n'y a pas eu de redondance, il faut raccourcir la liste taboue.

Bien que tabou réactif comporte plus de paramètres que tabou, nous verrons dans la section 5.3 qu'il est généralement plus simple à paramétrer.

Algorithme détaillé

La figure 5.3 présente l'algorithme tabou réactif. Cet algorithme est similaire à l'algorithme tabou de la figure 5.2 : seules quelques lignes permettant de mettre à jour la longueur de la liste sont ajoutées. Au début de l'algorithme, la longueur k de la liste est initialisée à son minimum $kmin$ (ligne 4). Si l'algorithme revisite un appariement (ligne 19), la liste est allongée de $kstep$ (ligne 20) en prenant soin de ne pas dépasser la longueur maximal $kmax$. Si au contraire la longueur de la liste n'a pas été modifiée depuis au moins $kfreq$ mouvements (ligne 23), la liste est raccourcie de $kstep$ (ligne 24) en prenant soin de ne pas la réduire en dessous de la longueur minimale $kmin$.

5.3 Résultats expérimentaux

Afin de mesurer leur performance, nous testons nos algorithmes sur quatre types de problèmes :

- des problèmes d'isomorphisme de sous-graphe (partiel et induit) ;
- des problèmes de recherche du plus grand sous-graphe induit commun à deux graphes étiquetés ;
- des problèmes d'appariement non bijectifs de graphes issus de [BRB04] ;
- des problèmes d'appariement multivoques de graphes.

Dans tous les cas, le problème initial a été transformé en un problème de calcul de distance entre deux graphes (de façon similaire à ce qui est proposé au chapitre 2 de la partie 1) ce qui nous a permis d'utiliser nos algorithmes génériques du calcul de la distance entre deux graphes pour résoudre les problèmes initiaux.

5.3.1 Isomorphisme de sous-graphe

Benchmark

Foggia *et al.* [FSV01] proposent un benchmark de problèmes d'isomorphisme de sous-graphe induit. Le benchmark de Foggia *et al.* contient des graphes générés aléatoirement de différentes tailles et de différentes connexités. En outre, différentes tailles relatives du petit graphe par rapport au grand graphe sont proposées. Toutes les classes d'instances sont donc nommées par une

Algorithme Tabou réactif
Entrée : 2 graphes $G = (V, E)$ et $G' = (V', E')$
une distance δ telle que définie au chapitre 3
une longueur de liste minimum $kmin$, maximale $kmax$,
une longueur $kstep$ de raccourcissement ou de rallongement de la liste
une fréquence $kfreq$ de raccourcissement de la liste
et un nombre $nbMaxMouv$ de mouvements
Sortie : un appariement $m \subseteq V \times V'$

1 $mCourant \leftarrow Glouton(G, G', \delta)$
2 $meilleurM \leftarrow mCourant$
3 $nbMouv \leftarrow 0$
4 $k \leftarrow kmin$, $dateDerniereMAJ \leftarrow nbMouv$
5 **tant que** $nbMouv < nbMaxMouv$ **faire**
6 $nbMouv \leftarrow nbMouv + 1$
7 $aspiration \leftarrow \{m' | m' \in voisins(mCourant) \wedge \delta_{m'}(G, G') < \delta_{meilleurM}(G, G')\}$
8 **si** $aspiration \neq \emptyset$ **alors**
9 $cand \leftarrow aspiration$
10 **sinon**
11 $cand \leftarrow \{m' | m' \in voisins(mCourant) \wedge nonTabou(k, diff(m, m'))$
$\wedge \delta_{m'}(G, G')$ est minimal$\}$
12 **fin si**
13 $ms \leftarrow$ sélection aléatoire d'un appariement de $cand$
14 rendre tabou le mouvement $diff(mCourant, ms)$
15 $mCourant \leftarrow ms$
16 **si** $\delta_{mCourant}(G, G') < \delta_{meilleurM}(G, G')$
17 $meilleurM \leftarrow mCourant$
18 **fin si**
19 **si** $mCourant$ a déjà été visité **faire**
20 $k \leftarrow min(kmax, k + kstep)$
21 $dateDerniereMAJ \leftarrow nbMouv$
22 **fin si**
23 **si** $nbMouv \geq dateDerniereMAJ + kfreq$ **faire**
24 $k \leftarrow max(kmin, k - kstep)$
25 $dateDerniereMAJ \leftarrow nbMouv$
26 **fin si**
27 **fin tant que**
28 **retourner** $meilleurM$

FIG. 5.3 – Recherche taboue réactive d'un appariement

chaîne de caractère "si.o.k.c" où "o" est la taille du grand graphe (c.-à-d. son nombre de sommets), "k" la taille du petit graphe, et "c" la connectivité moyenne des graphes (le pourcentage d'arcs par rapport au nombre d'arcs maximum $taille * (taille - 1)$). Par exemple, la classe "si.100.20.5" contient des problèmes d'isomorphisme de sous-graphe pour lesquels le grand graphe contient 100 sommets, le petit 20 sommets et où chaque graphe a en moyenne 5% du nombre d'arcs maximum.

Le benchmark de Foggia *et al.* [FSV01] propose uniquement des problèmes d'isomorphisme de sous-graphe induit pour lesquels il existe une solution (il existe toujours au moins une relation d'isomorphisme de sous-graphe entre les paires de graphes). Comme chaque relation d'isomorphisme de sous-graphe induit correspond également à une relation d'isomorphisme de sous-graphe partiel, nous avons cherché, pour chaque paire de graphes considérée, les deux types de relations possibles.

Nous avons testé nos algorithmes sur les classes de problèmes où la taille du grand graphe est fixée à 100 sommets. La taille du petit graphe est de 20, 40 ou 60 sommets. Finalement, la connectivité des graphes est de 1% ou 5% (des arcs sont ajoutés pour rendre les graphes connexes si ça n'est pas le cas).

Définition de la mesure

Pour chacune de ces expérimentations, le problème original a été transformé en un calcul de la distance entre deux graphes de façon similaire à ce qui a été présenté en partie 1 de cette thèse. Néanmoins, les formulations proposées pour les problèmes d'isomorphisme de sous-graphe ne permettent pas à nos algorithmes de mesurer la qualité d'un appariement "incomplet" : l'appariement induit une distance nulle s'il définit une fonction d'isomorphisme de sous-graphe et $+\infty$ dans tous les autres cas. Cette formulation ne permet donc pas à l'algorithme Glouton ou à Tabou de choisir un "meilleur" couple de sommets. Par conséquent, nous avons repris les définitions des distances δ^{psub} et δ^{sub} afin que la distance induite par un appariement soit proportionnelle au nombre de sommets et d'arcs restant à apparier.

De façon formelle, la distance $\delta^{psub} =< \delta_v^{psub}, \delta_e^{psub}, \otimes_{\sum} >$ permettant la résolution de l'isomorphisme de sous-graphe partiel est définie par :

$$
G \begin{cases} \forall v \in V, \forall s_v \subseteq V', \delta_v^{psub}(v, s_v) = \begin{array}{l} 1 \text{ si } s_v = \emptyset \\ 0 \text{ si } |s_v| = 1 \\ +\infty \text{ sinon} \end{array} \\ \forall (u,v) \in E, \forall s_e \subseteq E', \delta_e^{psub}(u, v, s_e) = \begin{array}{l} 1 \text{ si } s_e = \emptyset \\ 0 \text{ si } |s_e| = 1 \\ +\infty \text{ sinon} \end{array} \end{cases}
$$

$$
G' \begin{cases} \forall v \in V', \forall s_v \subseteq V, \delta_v^{psub}(v, s_v) = \begin{array}{l} 0 \text{ si } |s_v| \leq 1 \\ +\infty \text{ sinon} \end{array} \\ \forall (u,v) \in E', \forall s_e \subseteq E, \delta_e^{psub}(u, v, s_e) = \begin{array}{l} 0 \text{ si } |s_e| \leq 1 \\ +\infty \text{ sinon} \end{array} \end{cases}
$$

La distance $\delta_G^{sub} =< \delta_v^{sub}, \delta_{eG}^{sub}, \otimes_{\sum} >$ permettant la résolution de l'isomorphisme de sous-graphe induit est définie par :

$$
G'' \begin{cases}
1 & \forall v \in V, \forall s_v \subseteq V', \delta_v^{sub}(v, s_v) = \begin{array}{l} 1 \text{ si } s_v = \emptyset \\ 0 \text{ si } |s_v| = 1 \\ +\infty \text{ sinon} \end{array} \\
2.1 & \forall (u,v) \in V^2, \forall s_e \subseteq E', \delta_{eG}^{sub}(u, v, s_e) = \begin{array}{l} 1 \text{ si } (u,v) \in E \wedge s_e = \emptyset \\ 0 \text{ si } (u,v) \in E \wedge |s_e| = 1 \end{array} \\
2.2 & = \begin{array}{l} 0 \text{ si } (u,v) \notin E \wedge s_e = \emptyset \\ +\infty \text{ sinon} \end{array}
\end{cases}
$$

$$
G' \begin{cases}
3 & \forall v \in V', \forall s_v \subseteq V, \delta_v^{sub}(v, s_v) = \begin{array}{l} 0 \text{ si } |s_v| \leq 1 \\ +\infty \text{ sinon} \end{array} \\
4 & \forall (u,v) \in E', \forall s_e \subseteq E, \delta_{eG}^{sub}(u, v, s_e) = \begin{array}{l} 0 \text{ si } |s_e| \leq 1 \\ +\infty \text{ sinon} \end{array}
\end{cases}
$$

Notons que cette nouvelle formulation ne change pas la validité des théorèmes correspondant (ni les preuves de ces théorèmes). Simplement, cette définition permet à la distance de compter le nombre de sommets et d'arcs devant encore être retrouvés.

Voisinage et filtrage

Les solutions d'un problème d'isomorphisme de sous-graphe sont nécessairement des appariements univoques. Par conséquent, afin d'améliorer le temps d'exécution de nos algorithmes, ceux-ci sont utilisés avec une définition du voisinage $voisins_{uni}$ ne considérant que des appariements univoques : à chaque itération de nos algorithmes ne sont envisagés que les couples de sommets dont l'ajout ou la suppression permet d'obtenir un appariement univoque.

Le changement de voisinage ne change absolument pas le comportement de nos algorithmes : à chaque itération, le même couple de sommets est sélectionné avec un voisinage univoque et un voisinage multivoque. Par conséquent, le taux de réussite de nos algorithmes est le même avec un voisinage univoque ou multivoque. Néanmoins, en utilisant un voisinage univoque, le nombre d'appariements à examiner à chaque itération est fortement réduit et le nombre de mouvements réalisés par seconde est considérablement augmenté : Tabou est 13 fois plus rapide sur la classe de problèmes si100.60.1 et 18 fois plus rapide sur la classe de problèmes si100.20.1.

Quelques couples de sommets ont été filtrés en fonction du degré des sommets (c.-à-d. leur nombre d'arcs entrants et sortants). Pour le problème de l'isomorphisme de sous-graphe entre un graphe $G = (V, E)$ et un graphe $G' = (V', E')$, les appariements contenant des couples de sommets $(u, v) \in V \times V'$ tels que le nombre d'arcs sortants (resp. entrants) de u est strictement supérieur au nombre d'arcs sortants (resp. entrants) de v ont été supprimés de l'espace de recherche.

Filtrer ces couples de sommets n'est pas nécessaire à la résolution du problème de l'isomorphisme de graphe mais permet de réduire l'espace de recherche. Notre algorithme est plus performant en temps d'exécution comme en taux de réussite.

Paramétrage

Pour chaque classe de problèmes, les 50 premières instances (sur les 100 existantes) ont été étudiées. Chaque algorithme a été paramétré de façon à faire au plus 100 000 mouvements (c.-à-d. 100 000 ajouts ou retraits de couples de sommets). Au delà de 100 000 mouvements, nos résultats montrent que le gain de performance des algorithmes tabous sont faibles et que par conséquent, il est plus intéressant de relancer complètement l'algorithme à partir d'un autre point de l'espace de recherche.

Glouton a été itéré autant de fois que nécessaire pour trouver la solution du problème (c.-à-d. un appariement induisant une distance nulle) ou jusqu'à la réalisation de 100 000 ajouts de couples de sommets. Les algorithmes Tabous démarrent par le meilleur appariement trouvé lors de 10 exécutions de l'algorithme Glouton (le nombre de mouvements réalisés par ces exécutions de Glouton est retranché des 100 000 mouvements) puis poursuivent leur exploration de l'espace de recherche jusqu'à l'épuisement des mouvements autorisés restants.

Afin de mesurer l'influence de la longueur de la liste taboue, pour chaque problème, Tabou a été exécuté avec toutes les longueurs comprises entre 5 et 31. Ces bornes minimum et maximum ont été déterminées expérimentalement : en dehors de ces bornes, l'algorithme tabou donne de très mauvais résultats sur toutes les classes de problèmes.

Finalement, Tabou réactif a été paramétré en tenant compte des résultats obtenus par Tabou. La longueur minimale *min* de la liste taboue a été fixée à 15 (la plus petite longueur optimale pour Tabou toutes classes d'instances confondues), la longueur maximale *max* a été fixée soit à 30 soit à 35, la fréquence de réaction $freq$ soit à 500 soit à 1000, la longueur de raccourcissement et de rallongement *step* de la liste à 5, 10, 15 ou 20. Par conséquent, au total, 16 paramétrages différents ont été testés pour notre algorithme tabou réactif.

Chaque algorithme a été exécuté 10 fois sur chaque instance. Notons que, afin de mesurer le plus rigoureusement possible l'influence de la longueur de la liste taboue, les mêmes graines d'initialisation de notre générateur de nombres aléatoires ont été utilisées pour chaque tabou. Par conséquent, quel que soit le paramétrage de tabou ou de tabou réactif, les 10 exécutions de ces algorithmes ont démarré avec les 10 mêmes appariements initiaux.

Résultats

Classe	Glouton			Tabou							
				Meilleur tabou			Tabou k=16		Tabou k=18		t
	%r	%r10	t	%r	%r10	k	%r	%r10	%r	%r10	
si100.20.1.induit	**99,6**	100	*0,20*	89,6	100	15	89,6	100	86,4	100	0,32
si100.20.1.partiel	**99,6**	100	*0,11*	80,0	100	19	74,6	98	79,6	100	0,33
si100.40.1.induit	56,2	100	1,64	**99,2**	100	16	**99,2**	100	**98,8**	100	*0,24*
si100.40.1.partiel	63,8	98	1,40	**89,4**	**100**	22	**84,2**	**100**	**86,6**	**100**	*0,40*
si100.60.1.induit	49,0	88	2,15	100	100	15-16	100	100	100	100	*0,28*
si100.60.1.partiel	34,4	84	2,27	**93,0**	**100**	23	**89,8**	**100**	**89,6**	**100**	*0,47*
si100.20.5.induit	10,2	30	6,23	**45,8**	**98**	15	**44,6**	**88**	**33,2**	**74**	*1,49*
si100.20.5.partiel	13,8	40	6,98	**30,8**	**80**	16-17	**24,8**	**80**	**30,8**	**74**	*1,67*
si100.40.5.induit	55,2	96	8,00	**83,6**	**100**	20	**71,8**	**100**	**75,4**	**100**	*1,76*
si100.40.5.partiel	19,0	60	10,65	**26,8**	**96**	24	16,8	**84**	17,4	**80**	*2,27*
si100.60.5.induit	99,8	100	0,65	**100**	100	13-15	99,8	100	99,8	100	*0,47*
si100.60.5.partiel	81,6	100	5,07	**82,4**	100	22	75,8	100	79,4	100	*1,72*
Moyenne	56,8	83	3,78	**76,7**	**97,8**	19,6	**72,6**	**95,8**	**73,1**	**94**	*0,95*

TAB. 5.1 – Résultats de Glouton, du meilleur Tabou, de Tabou avec une longueur de liste k=16 (le meilleur en moyenne sur 10 exécutions) et de Tabou avec k=18 (le meilleur en moyenne sur 1 exécution) sur des problèmes d'isomorphisme de sous-graphe induit et partiel. %r est le pourcentage de réussite par exécution de l'algorithme considéré. %r10 est le pourcentage d'instances résolues au moins une fois sur les 10 exécutions. t est le temps moyen en secondes d'une exécution de l'algorithme (pour tabou, seule la moyenne pour tous les tabous est donnée car leurs temps d'exécution sont très similaires). k est la longueur de la liste ayant permis à Tabou d'obtenir les meilleurs résultats pour cette classe.

Le tableau 5.1 compare les performances de Glouton et de Tabou sur des problèmes d'isomorphisme de sous-graphe partiel et induit. Tous ces tests ont été réalisés sur un AMD Athlon à 2,4GHz sous Ubuntu. Le temps nécessaire à l'exécution de 100 000 mouvements varie en fonction de l'algorithme (Glouton est plus lent que Tabou) et de la taille des instances : il est compris entre 2 et 15 secondes pour Glouton et entre 1 et 5 secondes pour Tabou (ou Tabou réactif).

Comparaison avec une approche dédiée. Les temps d'exécutions de nos algorithmes par rapport à leur taux de réussite montrent qu'ils ne sont pas compétitifs avec une approche complète dédiée à ce problème. En effet, VFLib [CFCS01] résout la majorité de ces problèmes en un temps inférieur à 1 seconde alors qu'il faut en général plusieurs secondes à notre algorithme le plus rapide pour atteindre un taux de réussite proche de 100%. Ceci peut s'expliquer par le fait que l'isomorphisme de sous-graphe est un problème de satisfaction pour lequel il existe de nombreuses façons de filtrer efficacement l'espace de recherche et que nos algorithmes transforment le problème en un problème d'optimisation et n'utilisent pas de technique de filtrage pendant la recherche.

Notons cependant que quelques instances semblent cependant très difficiles pour VFlib. Par exemple, plus de 1200 secondes lui sont nécessaires pour résoudre une instance particulière de la classe si100.20.5.partiel et certaines autres instances nécessitent plus d'une minute de calcul alors que nos algorithmes trouvent généralement la solution de ces instances en quelques secondes seulement.

Comparaison Tabou/Glouton. De façon générale, Tabou donne de meilleurs résultats que Glouton : il permet de résoudre 76,7% des instances contre 56,8% pour Glouton (moyenne sur toutes les classes). Sur 10 classes de problèmes (sur les 12 considérées), Tabou a un meilleur comportement que Glouton. En outre, une exécution de Tabou est en moyenne 4 fois plus rapide qu'une exécution de Glouton. Ceci s'explique par le fait que le nombre de couples de sommets que Tabou doit examiner à chaque mouvement est généralement plus petit que pour Glouton. L'appariement courant de Tabou contient généralement un grand nombre de couples de sommets et la contrainte d'univalence de l'appariement recherché réduit le voisinage de l'appariement courant à quelques appariements. *A contrario*, en moyenne, l'appariement courant de Glouton est généralement plus petit et le nombre de couples de sommets à examiner pour construire l'ensemble des couples de sommets candidats est en moyenne plus élevé que pour Tabou.

Si Tabou donne de meilleurs résultats en moyenne, les performances de Tabou sont moins bonnes que Glouton sur les instances de la classe si100.20.1.

Longueur de la liste taboue. La longueur optimale de la liste taboue varie énormément d'une classe de problèmes à une autre. Par exemple, sur la classe des problèmes d'isomorphisme de sous-graphe induit de type "si100.20.1" la longueur optimale est de 15 alors que sur la classe "si100.40.5", elle est de 24.

La figure 5.4 montre le taux de réussite en fonction de la longueur de la liste taboue en moyenne pour toutes les instances ainsi que pour les problèmes des classes si100.20.1.induit et si100.40.5.partiel. Cette courbe montre à quel point la longueur optimale de la liste taboue varie d'une instance à l'autre et qu'un mauvais paramétrage de cette longueur dégrade fortement les résultats de certaines classes de problèmes.

La tableau 5.1 montre qu'il n'y a pas de corrélation entre la taille du problème (c.-à-d. le nombre de couples de sommets à considérer) et la longueur optimale de la liste taboue. En

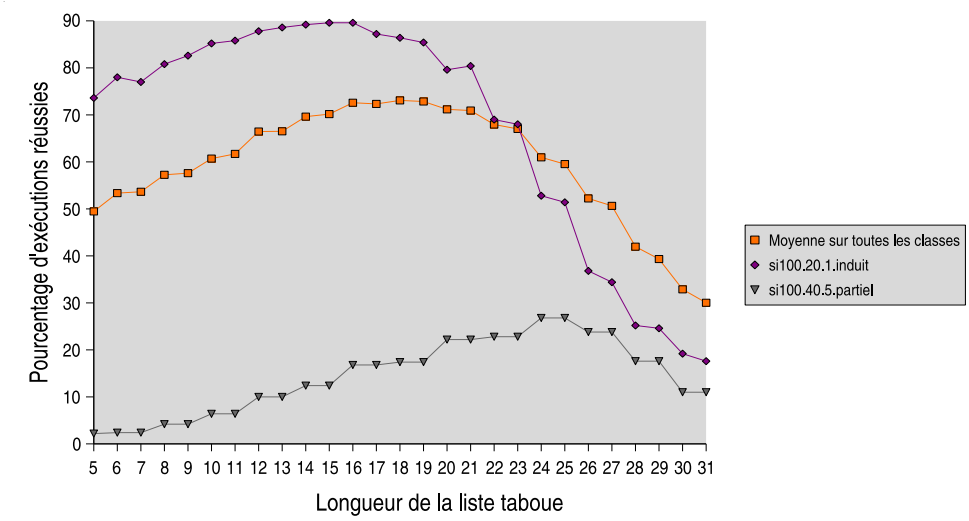


FIG. 5.4 – Évolution du taux de réussite en fonction de la longueur de la liste taboue en moyenne pour toutes les instances et pour les problèmes si100.20.1.induit et si100.40.5.partiel.

moyenne, les problèmes d'isomorphisme de sous-graphe induit nécessitent une liste taboue plus longue que les problèmes d'isomorphisme de sous-graphe partiel. Finalement, la connectivité des graphes ne semble pas influer sur la valeur optimale de la longueur de la liste taboue. Notons cependant que pour Tabou comme pour glouton, les instances avec des graphes ayant une connectivité de 5% semblent plus difficiles que les problèmes avec des graphes ayant une connectivité de 1%.

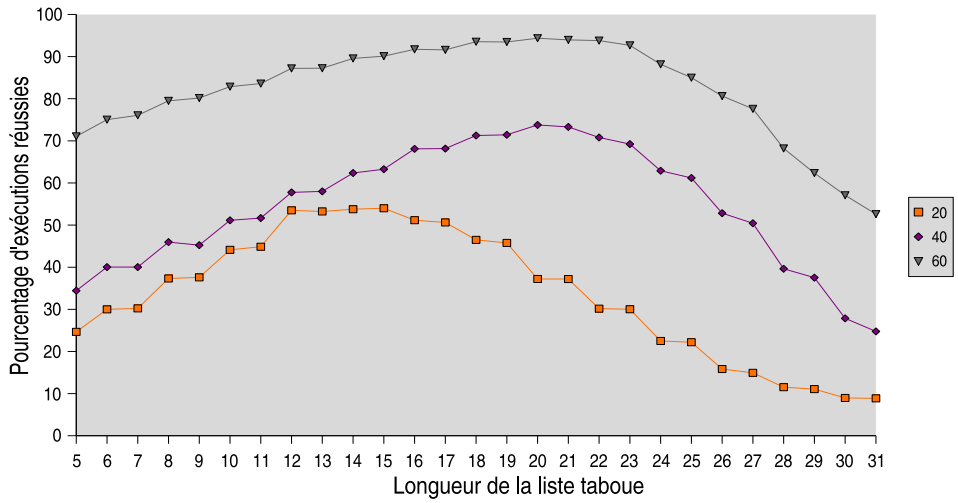


FIG. 5.5 – Évolution du taux de réussite en fonction de la longueur de la liste taboue par rapport à la taille du petit graphe (taille du gros graphe = 100 sommets).

La figure 5.5 montre l'évolution du taux de réussite en fonction de la longueur de la liste taboue selon si le petit graphe contient 20, 40 ou 60 sommets. Cette figure montre que la longueur

optimale de la liste varie en fonction de la proportion de sommets du petit graphe. Néanmoins, alors qu'il y a une différence entre les graphes de 20 et de 40 sommets, il ne semble pas y en avoir entre les graphes de 40 et de 60 sommets.

Nos résultats montrent que la longueur optimale de la liste taboue semble dépendre de nombreux paramètres et qu'il n'est pas possible de trouver a priori la longueur optimale pour une classe de problèmes donnée. En outre, un mauvais paramétrage de tabou peut donner de très mauvais résultats. Par conséquent, l'algorithme tabou n'est pas robuste à un mauvais paramétrage.

Classe	Tabou 15-24				RTS			
	Min (k)	Moy	Max (k)	Tabou k=18	Min	Moy	Max	15-35-20-1000
si100.20.1.induit	52,8 (24)	78,8	89,6 (15)	86,4	**93,4**	**94,7**	**95,8**	**94,8**
si100.20.1.partiel	63,2 (24)	74,2	80,0 (19)	79,6	**89,0**	**91,0**	**92,2**	**91,6**
si100.40.1.induit	86,2 (24)	96,2	99,2 (17)	98,8	**98,8**	**99,2**	**99,6**	**99,6**
si100.40.1.partiel	82,2 (15)	86,4	89,4 (22)	86,6	**90,0**	**91,9**	**95,6**	**95,6**
si100.60.1.induit	95,6 (24)	99,4	100 (15)	100	**100**	**100**	100	100
si100.60.1.partiel	89,4 (15)	90,6	93,0 (23)	89,6	**92,0**	**94,4**	**97,4**	**97,4**
si100.20.5.induit	0,6 (24)	23,2	**45,8** (16)	**33,2**	**31,2**	**35,8**	41,0	33,0
si100.20.5.partiel	6,2 (24)	21,3	**30,8**(19)	**30,8**	**26,0**	**27,1**	29,0	26,4
si100.40.5.induit	56,6 (24)	**72,6**	**83,6** (20)	**75,4**	**59,8**	64,4	67,4	67,4
si100.40.5.partiel	12,4 (15)	19,8	**26,8** (24)	17,4	**17,8**	**20,9**	24,0	**24,0**
si100.60.5.induit	96,2 (23)	99,0	100 (15)	99,8	**99,6**	**99,9**	**100**	**100**
si100.60.5.partiel	68,8 (24)	77,5	82,4 (22)	79,4	**77,6**	**84,3**	**88,2**	**88,2**
Moyenne	59,2	69,9	76,7	73,1	**72,9**	**75,3**	**77,5**	**76,5**

TAB. 5.2 – Résultats de Tabou et de Tabou réactif (RTS) avec différents paramétrages : tabou avec des longueurs de liste allant de 15 à 24 et RTS avec tous les paramétrages considérés. Pour chaque classe de problèmes et chaque algorithme sont présentés le plus petit (resp. grand) taux de réussite *min* (resp. *max*), la moyenne *moy* des taux de réussite et le taux de réussite du meilleur paramétrage toutes classes de problèmes confondues (Tabou avec k=18 et RTS15-35-20-1000). Pour tabou, dans les colonnes Min et Max est précisées entre parenthèses la longueur de la liste k utilisée.

Tabou réactif. Le tableau 5.2 compare les résultats obtenus par tabou et par tabou réactif. Afin de comparer leur robustesse, nous comparons simultanément les résultats obtenus avec plusieurs paramétrages différents :

- Les résultats donnés pour tabou sont ceux obtenus avec différentes exécutions de tabou avec différentes longueurs de liste ($15 \leq k \leq 24$). Ces bornes correspondent aux bornes minimales et maximales des longueurs optimales par classe de problèmes (voir tableau 5.1) ;
- Les résultats donnés pour tabou réactif sont tous ceux qui ont été testés (c.-à-d. les 16 paramétrages différents).

Pour chacune des classes d'algorithmes, sont donnés le taux de réussite minimum Min (obtenu avec le plus mauvais paramétrage), le taux de réussite moyen Moy, le taux de réussite maximum Max (obtenu avec le meilleur paramétrage) et le taux de réussite obtenu avec le meilleur paramétrage moyen toutes classes de problèmes confondues (c.-à-d. une longueur de 18 pour tabou, une longueur minimale (resp. maximale) de 15 (resp. 35) pour RTS, une variation de liste de 20 et une fréquence $freq$ de rafraîchissement à 1000). Les temps d'exécutions de Tabou et de RTS sont quasiment identiques, par conséquent, ils ne sont pas répétés dans ce tableau.

Robustesse au paramétrage. Pour chacune des classes de problèmes, la différence entre le taux de réussite minimum de RTS et son taux de réussite maximum est faible par rapport à celui de tabou. En moyenne, cette différence est de 4,6 points pour RTS contre 17,5 pour tabou. Par conséquent, RTS est beaucoup plus robuste que Tabou au paramétrage.

Taux de réussite minimaux. Les taux de réussite minimaux de Tabou réactif sont tous supérieurs à ceux de Tabou : l'écart s'échelonne entre 3,2 points pour la classe si100.40.5.induit et 40,6 points pour la classe si100.20.1.induit. Par conséquent, un mauvais paramétrage de RTS donnera systématiquement un meilleur résultat qu'un mauvais paramétrage de Tabou.

Taux de réussite moyens. Les taux de réussite en moyenne de Tabou réactif sont supérieurs à ceux de Tabou pour 11 classes de problèmes sur 12. Par conséquent, RTS donne globalement de bien meilleurs résultats que Tabou.

Taux de réussite maximaux. Si on compare les taux de réussite maximaux par classe de problèmes, RTS est meilleur que tabou pour 6 classes contre 4 classes pour tabou. Toutes instances confondues, tabou obtient un taux de réussite maximum moyen de 76,7% contre 77,5% pour RTS. RTS est donc encore une fois meilleur pour ce critère. Notons cependant que pour la classe si100.40.5.induit, les résultats de RTS sont particulièrement décevants, la perte d'efficacité par rapport à tabou étant particulièrement importante (16,2 points en moins).

Algorithme moyen. Le meilleur tabou réactif moyen donne de meilleurs résultats que le meilleur tabou moyen sur 8 classes de problèmes (contre 3 pour le meilleur tabou). En moyenne, son taux de réussite est de 77,5% contre 73,1% pour tabou.

Notons que sur certaines classes d'instances, tabou réactif donne de bien meilleurs résultats que tabou : pour les problèmes si100.20.1 (induits et partiels) et pour si100.40.1.partiel, le taux de réussite minimum de tabou réactif sont supérieurs au taux de réussite maximum de tabou. En outre, pour 10 classes d'instances sur 12, les taux de réussite minimaux de RTS sont supérieurs aux taux de réussites moyens de tabou.

Les résultats montrent que tabou réactif est beaucoup plus robuste au paramétrage que tabou. En outre, en moyenne, tabou réactif donne de meilleurs résultats que tabou.

5.3.2 Plus grand sous-graphe commun.

Benchmark

Le benchmark de Foggia *et al.* [FSV01] propose également des problèmes de recherche du plus grand sous-graphe induit commun à deux graphes[10]. Les sommets et les arcs de ces graphes sont étiquetés. Par conséquent, les couples de sommets appariés (et les arcs) doivent avoir la même étiquette. Foggia *et al.* [FSV01] soulignent le fait que la difficulté de la recherche du plus grand sous-graphe commun dépend directement du nombre d'étiquettes différentes sur les sommets et les arcs : plus ce nombre est grand, plus le nombre de couples de sommets pouvant appartenir au meilleur appariement est faible et plus le problème est simple. Par conséquent, Foggia *et al.* codent les étiquettes sur 16 bits et un décalage de bits sur ces étiquettes permet de faire varier le nombre total d'étiquettes de sommets et d'arcs.

Nous avons testé nos algorithmes sur des instances où les graphes ont 100 sommets et une connectivité de 1% ou de 5%. Le nombre de décalages de bits sur les étiquettes a été compris

[10] La taille $|G|$ d'un graphe $G = (V, E)$ est ici définie comme son nombre de sommets, c.-à-d. que $|G| = |V|$.

entre 10 et 16 bits : les graphes ont donc entre une seule et 64 étiquettes de sommets et d'arcs différentes. Le benchmark garantit qu'entre chaque paire de graphes, la taille du plus grand sous-graphe commun est d'au moins 50 sommets (cette taille pouvant augmenter quand le nombre d'étiquettes diminue). Chaque classe de problèmes est identifiée par un nom de type "100.c.l" où "c" est la connectivité (1% ou 5%) et "l" le nombre maximum d'étiquettes.

Définition de la mesure

Chaque instance a été transformée en un problème de calcul de la distance entre deux graphes de façon similaire à ce qui a été présenté en partie 1 de cette thèse. Néanmoins, la modélisation proposée s'applique à des graphes non-étiquetés et il faut donc l'adapter afin qu'elle n'apparie que des sommets et des arcs ayant la même étiquette.

De façon formelle, étant donnés deux graphes étiquetés $G = (V, E, L_V, \alpha, L_E, \beta)$ et $G' = (V', E', L_V, \alpha', L_E, \beta')$, le problème consiste à calculer la distance $\delta^{mcs} =< \delta_v^{mcs}, \delta_e^{mcs}, \otimes_{\sum} >$ entre les deux graphes $G_2 = (V, V \times V)$ et $G'_2 = (V', V' \times V')$ telle que :

$$G_2 \begin{cases} \forall v \in V, \forall s_v \subseteq V', \delta_v^{mcs}(v, s_v) = \begin{array}{l} 0 \text{ si } s_v = \{v'\} \wedge \alpha(v) = \alpha(v') \\ 1 \text{ si } s_v = \emptyset \\ +\infty \text{ sinon} \end{array} \\ \forall (u,v) \in V^2, \forall s_e \subseteq V'^2, \delta_{eGG'}^{mcs}(u, v, s_e) = \begin{array}{l} 0 \text{ si } s_e = \emptyset \\ 0 \text{ si } s_e = \{(u', v')\} \wedge (u, v) \in E \\ \wedge (u', v') \in E' \wedge \beta((u, v)) = \beta'((u', v')) \\ 0 \text{ si } s_e = \{(u', v')\} \wedge (u, v) \notin E \wedge (u', v') \notin E' \\ +\infty \text{ sinon} \end{array} \end{cases}$$

$$G'_2 \begin{cases} \forall v \in V', \forall s_v \subseteq V, \delta_v^{mcs}(v, s_v) = \begin{array}{l} 0 \text{ si } |s_v| \leq 1 \\ +\infty \text{ sinon} \end{array} \\ \forall (u,v) \in V'^2, \forall s_e \subseteq V^2, \delta_{eGG'}^{mcs}(u, v, s_e) = \begin{array}{l} 0 \text{ si } s_e = \emptyset \\ 0 \text{ si } s_e = \{(u', v')\} \wedge \\ ((u, v) \in E' \Leftrightarrow (u', v') \in E) \\ +\infty \text{ sinon} \end{array} \end{cases}$$

Voisinage et filtrage

Les solutions d'un problème de recherche d'un plus grand sous-graphe commun sont nécessairement des appariements univoques. Par conséquent, afin d'améliorer le temps d'exécution de nos algorithmes, ceux-ci sont utilisés avec une définition du voisinage $voisins_{uni}$ ne considérant que des appariements univoques : à chaque itération de nos algorithmes ne sont envisagés que les couples de sommets dont l'ajout ou la suppression permet d'obtenir un appariement univoque.

Finalement, comme aucun appariement solution ne peut contenir des couples de sommets n'ayant pas la même étiquette, tous ces couples de sommets ont été filtrés.

Paramétrage

Pour chaque classe de problèmes, 50 instances ont été étudiées. Comme pour le problème de l'isomorphisme de sous-graphe, la recherche a été interrompue au bout de 100 000 mouvements. Notons que comme la recherche du plus grand sous-graphe commun est un problème d'optimisation et que nos algorithmes ne peuvent montrer l'optimalité d'une solution trouvée, chaque

exécution a fait exactement 100 000 mouvements. Les temps donnés sont ceux nécessaires pour trouver la meilleure solution trouvée pendant une exécution.

Chaque algorithme a été exécuté 10 fois sur chacune des instances dans les mêmes conditions que pour les problèmes d'isomorphisme de sous-graphe.

L'algorithme tabou a été exécuté avec des longueurs de liste comprises entre 0 et 5.

Résultats

Classe	Glouton						Tabou0					
	r.1	r.10	50.1	50.10	Moy	t	r.1	r.10	50.1	50.10	Moy	t
100-1-64	0,0	0	0,6	6	44,53	0,91	100	100	100	100	52,60	0,21
100-1-32	0,0	0	0,6	6	46,11	1,32	81,4	100	100	100	55,17	1,06
100-1-16	0,0	0	5,6	16	46,99	2,42	44,0	100	100	100	56,70	2,35
100-1-8	0,0	0	32,0	68	48,90	3,05	30,8	98	100	100	56,92	3,30
100-1-4	0,0	0	88,4	100	50,96	5,67	26,6	100	100	100	59,22	5,64
100-1-2	0,0	0	100	100	53,56	10,95	24,0	100	100	100	63,80	9,27
100-1-1	0,0	0	100	100	58,25	22,41	18,8	100	100	100	69,11	13,02
100-5-64	1,6	14	1,6	14	38,45	0,10	100	100	100	100	50,00	14,83
100-5-32	0,0	0	0,0	0	29,18	1,07	100	100	100	100	50,00	0,24
100-5-16	0,0	0	0,0	0	25,50	1,98	100	100	100	100	50,00	0,44
100-5-8	0,0	0	0,0	0	25,50	2,51	98,6	100	98,6	100	49,75	2,92
100-5-4	0,0	0	0,0	0	26,68	4,85	36,2	94	27,6	64	37,92	15,93
100-5-2	0,0	0	0,0	0	28,47	9,96	25,4	100	0,0	0	35,07	24,13
100-5-1	0,0	0	0,0	0	31,31	20,91	26,2	100	0,0	0	38,27	41,22

TAB. 5.3 – Résultats de Glouton et de Tabou 0 sur des problèmes de recherche de plus grand sous-graphe commun à deux graphes. Pour chacun des algorithmes est donné le pourcentage d'exécutions (resp. d'instances) r.1 (resp. r.10) pour lesquelles l'algorithme renvoie la meilleure solution trouvée ; le pourcentage d'exécution (resp. d'instances) 50.1 (resp. 50.10) pour lesquelles l'algorithme trouve une solution supérieure ou égale à la borne minimum des 50 sommets ; la taille moyenne Moy des solutions trouvées et le temps t en secondes.

Le tableau 5.3 montre les résultats obtenus par Tabou 0 et Glouton sur la recherche du plus grand sous-graphe commun.

La taille du plus grand sous-graphe commun pour chaque instance n'est pas connue, toutefois, les instances ont été construites afin de garantir que la taille de ce graphe est supérieure ou égale à 50 sommets. Par conséquent, nous considérons trois critères pour l'évaluation :

- le pourcentage d'exécutions où l'algorithme a trouvé un sous-graphe commun ayant au moins 50 sommets (50-1 donne les résultats pour une exécution, 50-10 donne les résultats sur les 10 exécutions) ;
- le pourcentage d'exécutions où l'algorithme a trouvé un sous-graphe commun ayant une taille égale au plus grand sous-graphe commun trouvé par l'ensemble de nos algorithmes (r-1 donne les résultats pour une exécution, r-10 donne les résultats sur les 10 exécutions) ;
- la moyenne des tailles des plus grands sous-graphes communs trouvés.

Comparaison avec une approche dédiée. À notre connaissance, aucun algorithme complet ne permet de rechercher le plus grand sous-graphe commun à deux graphes de cette taille : dans [BFG+02], Bunke *et al.* proposent une comparaison expérimentale de 5 algorithmes complets de résolution de ce problème et, malgré un fort étiquetage des sommets, montrent qu'il faut

parfois plusieurs centaines de secondes pour apparier des graphes de 30 sommets seulement. Notre algorithme tabou est donc très compétitif sur ce type de problèmes.

Difficulté des instances. Nos résultats montrent que la difficulté des instances varie fortement selon le nombre d'étiquettes de sommets et d'arcs : les problèmes sont d'autant plus difficiles que le nombre d'étiquettes est faible. D'autre part, les problèmes où les graphes ont une connexité moyenne de 5% sont plus difficiles que ceux ayant une connexité moyenne de 1%.

Comparaison Tabou/Glouton. L'algorithme Glouton ne permet pas de trouver de bons appariements : la moyenne des tailles des plus grands sous-graphes communs qu'il trouve sont généralement très éloignées de l'optimum (l'optimum étant toujours égal ou supérieur à 50). *A contrario*, Tabou donne de bien meilleurs résultats (à l'exception des 4 classes les plus dures) et surpasse systématiquement Glouton dans la qualité des appariements retournés. À l'exception des graphes ayant une connexité de 5% et un nombre d'étiquettes de sommets et d'arcs inférieur ou égal à 8, Tabou permet toujours de trouver des sous-graphes communs ayant une taille supérieure ou égale à la borne minimale de 50 sommets.

Longueur de la liste taboue. Nos expérimentations ont montré que, quel que soit le type de graphes considéré, la longueur optimale de la liste taboue est de 0 et que le pourcentage de réussite diminue quand la liste s'allonge. Avec une liste de longueur nulle, notre algorithme tabou choisit systématiquement un meilleur voisin de l'appariement courant et ne s'interdit aucun mouvement.

Le fait que cette heuristique donne de bons résultats peut paraître surprenant. En effet, selon le paysage de recherche, utiliser une telle heuristique ne permet pas de quitter les minima locaux. Néanmoins, cette heuristique fonctionne du fait des particularités du problème de la recherche du plus grand sous-graphe commun : tout appariement m induisant une distance (non infinie) égale à d ($d = |m|$) admet au moins $|m|$ voisins induisant une distance égale à $|m|+1$. Par conséquent, lorsque l'algorithme atteint un optimum local m, le prochain voisin est choisi aléatoirement parmi ces $|m|$ voisins. Généralement, parmi ces voisins, il en existe au moins un ayant un meilleur voisin autre que m. Par conséquent, l'algorithme ne stagne généralement pas sur un minimum local et il n'est pas nécessaire de diversifier la recherche avec une liste taboue. Par conséquent, nous ne présentons en détails que les résultats obtenus avec une longueur de 0. Notons cependant que, si le pourcentage %r des tabous avec une longueur supérieure à 0 sont mauvais par rapport à ceux obtenus par Tabou0, le pourcentage %50 des tabous est presque toujours égal à 100% et la moyenne des tailles des plus grand sous-graphes trouvés est proche de celle de Tabou0. Par conséquent, quel que soit le paramétrage de Tabou, les résultats obtenus sont bien meilleurs que ceux de Glouton.

Tabou réactif. Les résultats obtenus par Tabou montrent que même avec une longueur de liste au minimum, la recherche est suffisamment diversifiée dans le cas général. Par conséquent, introduire un processus de réactivité pour augmenter la diversification de la recherche ne semble pas utile. Néanmoins, pour une instance de la classe 100-1-8 et trois instances de la classe 100-5-4, Tabou1 trouve une meilleure solution que Tabou0. Nous avons donc testé Tabou réactif avec une liste taboue ayant une longueur minimale de 0 et une longueur maximale de 1 en espérant obtenir un algorithme aussi bon que Tabou0 et Tabou1. Ce test a montré que, comme il est très fréquent que le même appariement soit visité à quelques mouvements d'intervalle, la longueur de

la liste Taboue atteignait systématiquement son maximum et le comportement de cet algorithme tabou réactif était le même que celui de Tabou1.

5.3.3 Appariement non bijectif de graphes de Boeres *et al.*

Jeu d'essais.

Nous avons vu en partie 1 qu'il était possible de définir notre distance de deux graphes de façon à résoudre le problème de l'appariement non bijectif de graphe introduit par Boeres *et al.* dans [BRB04]. Nous avons donc transformé les 7 instances du problème de Boeres *et al.* en un problème de calcul de distance de graphes. Pour ces 7 instances, le graphe modèle contient entre 10 et 50 sommets et le graphe image entre 30 et 250 sommets. Sur ces instances, nous comparons notre algorithme Tabou réactif avec deux algorithmes :

- *LS+*, l'algorithme de construction aléatoire d'appariements proposé par Boeres *et al.* [BRB04] ;
- `ANT-GM'06`, l'algorithme d'optimisation par colonies de fourmis [SSSG06a, SSJ06a] pour le calcul de la mesure de similarité de graphes de [CS03] qui permet également le calcul de la similarité de Boeres *et al.* Cet algorithme est détaillé dans la section 4.3.6 du chapitre 4.

Pour ces instances et afin de nous comparer avec ces deux algorithmes, nous ne donnons que les résultats de tabou réactif avec un paramétrage optimal (ceux des différents tabous étant en moyenne tous moins bons). Les résultats de l'algorithme `ANT-GM'06` sont également ceux obtenus avec le meilleur paramétrage trouvé. Les résultats sont donnés en moyenne sur 20 exécutions. Les résultats de *LS+* sont ceux publiés par Boeres *et al.* dans [BRB04].

LS+

L'algorithme *LS+* proposé par Boeres *et al.* [BRB04] se déroule en deux temps : la génération d'un appariement respectant toutes les contraintes dures du problème par un algorithme glouton puis l'amélioration par recherche locale de cet appariement.

L'algorithme glouton construit des appariements en démarrant d'un appariement vide $m = \emptyset$ et en ajoutant des couples de sommets à m. À chaque itération de l'algorithme, un sommet v non apparié du graphe image est sélectionné. Un sommet v' du graphe modèle est alors choisi aléatoirement parmi l'ensemble des sommets du graphe modèle pouvant être apparié à v (c.-à-d. que (v, v') n'est pas un couple de sommets interdit et que l'ensemble des sommets appariés à v' sont connectés). S'il existe un tel sommet v', le couple de sommets (v, v') est ajouté à l'appariement m. Si au contraire il n'existe pas, la construction est interrompue et une nouvelle construction à partir de l'appariement vide est démarrée. S'il a été possible d'apparier tous les sommets du graphe image, l'appariement obtenu est valide (il respecte toutes les contraintes dures) et est mémorisé. La construction des appariements est itérée tant que le nombre d'essais maximum n'est pas atteint. Si plusieurs appariements valides ont été trouvés, le meilleur (au sens de la similarité de Boeres) est retourné.

Dans une deuxième étape, l'appariement m trouvé par glouton est amélioré par recherche locale. À chaque itération, un couple de sommets (v, v') de m est remplacé par un autre couple de sommets (v, v'') si cette modification permet d'augmenter la similarité induite par m tout en continuant à respecter les contraintes dures du problème. Lorsque plus aucun remplacement n'améliore l'appariement, le voisinage de la recherche locale est modifié : une paire de couples de sommets (v_1, v'_1) et (v_2, v'_2) peut alors être remplacé par la paire (v_1, v'_2) et (v_2, v'_1) si cette modification permet d'améliorer l'appariement. La recherche locale est interrompue quand plus aucune des deux types de modifications ne peut améliorer l'appariement.

Définition de la mesure

Pour notre algorithme tabou réactif, nous avons défini notre mesure de la distance de deux graphes conformément à ce qui est décrit en partie 1. `ANT-GM'06` est dédié au calcul de la mesure de similarité de graphes multi-étiquetés de Champin et Solnon. Cette mesure permet également le calcul de la similarité de Boeres [SS05a] : il faut pour cela étiqueter les sommets du graphe image et du graphe modèle de façon à ce qu'une étiquette $l_{(u,v)}$ de sommet ne soit retrouvé par un appariement m si et seulement si $(u, v) \in m$ (voir la section 2.3.3 pour la définition de cet étiquetage). La fonction f doit être définie comme une somme des similarités des sommets et des arcs appariés entre eux mais doit retourner $-\infty$ quand au moins un sommet n'est pas apparié ou qu'un couple de sommets interdit est utilisé. Finalement, la fonction g doit retourner $-\infty$ quand un sommet de l'image est éclaté ou qu'un sommet du modèle est éclaté vers un ensemble de sommets du graphe image non connecté et 0 sinon.

Paramétrage d'`ANT-GM'06`

Pour `ANT-GM'06`, le poids α du facteur phéromonal a été fixé à 2. Le taux d'évaporation de la phéromone ρ est fixé à 2%, le poids du facteur heuristique est fixé à 10, le nombre de cycles maximum est fixé à 2000. Le nombre de fourmis de la colonie est 20. Les bornes τ_{min} et τ_{max} de la quantité de phéromone sont fixées à $0,01$ et 6.

Afin d'évaluer l'intérêt d'hybrider l'optimisation par colonies de fourmis avec de la recherche locale, `ANT-GM'06` est testé avec et sans phase d'amélioration des appariements par recherche locale. Ces algorithmes sont respectivement notés `ANT-GM'06+LS` et `ANT-GM'06`.

Paramétrage de RTS

Tabou réactif a été testé sur ces instances avec de très nombreux paramétrages différents. Nous présentons seulement les résultats obtenus avec le meilleur paramétrage trouvé.

La longueur *min* de la liste taboue est fixée à 10, la longueur maximale *max* à 50, la liste est rallongée de 15 en cas de collision et réduite tous les 1000 mouvements sans redondance. La recherche fait 50000 mouvements. Notons que l'exécution de notre algorithme glouton sur ces instances est déterministe : les poids utilisés sont des nombres réels tous différents deux à deux et par conséquent, il n'y a jamais de couples de sommets ex-æquo à départager aléatoirement. Il est donc inutile de réitérer tabou réactif et une seule exécution de RTS est réalisée.

Résultats

Le tableau 5.4 présente les résultats sur les 7 instances du problème de l'appariement non-bijectif de graphes [BRB04] du benchmark 2. RTS et `ANT-GM'06` obtiennent de meilleurs résultats que l'algorithme de référence proposé par Boeres *et al.* [BRB04] (6 instances sur 7 sont mieux résolues par RTS et 7 instances sur 7 par les algorithmes ACO).

Sur ces instances, les algorithmes de fourmis donnent des résultats systématiquement meilleurs que Tabou réactif. Néanmoins, Tabou réactif est nettement plus rapide que les fourmis et permet d'obtenir tout de même de bons résultats (meilleurs que ceux de LS+). Les résultats obtenus par Tabou et `ANT-GM'06` sont donc complémentaires.

5.3.4 Recherche d'un meilleur appariement multivoque

Les problèmes d'appariement que nous avons résolus jusqu'alors sont basés sur des appariements univoques (problème de l'isomorphisme de sous-graphe ou recherche du plus grand

Problème			LS+	RTS		ANT-GM'06			ANT-GM'06+LS		
GM-i	($\mid V_1 \mid, \mid V_2 \mid$)	L	Max	Max	T.	Max	Moy	T.	Max	Moy	T.
5	(10, 30)	18	.5474	.5481	0.9	.5601	.5598	16	**.5608**	.5604	15
5a	(10, 30)	19	.5435	.5529	4.6	.5638	.5638	10	**.5645**	.5641	7
6	(12, 95)	269	.4248	.4213	0.0	**.4252**	.4251	211	**.4252**	.4251	215
7	(14, 28)	13	.6319	.6333	2.1	.6369	.6369	7	**.6376**	.6369	5
8	(30, 100)	595	.5186	.5210	1.3	.5229	.5226	462	**.5232**	.5228	229
8a	(30, 100)	595	.5222	.5245	1.3	.5263	.5261	456	**.5269**	.5264	241
9	(50, 250)	6018	.5187	.5199	81.7	.5201	.5201	4133	**.5203**	.5202	2034

TAB. 5.4 – Résultats sur les problèmes d'appariement non-bijectif de graphes de Boeres *et al.* [BRB04]. Pour chacune des instances est précisé le nom de l'instance, le nombre de sommets des deux graphes, la limite en temps CPU L de chacune des exécutions de `ANT-GM` (sur un Pentium IV à 1,7GHz). La similarité maximum Max obtenue par chacun des algorithmes est donnée. Il ne nous a pas été possible de connaître le temps d'exécution nécessaire à LS+. Le temps $T.$ moyen en secondes d'une exécution des algorithmes RTS, `ANT-GM'06` et `ANT-GM'06+LS` est donné. Une seule exécution de tabou a été réalisée (à cause de son comportement déterministe sur ces instances). Chacun des algorithmes de fourmis a été exécuté 20 fois sur chaque instance. La moyenne Moy de la similarité obtenue sur 20 exécutions est donnée.

sous-graphe commun) ou des appariements fonctionnels non bijectifs (problème de Boeres *et al.* [BRB04]). À notre connaissance, il n'existe pas de benchmark de problèmes basés sur des appariements multivoques (et non fonctionnels). Nous avons donc générés de tels problèmes.

Problème de recherche d'un meilleur appariement multivoque

Le problème considéré est directement inspiré de la mesure de similarité de Champin et Solnon [CS03]. Étant donnés deux graphes étiquetés, le problème consiste à trouver l'appariement qui maximise le nombre d'étiquettes de sommets et d'arcs retrouvées (une étiquette est retrouvée si le sommet ou l'arc auquel elle est attachée est apparié à un sommet ou un arc ayant la même étiquette) tout en minimisant les éclatements de sommets (c.-à-d. le nombre de fois ou un sommet est apparié à plus d'un sommet).

De façon plus formelle, en reprenant les notations utilisées par Champin et Solnon présentées en partie 1 de cette thèse, on définit la fonction f (appliquée à l'ensemble des étiquettes de sommets et d'arcs retrouvées par l'appariement) comme la fonction cardinalité et la fonction g (appliquée à l'ensemble des couples (v, s_v) tels que v est un sommet apparié à chacun des sommets de s_v) par :

$$g(S) = w * \sum_{(v,s_v)\in S} (|s_v| - 1)$$

où w est le poids d'un éclatement de sommet.

Le choix du poids w des éclatements de sommets peut changer radicalement la difficulté d'une instance. Quand ce poids est nul, le problème est résolu de façon trivial : il est possible de faire autant d'éclatements de sommets que nécessaire pour retrouver une étiquette de sommet ou d'arc. Quand ce poids est très élevé, les solutions optimales ne peuvent pas éclater de sommets et le problème devient alors un problème d'appariement univoque de sommets généralement plus simple qu'un problème d'appariement multivoque. Lorsqu'un poids "intermédiaire" est choisi, le

problème est plus difficile : il est nécessaire de trouver un équilibre entre retrouver des étiquettes de sommets et d'arcs et éclater des sommets.

Afin de mesurer la capacité des algorithmes à trouver cet équilibre, nous les avons testés avec deux différents poids "intermédiaires". Le premier poids w est égal à 1, si bien qu'il y a généralement un grand nombre d'éclatements de sommets dans la solution optimale. Le second poids w est égal à 3, si bien qu'il y a relativement peu d'éclatements de sommets dans la solution optimale.

Les résultats obtenus par l'algorithme tabou réactif sont comparés à ceux obtenus par l'algorithme `ANT-GM'06` (avec et sans hybridation avec de la recherche locale) de [SSSG06a, SSJ06a] et déjà utilisé pour les problèmes d'appariement non-bijectifs.

Définition de notre mesure

Nous avons vu en partie 1 de cette thèse que quel que soit la façon dont les fonctions f et g de la similarité de Champin et Solnon est définie, il est possible de paramétrer notre mesure de distance de graphe afin de trouver les mêmes meilleurs appariements. Dans le cas général, la définition de cette mesure est relativement compliquée. Néanmoins, si la fonction f est une simple fonction de cardinalité et que la fonction g est définie comme ci dessus, la définition de notre mesure est beaucoup plus simple. Les fonctions de distance de sommets et d'arcs doivent renvoyer une valeur proportionnelle au nombre d'étiquettes non retrouvées et la fonction de distance des sommets doit également prendre en compte les éclatements de sommets.

De façon plus formelle, étant donnés deux graphes multi-étiquetés $G = (V, r_V, r_E)$ et $G' = (V', r'_V, r'_E)$ définis sur les ensembles d'étiquettes de sommets et d'arcs L_V et L_E, il faut calculer la distance $\delta =< \delta_v, \delta_e, \otimes_{\sum} >$ entre les graphes $G_2 = (V, E)$ et $G'_2 = (V', E')$ telle que :

- E est l'ensemble des arcs du graphe G, c.-à-d. que $E = \{(u, v) \in V^2 | \exists l \in L_E \wedge (u, v, l) \in r_E\}$;
- E' est l'ensemble des arcs du graphe G, c.-à-d. que $E' = \{(u', v') \in V'^2 | \exists l \in L_E \wedge (u', v', l) \in r'_E\}$;
- δ_v compte pour chaque sommet le nombre d'étiquettes non retrouvées et ajoute le nombre de sommets éclatés, c.-à-d. que $\forall v \in V, \delta_v(v, s_v) = |\{(v, l) | (v, l) \in r_V, \nexists v' \in s_v, (v', l) \in r'_V\}| + max(0, (|s_v| - 1) * k)$ et que $\forall v \in V', \delta_v(v, s_v) = |\{(v, l) | (v, l) \in r'_V, \nexists v' \in s_v, (v', l) \in r_V\}| + max(0, (|s_v| - 1) * k)$;
- δ_e compte pour chaque arc le nombre d'étiquettes non retrouvées, c.-à-d. que $\forall (u, v) \in E, \delta_e(u, v, s_e) = |\{(u, v, l) | (u, v, l) \in r_E, \nexists (u', v') \in s_e, (u', v', l) \in r'_E\}|$ et que $\forall (u, v) \in E', \delta_e(u, v, s_e) = |\{(u, v, l) | (u, v, l) \in r'_E, \nexists (u', v') \in s_e, (u', v', l) \in r_E\}|$.

Jeu d'essais

Nous avons utilisé un générateur aléatoire de paires de graphes "similaires" : un graphe est tout d'abord généré aléatoirement puis 5 fusions et éclatements de sommets et 10 suppressions et insertions d'arcs et de sommets (tous choisis aléatoirement) lui sont appliquées afin de générer un deuxième graphe similaire au premier.

Lorsque les composants des graphes ont beaucoup d'étiquettes différentes, le meilleur appariement est généralement trouvé de façon triviale. Par conséquent, afin d'obtenir des instances plus difficiles, les arcs et les sommets des graphes que nous avons générés ont tous la même étiquette. Nous avons généré 100 graphes similaires. Ces graphes ont entre 80 et 100 sommets et entre 200 et 360 arcs. Le second graphe a été obtenu en faisant 5 fusions ou éclatements de sommets et 10 insertions ou suppressions de sommets ou d'arcs.

Le nombre de transformations faites nous permet de déterminer une borne minimum de la similarité des deux graphes. Une instance est considérée comme résolue quand cette borne est trouvée. Sur les 100 instances générées, 87 sont relativement simples dans le sens où elles sont résolues assez rapidement par l'algorithme Glouton. Par conséquent, nous n'avons gardé que les 13 instances les plus difficiles, c.-à-d. celles pour lesquelles l'algorithme Glouton n'arrivait pas à trouver la borne minimum.

Paramétrage d'ANT-GM'06

ANT-GM'06 a été testé sur ces instances avec de très nombreux paramétrages différents. Nous avons gardé le meilleur paramétrage, c.-à-d. celui qui donnait le meilleur résultat en moyenne sur toutes les instances. Certains paramètres, tels que le taux minimum et maximum de la phéromone et le nombre de fourmis de la colonie ont été choisis en fonction de l'expérience des auteurs de ANT-GM'06 sur le paramétrage des algorithmes ACO pour la résolution d'autres types de problèmes. Le taux d'évaporation, le poids du facteur heuristique et du facteur phéromonal ont été déterminés expérimentalement en essayant de trouver le meilleur compromis diversification/instensification.

Le poids α du facteur phéromonal a été fixé à 1. Le taux d'évaporation de la phéromone ρ est fixé à 2%, le poids du facteur heuristique est fixé à 10, le nombre de cycles maximum est fixé à 1000. Le nombre de fourmis de la colonie est 20. Les bornes τ_{min} et τ_{max} de la quantité de phéromone sont fixées à $0,01$ et 6.

Afin d'évaluer l'intérêt d'hybrider l'optimisation par colonies de fourmis avec de la recherche locale, ANT-GM'06 est testé avec et sans phase d'amélioration des appariements par recherche locale. Ces algorithmes sont respectivement notés ANT-GM'06+LS et ANT-GM'06.

Paramétrage de RTS

Tabou réactif a été testé sur ces instances avec de très nombreux paramétrages différents. Nous avons gardé le meilleur paramétrage, c.-à-d. celui qui donnait le meilleur résultat en moyenne sur toutes les instances. Les plages de valeurs testées pour les paramètres ont été déterminées en fonction de résultats obtenus par Tabou.

La longueur *min* de la liste taboue est fixée à 15, la longueur maximale *max* à 50, la liste est rallongée de 15 en cas de collision et réduite tous les 5000 mouvements sans redondance. La recherche fait 50000 mouvements et, comme 50000 mouvements de recherche locale sont réalisés beaucoup plus rapidement que 1000 cycles de ANT-GM'06, notre recherche locale est réitérée à partir de différents appariements initiaux. Le nombre d'itérations de notre recherche locale taboue est fixé de telle sorte que le temps d'exécution soit le même pour ACO et pour Tabou.

Pour chacun des algorithmes, les résultats sont donnés pour au moins 20 exécutions de chaque instance.

Résultats

Le tableau 5.5 présente les résultats obtenus sur les 13 instances de notre problème avec un poids des éclatements de sommets fixé à 1. Le *score* correspond au nombre d'étiquettes que le meilleur appariement trouvé permet de retrouver auquel est retranché le coût des éclatements de sommets.

Tout d'abord, chacun de ces algorithmes trouve la borne minimum du *score* calculée lors de la génération des instances : les scores trouvés sont même généralement légèrement supérieurs à cette borne.

Problème				RTS			ANT-GM'06			ANT-GM'06+LS		
Nbr	($\mid V_1 \mid$,$\mid E_1 \mid$)	($\mid V_2 \mid$,$\mid E_2 \mid$)	L	Best	Moy	T.	Best	Moy	T.	Best	Moy	T.
Poids des splits fixé à 1												
1	(80, 200)	(74, 186)	1512	511	511.00	57	511	511.00	131	**512**	**511.10**	140
2	(80, 240)	(82, 261)	1415	644	644.00	60	644	644.00	266	644	644.00	239
3	(80, 320)	(83, 362)	1445	821	820.97	279	821	820.50	498	**822**	**821.20**	660
4	(80, 340)	(72, 302)	1174	753	753.00	55	753	753.00	111	753	753.00	130
5	(80, 360)	(77, 367)	1139	**856**	**855.97**	187	855	855.00	321	855	855.00	249
6	(80, 360)	(78, 367)	1196	863	863.00	21	863	863.00	187	**864**	**863.94**	565
7	(90, 300)	(91, 307)	1670	762	762.00	98	762	762.00	326	762	762.00	213
8	(90, 320)	(87, 310)	1611	780	780.00	51	780	780.00	572	780	780.00	409
9	(90, 320)	(90, 339)	1716	816	**816.00**	69	816	815.45	546	816	815.45	602
10	(100, 260)	(96, 263)	2093	697	696.63	628	697	696.90	976	697	**697.00**	812
11	(100, 300)	(100, 304)	2078	780	780.00	148	780	780.00	278	780	780.00	279
12	(100, 320)	(98, 331)	2080	828	828.00	46	828	828.00	286	828	828.00	218
13	(100, 360)	(99, 371)	2455	915	915.00	90	915	915.00	267	915	915.00	152
Poids des splits fixé à 3												
1	(80, 200)	(74, 186)	659	496	496.00	28	496	496.00	132	496	496.00	121
2	(80, 240)	(82, 261)	798	624	624.00	26	624	624.00	108	624	624.00	88
3	(80, 320)	(83, 362)	896	801	801.00	17	801	801.00	213	801	801.00	218
4	(80, 340)	(72, 302)	737	732	732.00	27	732	732.00	185	732	732.00	194
5	(80, 360)	(77, 367)	852	846	846.00	198	846	846.00	116	846	846.00	77
6	(80, 360)	(78, 367)	855	840	840.00	36	840	840.00	94	840	840.00	67
7	(90, 300)	(91, 307)	1140	748	748.00	82	748	748.00	186	748	748.00	150
8	(90, 320)	(87, 310)	1079	766	766.00	44	766	766.00	187	766	766.00	187
9	(90, 320)	(90, 339)	1127	802	802.00	70	802	802.00	167	802	802.00	163
10	(100, 260)	(96, 263)	1346	683	683.00	114	683	682.75	556	683	683.00	354
11	(100, 300)	(100, 304)	1466	769	769.00	358	769	769.00	274	769	769.00	285
12	(100, 320)	(98, 331)	1463	814	814.00	51	814	814.00	241	814	814.00	201
13	(100, 360)	(99, 371)	1528	900	900.00	54	900	900.00	245	900	900.00	243

TAB. 5.5 – Résultats sur les problèmes d'appariement multivoque de graphes. Pour chaque instance, le tableau précise le nombre de sommets et d'arcs des deux graphes, la limite de temps CPU L pour chaque exécution (sur un Pentium IV 1.7 GHz). Pour chaque algorithme est précisé le meilleur *score* (Best) trouvé et le *score* moyen (Moy) obtenu pour 20 exécutions, le temps moyen en secondes (T.) nécessaire à l'obtention du meilleur *score*.

Nous pouvons également constater que les algorithmes sont robustes dans le sens où le *score* moyen sur 20 exécutions est généralement très proche du meilleur *score* trouvé (pour 9 instances sur 13, le *score* moyen est égal au meilleur *score* trouvé). La recherche locale taboue réactive donne de meilleurs résultats que `ANT-GM'06` : elle obtient un *score* meilleur que `ANT-GM'06` sur une instance et est toujours au moins aussi rapide. Cependant, l'ajout d'une procédure de recherche locale améliore les résultats de `ANT-GM'06` : `ANT-GM'06+LS` obtient un meilleur (resp. moins bon) *score* que RTS sur 3 (resp. 1) instances et les même résultats que RTS sur 9 instances. Sur ce jeu d'essais, `ANT-GM'06+LS` et RTS obtiennent donc des résultats complémentaires : `ANT-GM'06+LS` obtient plus fréquemment de meilleurs résultats que RTS mais RTS est plus rapide que `ANT-GM'06+LS`, même lorsque ces deux algorithmes obtiennent des *scores* identiques.

Le tableau 5.5 présente également les résultats sur les 13 instances avec un poids des éclatements de sommets fixé à 3. Sur chacune des instances, les trois algorithmes trouvent toujours le même meilleur *score* et le même *score* moyen (sauf `ANT-GM'06` sur une instance). Cependant, RTS trouve ces solutions en un temps plus court que `ANT-GM'06` et `ANT-GM'06+LS` à l'exception de deux instances. Ces résultats montrent donc que sur ces instances, il est nettement préférable d'utiliser la recherche locale taboue réactive.

5.3.5 Discussion

Performance de nos algorithmes

Nous proposons d'utiliser une recherche locale taboue et une version réactive de cette recherche locale pour calculer notre distance de deux graphes. Ces deux algorithmes sont génériques dans le sens où il peuvent être utilisés indépendemment de la façon dont la distance est paramétrée et peuvent donc être utilisés pour la résolution de nombreux problèmes différents.

Étant génériques, ces deux algorithmes n'utilisent aucune méthode de filtrage propre à un problème particulier. Par conséquent, sur les problèmes pour lesquels il existe des méthodes efficaces de filtrage de l'espace de recherche (isomorphisme de graphes ou de sous-graphe), nos algorithmes ne sont pas compétitifs avec des approches dédiées.

Sur des problèmes plus difficiles (par exemple la recherche du plus grand sous-graphe commun à deux graphes), nos algorithmes se montrent très performants. Les solutions retournées sont de très bonne qualité et nos algorithmes peuvent être utilisés sur des instances qui ne peuvent pas être résolues par des méthodes arborescentes complètes. Ainsi, quelques secondes suffisent à nos algorithmes pour trouver le plus grand sous-graphe (ou un graphe de taille proche) de deux graphes de 100 sommets là où des approches complètes ne peuvent pas résoudre des instances avec des graphes de plus de 30 sommets.

Sur des problèmes de recherche d'un meilleur appariement multivoque, nos algorithmes se montrent très performants par rapport aux approches complètes : ils peuvent résoudre des instances ayant des graphes d'une centaine de sommets là où les méthodes complètes ne peuvent dépasser la (petite) dizaine de sommets. Les résultats obtenus sur les problèmes de l'appariement non-bijectif de graphes de [BRB04] et les problèmes du calcul de la similarité de deux graphes de Champin et Solnon [CS03] montrent que notre algorithme de recherche locale réactive (RTS) obtient des résultats complémentaires avec l'algorithme `ANT-GM'06` à base de colonies de fourmis de Sammoud *et al.* [SSSG06a]. RTS obtient des résultats parfois légèrement moins bons que ceux de `ANT-GM'06` mais est plus rapide que ce dernier.

Paramétrage

Notre algorithme tabou a un seul paramètre : la longueur de la liste taboue qui permet de réguler l'intensification et la diversification de la recherche. Nous montrons (sur des problèmes d'isomorphisme de sous-graphe) que le réglage de ce paramètre est critique : la longueur optimale de la liste varie d'un type d'instance à un autre et un mauvais paramétrage dégrade fortement les résultats de Tabou.

Nous proposons un algorithme de recherche taboue réactive (RTS) pour le calcul de la distance entre deux graphes. Cet algorithme adapte la longueur de la liste taboue dynamiquement pendant la recherche selon les besoins de diversification et d'intensification de la recherche. Cet algorithme donne de meilleurs résultats que Tabou sur les problèmes d'isomorphisme de sous-graphe : le taux de réussite est globalement meilleur. En outre, RTS est beaucoup plus simple à paramétrer que Tabou : un paramétrage non optimal ne dégrade que peu les résultats.

Sur les problèmes de recherche d'un plus grand sous-graphe commun à deux graphes, la diversification d'une recherche locale sans heuristique taboue est suffisante et permet des résultats meilleurs qu'une recherche gloutonne. Par conséquent, utiliser une liste taboue dégrade les résultats. Néanmoins, notre recherche locale permet de trouver de bonnes solutions à des problèmes ne pouvant pas être résolus par une approche complète.

Nos expérimentations sur des problèmes de recherche d'un meilleur appariement multivoque de deux graphes montrent que sur les problèmes très difficiles, RTS est meilleur que Tabou. Néanmoins, la robustesse au paramétrage de Tabou réactif a ses limites et il est nécessaire de paramétrer correctement Tabou réactif pour obtenir des résultats aussi bons que les fourmis de l'algorithme `ANT-GM'06` (avec un paramétrage optimal).

En résumé, nos algorithmes permettent de résoudre efficacement de nombreux problèmes d'appariement de graphes difficiles. En particulier, ils permettent de résoudre (ou de proposer une bonne solution) en un temps très court des problèmes de recherche d'un plus grand sous-graphe commun à deux graphes qui ne sont pas résolus par des méthodes de recherche complètes. Malgré l'intégration d'un processus de réactivité pour réguler automatiquement l'intensification et la diversification de la recherche, le paramétrage reste le point faible de nos algorithmes.

6

Programmation par contraintes et isomorphisme de graphes

La programmation par contraintes est une solution élégante pour la résolution des problèmes combinatoires : le problème est exprimé de façon déclarative en termes de variables et de contraintes sur l'affectation de ces variables puis un solveur de contraintes générique se charge de la résolution du problème. Cependant, cette généricité a un coût : les problèmes sont souvent moins efficacement résolus par des solveurs de contraintes que par des algorithmes dédiés. En effet, alors que les algorithmes dédiés utilisent fortement la sémantique globale du problème pour filtrer l'espace de recherche de ces problèmes, cette sémantique est généralement perdue lors de la décomposition du problème en un ensemble de contraintes et les solveurs ne peuvent pas filtrer efficacement le problème.

Une façon de pallier cette perte d'efficacité consiste à utiliser des contraintes globales, c.-à-d. des contraintes qui expriment plus globalement la sémantique du problème, et des algorithmes de filtrage associés à ces contraintes. Nous présentons dans ce chapitre une contrainte globale dédiée à la résolution du problème de l'isomorphisme de graphes (un cas très particulier de comparaison de deux graphes) par la programmation par contraintes. Nous présentons un algorithme de filtrage associé à cette contrainte globale et nous montrons que cet algorithme permet à la programmation par contraintes d'être compétitive par rapport aux algorithmes dédiés au problème de l'isomorphisme de deux graphes.

6.1 Introduction

De nombreuses applications nécessitent de comparer des graphes afin de déterminer si leurs structures sont identiques : c'est le problème de l'isomorphisme de graphes (GIP pour Graph Isomorphism Problem), un cas très particulier de la comparaison de deux graphes. Puget [Pug05] et Zampelli *et al.* [ZDD06] montrent également que ce problème peut être utilisé pour détecter des symétries dans des problèmes de satisfaction de contraintes.

Il existe de nombreux algorithmes dédiés au GIP ([Ull76, McK81, CFSV00, CFCS01]). Ces algorithmes sont très efficaces en pratique, même si leur complexité dans le pire des cas est exponentielle. Cependant, ces algorithmes sont difficilement utilisables pour résoudre des problèmes plus spécifiques, par exemple un GIP auquel des contraintes supplémentaires sont ajoutées.

Une alternative intéressante à ces algorithmes dédiés est l'utilisation de la programmation par contrainte (PPC) qui fournit un cadre générique pour la résolution des problèmes de satisfaction de contraintes (CSP). En effet, les GIPs peuvent être aisément transformés en CSPs

[McG79, McG82] et il est possible d'utiliser un solveur générique de contraintes pour les résoudre. Cependant, la transformation d'un GIP en CSP, entraîne la perte de la sémantique globale du problème et la PPC est moins efficace pour résoudre les problèmes d'isomorphisme que les algorithmes dédiés qui gardent la sémantique globale du problème.

Le but de notre travail est de permettre aux solveurs de contraintes d'appréhender la sémantique des GIPs de façon globale afin qu'ils puissent les résoudre efficacement sans pour autant perdre la souplesse de la PPC. Pour cela, nous introduisons une contrainte globale dédiée aux GIPs et nous proposons une consistance partielle pour cette contrainte globale (la Iterated Distance Label consistance ou IDL-consistance) ainsi que l'algorithme permettant de l'établir.

Plan du chapitre. En section 2, nous introduisons le problème de l'isomorphisme de graphes et la façon dont il peut être modélisé en un problème de satisfaction de contraintes. Cela nous amène à proposer une nouvelle contrainte globale dédiée au problème de l'isomorphisme de graphes non-orientés. En section 3, nous définissons une nouvelle consistance pour cette contrainte, la *d-Iterated Distance Label* consistance (d-IDL-consistance) et un algorithme de filtrage permettant de l'établir. Cette consistance est basée sur un réétiquetage des sommets des graphes consistant par rapport à un problème d'isomorphisme et basé sur les propriétés d'un sommet v par rapport aux autres sommets du graphe distants d'au plus d arêtes du sommet v. Cette consistance est une généralisation des deux consistances partielles que nous avons proposé dans [SS04b, SS06] : la label-consistance (qui correspond à la $+\infty$-IDL-consistance) et à l'ILL-consistance (qui correspond à la 1-IDL-consistance). En section 4, nous présentons un exemple complet de chacune de ces deux consistances. En section 5, nous discutons de l'extension de ce travail aux graphes orientés et aux graphes étiquetés. En section 6, nous testons expérimentalement ces deux consistances partielles et les comparons aux résultats obtenus par Nauty [McK81], le meilleur algorithme connu pour le problème de l'isomorphisme de graphes.

6.2 Le problème de l'isomorphisme de graphes

6.2.1 Algorithmes dédiés

Il est possible de résoudre un GIP en recherchant directement un appariement des sommets des deux graphes. L'espace de recherche composé de tous les appariements est exploré par *séparation et évaluation* et l'élagage de l'arbre de recherche se fait en exploitant les propriétés des graphes manipulés [CFSV00, CFCS01, Ull76] (distribution des arcs, voisinage des sommets...). Cette approche, très efficace, permet de résoudre des problèmes de plus de 1000 sommets très rapidement (moins d'une seconde). Dans [SD76], Schmidt *et al.* propose un algorithme de ce type s'aidant de la matrice des distances des sommets d'un graphe pour élaguer efficacement l'arbre de recherche.

Brendan McKay [McK81] propose une autre approche qui consiste à calculer une représentation canonique des deux graphes à comparer telle que deux graphes ont la même représentation si et seulement si ces deux graphes sont isomorphes (voir la section 4.2 de cette thèse). Dans le cas général, Nauty est l'algorithme existant le plus rapide pour la résolution du problème de l'isomorphisme de graphes. Darga *et al.* [DLSM04] proposent un algorithme similaire à Nauty et appelé Saucy. Cet algorithme est dédié aux graphes creux (c.-à-d. de faible densité) et est beaucoup plus rapide que Nauty sur ce type de graphes. Puget [Pug05] propose un algorithme nommé Autom encore plus rapide sur ce type de graphes.

Si les algorithmes dédiés sont très efficaces pour résoudre des GIPs (malgré leur complexité exponentielle dans le pire des cas), ils ne permettent pas la résolution de problèmes plus spécifiques

tels que des GIPs auxquels des contraintes sont ajoutées. La programmation par contraintes est donc une solution attractive pour modéliser simplement des problèmes plus complexes.

6.2.2 Programmation par contraintes et GIP

La programmation par contraintes (PPC) est un outil générique pour la résolution de problèmes de satisfaction de contraintes (CSP). Un GIP peut donc être modélisé sous forme de CSP [GJ79, Rég95] puis résolu par un solveur de contraintes (par exemple CHOCO [Lt00], Ilog solver [ILO00], CHIP [AB93]...).

Un GIP peut être aisément formulé en CSP binaire (c.-à-d. un CSP qui ne contient que des contraintes ne portant que sur exactement deux variables). Étant donnés deux graphes $G = (V, E)$ et $G' = (V', E')$, nous définissons le CSP (X, D, C) suivant :

- une variable x_u est associée à chaque sommet $u \in V$, c.-à-d. que $X = \{x_u / u \in V\}$;
- le domaine de chaque variable x_u est l'ensemble des sommets de G' ayant le même nombre d'arcs entrants et le même nombre d'arcs sortants que u, c.-à-d. que :

$$D(x_u) = \{u' \in V' \mid \begin{array}{l} |\{(u,v) \in E\}| = |\{(u',v') \in E'\}| \text{ et} \\ |\{(v,u) \in E\}| = |\{(v',u') \in E'\}| \} \end{array}$$

- il existe une contrainte binaire $C_{edge}(x_u, x_v)$ entre chaque paire de variables (x_u, x_v) exprimant le fait que les sommets de G' affectés à x_u et x_v doivent être connectés par un arc si et seulement si $(u, v) \in E$:,

$$\begin{array}{ll} \text{si } (u,v) \in E, & C_{edge}(x_u, x_v) = E' \\ \text{sinon} & C_{edge}(x_u, x_v) = \{(u',v') \in V'^2 \mid u' \neq v' \text{ et } (u',v') \notin E'\} \end{array}$$

Discussion. Lors de la formulation d'un GIP en CSP, la sémantique globale du problème est décomposée en un ensemble de contraintes binaires d'arcs (C_{edge}), chacune d'elles exprimant localement la présence ou l'absence d'un arc. Cela a pour conséquence de rendre la PPC moins efficace que les algorithmes dédiés.

Afin d'améliorer la résolution des CSPs associés à des GIPs, il est possible d'ajouter une contrainte globale *allDiff* [Rég95]. La contrainte *allDiff* permet d'exprimer de façon globale qu'un ensemble de variables doivent avoir des valeurs différentes deux à deux.

Dans ce chapitre, nous introduisons une nouvelle contrainte globale pour modéliser les GIPs. Cette contrainte n'est pas sémantiquement globale, elle peut être remplacée par l'ensemble des contraintes binaires présenté plus haut. Cependant, elle permet de considérer les arcs des graphes de façon globale pour pouvoir filtrer plus efficacement l'espace de recherche. Notons que cette contrainte peut être combinée avec la contrainte *allDiff* afin de filtrer encore plus de valeurs.

6.2.3 Une contrainte globale pour les GIP

Nous introduisons ici une nouvelle contrainte globale dédiée aux problèmes d'isomorphisme de graphes. Syntaxiquement, cette contrainte est définie par la relation $gip(V, E, V', E', L)$ où

- V et V' sont deux ensembles de valeurs tels que $|V| = |V'|$;
- $E \subseteq V \times V$ est un ensemble de couples de valeurs de V,
- $E' \subseteq V' \times V'$ est un ensemble de couples de valeurs de V' ;
- L est un ensemble de couples qui associent une variable différente du CSP à chaque valeur de V, c.-à-d. que L est un ensemble de $|V|$ couples (x_u, u) où x_u est une variable du CSP, u une valeur de V et que pour toute paire de couples (x_u, u) et (x_v, v) différents de L, x_u et x_v sont des variables différentes et $u \neq v$.

Sémantiquement, la contrainte globale $gip(V, E, V', E', L)$ est consistante si et seulement s'il existe une fonction d'isomorphisme $f : V \rightarrow V'$ telle que pour chaque couple $(x_u, u) \in L$, il existe une valeur $u' \in D(x_u)$ telle que $u' = f(u)$.

Cette contrainte globale n'est pas sémantiquement globale dans le sens où elle est sémantiquement équivalente à l'ensemble des contraintes binaires décrit en section 6.2.2. La contrainte *gip* nous permet cependant d'utiliser la sémantique globale des GIPs afin de les résoudre plus efficacement.

Nous proposons dans la section suivante, une consistance partielle associée à la contrainte globale *gip* et un algorithme de filtrage qui lui est associé. Nous décrivons également comment propager efficacement les réductions de domaines des variables soumises à cette contrainte.

6.3 d-Iterated Distance Label consistance

Une fonction d'isomorphisme n'associe que des sommets *similaires*. Les méthodes basées sur les *invariants de sommets* utilisent cette propriété pour filtrer l'espace de recherche d'un GIP. Un invariant de sommet est une étiquette $l(v)$ affectée à un sommet v telle que, s'il existe une fonction d'isomorphisme associant v et v', alors $l(v) = l(v')$ (l'inverse n'étant cependant pas toujours vrai). L'exemple le plus simple d'invariants de sommets est le degré d'un sommet (c.-à-d. ses nombres d'arcs entrants et sortants) : si f est une fonction d'isomorphisme entre $G = (V, E)$ et $G' = (V', E')$, alors pour chaque sommet $v \in V$, les sommets v et $f(v)$ ont le même degré.

Nous introduisons dans cette section quelques définitions et théorèmes utilisés par la suite pour définir un nouvel invariant de sommets basé sur les relations en termes de distance d'un sommet avec les autres sommets du graphe. Étiqueter les sommets des graphes par cet invariant puis propager cet étiquetage de façon itérative sur les sommets du graphe permet de définir une consistance partielle pour la contrainte *gip*.

Nous focaliserons notre attention sur les graphes non-orientés, c.-à-d. des graphes avec des arêtes non-orientées (les arêtes (u, v) et (v, u) sont considérées comme identiques). L'extension de notre travail aux graphes orientés est discutée en section 6.5. Nous supposerons que les graphes sont connexes, c.-à-d. que chaque sommet est accessible à partir de tous les sommets.

6.3.1 Fonctions d'étiquetage et de réétiquetage isomorphe-consistantes

Définition (Fonction d'étiquetage). Une fonction d'étiquetage est une fonction notée α qui, étant donné un graphe $G = (V, E)$ associe une étiquette $\alpha_G(v)$ à chaque sommet $v \in V$ de ce graphe. Nous notons $image(\alpha_G)$ l'ensemble des étiquettes retournées par α pour les sommets du graphe $G = (V, E)$:

$$image(\alpha_G) \quad = \quad \{l | \exists v \in V, \alpha_G(v) = l\}$$

Définition (Fonction d'étiquetage isomorphe-consistante). Une fonction d'étiquetage α est isomorphe-consistante si pour chaque paire de graphes isomorphes $G = (V, E)$ et $G' = (V', E')$, et pour toute fonction d'isomorphisme f entre G et G', les sommets mis en correspondance par f ont la même étiquette, c.-à-d. que $\forall v \in V, \alpha_G(v) = \alpha_{G'}(f(v))$.

Exemple. Soit la fonction d'étiquetage α^{deg} qui associe à chaque sommet son degré, c.-à-d. que :

$$\forall v \in V, \alpha^{deg}_{G=(V,E)} \quad = \quad |\{u \in V | (u, v) \in E\}|$$

La fonction d'étiquetage α^{deg} est isomorphe-consistante car une fonction d'isomorphisme n'associe que des sommets ayant le même nombre de sommets adjacents.

Une fonction d'étiquetage isomorphe-consistante permet de réduire le domaine des variables d'un CSP associé à un GIP : le domaine de chaque variable x_u correspondant à un sommet u peut être réduit à l'ensemble des sommets ayant la même étiquette que u.

Définition (Fonction d'étiquetage isomorphe-consistante au moins aussi forte). Une fonction d'étiquetage isomorphe-consistante α est au moins aussi forte qu'une autre fonction d'étiquetage isomorphe-consistante α' si son étiquetage discrimine au moins aussi fortement les sommets du graphe, c.-à-d. si étant donné un graphe G :

$$\forall (u,v) \in V^2, \alpha'_G(u) \neq \alpha'_G(v) \quad \Rightarrow \quad \alpha_G(u) \neq \alpha_G(v)$$

Comme une fonction d'étiquetage discrimine d'autant plus les sommets d'un graphe qu'elle est forte, le filtrage des variables d'un CSP associé à un GIP sera d'autant plus important que la fonction d'étiquetage utilisée est forte.

Définition (Fonction de réétiquetage). Une fonction de réétiquetage est une fonction notée β qui, étant donnée une fonction d'étiquetage α retourne une autre fonction d'étiquetage $\beta(\alpha)$.

Définition (Fonction de réétiquetage isomorphe-consistante). Une fonction de réétiquetage β est isomorphe-consistante si, quelle que soit la fonction d'étiquetage α, si α est isomorphe-consistante, alors $\beta(\alpha)$ l'est aussi.

Exemple. Étant donnée une fonction d'étiquetage α, la fonction α_{adj} associant à chaque sommet v l'ensemble des étiquettes α des sommets adjacents de v est une fonction de réétiquetage isomorphe-consistante car deux sommets peuvent être associés par une relation d'isomorphisme si et seulement si leurs voisins peuvent l'être.

Réétiqueter les sommets d'un graphe à partir d'une fonction d'étiquetage α est intéressant s'il permet d'obtenir une fonction d'étiquetage $\beta(\alpha)$ (strictement) plus forte que α. Notons que, comme $\beta(\alpha)$ est également une fonction d'étiquetage, il est possible de réitérer le réétiquetage pour obtenir des fonctions d'étiquetage (éventuellement encore plus fortes) $\beta(\beta(\alpha))$, $\beta(\beta(\beta(\alpha)))$...

6.3.2 Réétiquetage basé sur les distances

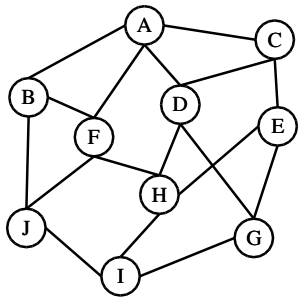
Nous présentons dans cette section quelques propriétés et théorèmes relatifs au problème de l'isomorphisme de graphes nous permettant de définir une fonction de réétiquetage isomorphe-consistante.

Définition (Chaîne dans un graphe). Soit un graphe $G = (V, E)$, une *chaîne* entre deux sommets u et v est une suite $< v_0, v_1, v_2, ..., v_k >$ de sommets telle que $v_0 = u$, $v_k = v$ et telle que pour tout $i \in [1, k]$, $(v_{i-1}, v_i) \in E$. La longueur d'une chaîne π, notée $|\pi|$, est son nombre d'arêtes, c.-à-d. k.

Définition (Plus courte chaîne et distance entre deux sommets). Soit un graphe $G = (V, E)$, une *plus courte chaîne* entre deux sommets u et v est une chaîne entre u et v de longueur minimale. La *longueur d'une plus courte chaîne* entre u et v est notée $\delta_G(u, v)$. Nous dirons que $\delta_G(u, v)$ est la *distance* entre u et v.

Définition (Diamètre d'un graphe). Le *diamètre d'un graphe* est la plus grande distance existante entre deux de ses sommets. Si le diamètre d'un graphe $G = (V, E)$ n'est pas infinie, le graphe est connexe et ce diamètre est plus petite ou égale à $|V|$.

Théorème 1. Soient deux graphes $G = (V, E)$ et $G' = (V', E')$ tels que $|V| = |V'|$, et une fonction bijective $f : V \to V'$, les deux propositions suivantes sont équivalentes :



$\delta_G(u,v)$	A	B	C	D	E	F	G	H	I	J
A	0	1	1	1	2	1	2	2	3	2
B	1	0	2	2	3	1	3	2	2	1
C	1	2	0	1	1	2	2	2	3	3
D	1	2	1	0	2	2	1	1	2	3
E	2	3	1	2	0	2	1	1	2	3
F	1	1	2	2	2	0	3	1	2	1
G	2	3	2	1	1	3	0	2	1	2
H	2	2	2	1	1	1	2	0	1	2
I	3	2	3	2	2	2	1	1	0	1
J	2	1	3	3	3	1	2	2	1	0

FIG. 6.1 – Un graphe $G = (V, E)$ et les distances entre chaque couple de sommets. La plus grande distance existante entre ses couples de sommets étant 3, le diamètre de ce graphe est de 3.

1. f est une fonction d'isomorphisme, c.-à-d. telle que $\forall (u,v) \in V^2, (u,v) \in E \Leftrightarrow (f(u), f(v)) \in E'$;
2. $\forall (u,v) \in V^2, \delta_G(u,v) = \delta_{G'}(f(u), f(v))$.

Démonstration. (1) $\Rightarrow$ (2) : si f est une fonction d'isomorphisme, alors (u,v) est une arête de G si et seulement si $(f(u), f(v))$ est une arête de G' donc $< v_1, v_2, ..., v_n >$ est une chaîne dans G si et seulement si $< f(v_1), f(v_2), ..., f(v_n) >$ est une chaîne dans G', et par conséquent $< v_1, v_2, ..., v_n >$ est une plus courte chaîne dans G si et seulement si $< f(v_1), f(v_2), ..., f(v_n) >$ est une plus courte chaîne dans G'. La propriété 2 est donc vérifiée.
(2) $\Rightarrow$ (1) : Pour chaque paire de sommets $(u,v) \in V \times V$, si (u,v) est une arête de G, alors $< u, v >$ est la plus courte chaîne entre u et v, $\delta_G(u,v) = 1$ et donc $\delta_{G'}(f(u), f(v)) = 1$. $(f(u), f(v))$ est donc un arc de G' (et vice versa). □

Le théorème 1 montre que les fonctions d'isomorphisme préservent les distances entre les sommets des graphes. Nous pouvons utiliser cette propriété pour définir une fonction de réétiquetage β. À partir d'un étiquetage α, la fonction de réétiquetage β_d retourne une nouvelle fonction d'étiquetage $\beta_d(\alpha)$ étiquetant chaque sommet v par son étiquette $\alpha(v)$ et par les étiquettes $\alpha(u)$ des sommets u tels que $\delta_G(u,v) \leq d$.

Définition (β_d). Étant donnés un graphe $G = (V, E)$, une fonction d'étiquetage α et un entier positif d, nous définissons la fonction de réétiquetage $\beta_{d,G}(\alpha) : V \to \wp(\mathbb{N} \times \mathbb{N}^* \times image(\alpha_G))$:

$$\forall v \in V, \beta_{d,G}(\alpha)(v) = \{(j,k,l) | \; j \in \mathbb{N}, k \in \mathbb{N}^*, l \in image(\alpha_G), \\ k = |\{u \in V | j = \delta(u,v) \wedge j \leq d \wedge l = \alpha_G(u)\}| \wedge k \neq 0\}$$

Étant donné un entier d et une fonction d'étiquetage α, la fonction $\beta_d(\alpha)$ associe à chaque sommet u d'un graphe l'ensemble des triplets (j,k,l) exprimant le fait qu'il y a k sommets à distance j du sommet u (cette distance j devant être plus petite ou égale à d) étiquetés par l par la fonction α.

Exemple. En prenant le graphe de la figure 6.1, et en choisissant la fonction α qui associe à chaque sommet son degré (c.-à-d. que $\alpha(A) = \alpha(D) = \alpha(F) = \alpha(H) = 4$ et que $\alpha(B) = \alpha(C) = \alpha(E) = \alpha(G) = \alpha(I) = \alpha(J) = 3$), la fonction $\beta(\alpha)_{2,G}$ associe un ensemble de 5 triplets au sommet A :

$$\beta_{2,G}(\alpha)(A) = \{(0,1,4), (1,2,3), (1,2,4), (2,1,4), (2,3,3)\}$$

Le premier triplet exprime le fait qu'il y a 1 sommet (le sommet A lui même) à distance 0 du sommet A étiqueté par 4. Le second triplet exprime le fait qu'il y a deux sommets (les sommets

B et C) distants de 1 du sommet A ayant pour étiquette 3. Le troisième triplet exprime le fait qu'il y a deux sommets (les sommets D et F) également à distance de 1 du sommet A mais ayant cette fois l'étiquette 4. Le quatrième triplet exprime le fait qu'il y a 1 sommet (le sommet H) à distance de 2 du sommet A ayant pour étiquette 4. Enfin, le dernier triplet exprime le fait qu'il y a 3 sommets (les sommets E, G et J) à distance de 2 du sommet A et étiquetés par 3.

Notons que dans cet exemple, la distance d est limitée à 2. Par conséquent, il n'y a pas de triplet exprimant le fait que le sommet I a pour étiquette 3 car le sommet I est à distance de 3 du sommet A.

Théorème 2. Quel que soit l'entier d, la fonction β_d est une fonction de réétiquetage isomorphe-consistante.

Démonstration. Si α est une fonction isomorphe-consistante, alors, pour toute paire de graphes $G = (V, E)$ et $G' = (V', E')$ et pour toute fonction d'isomorphisme f entre G et G', $\forall u \in V, \alpha_G(u) = \alpha_{G'}(f(u))$. De plus, comme f est une fonction d'isomorphisme et étant donné le théorème 1, $\forall (u, v) \in V^2, \delta_G(u, v) = \delta_{G'}(f(u), f(v))$. Par conséquent, $\forall u \in V, \forall l \in image(\alpha_G), \forall j \in [0, |V|], |\{v | v \in V \wedge \delta(u, v) = j \wedge \alpha_G(v) = l\}| = |\{v' | v' \in V' \wedge \delta(f(u), v') = j \wedge \alpha_{G'}(v') = l\}|$ (car f est une fonction bijective). Par conséquent, $\forall u \in V, \beta_{d,G}(\alpha)(u) = \beta_{d,G'}(\alpha)(f(u))$ et le théorème est valide. □

Une conséquence directe du théorème 2 est que la fonction β_d peut être utilisée pour étendre les étiquettes des sommets de deux graphes G et G' sans changer les propriétés d'isomorphisme entre les deux graphes G et G'.

Le théorème suivant montre que, quels que soient l'entier d et la fonction d'étiquetage α isomorphe-consistante, la fonction d'étiquetage $\beta_d(\alpha)$ est au moins aussi forte que la fonction α et qu'une fonction d'étiquetage $\beta_d(\alpha)$ est d'autant plus forte que la distance maximale d est élevée.

Théorème 3. Étant donnés un graphe $G = (V, E)$, une fonction d'étiquetage α et deux entiers k et l tels que $0 \leq k \leq l$, les fonctions $\beta_k(\alpha)$ et $\beta_l(\alpha)$ sont telles que : $\forall v \in V, |\{u | u \in V, \beta_{l,G}(\alpha)(u) = \beta_{l,G}(\alpha)(v)\}| \leq |\{u | u \in V, \beta_{k,G}(\alpha)(u) = \beta_{k,G}(\alpha)(v)\}| \leq |\{u | u \in V, \alpha_G(u) = \alpha_G(v)\}|$.

Démonstration. Étant donnés un graphe $G = (V, E)$ et une fonction d'étiquetage $\alpha, \forall v \in V, \forall d \in \mathbb{N}, \alpha_G(v)$ entre dans la composition de $\beta_{d,G}(\alpha)(v)$ (car $d \geq 0$ et que tout sommet est à distance de 0 de lui-même). Par conséquent, $\forall (u, v) \in V^2, \alpha_G(u) \neq \alpha_G(v) \Rightarrow \beta_{d,G}(\alpha)(u) \neq \beta_{d,G}(\alpha)(v)$. En outre, étant donnée la définition de β, $\forall v \in V, \forall k, l \in \mathbb{N}$ tels que $k \leq l$, $\beta_{k,G}(\alpha)(v) \subseteq \beta_{l,G}(v)$. Par conséquent, $\forall (u, v) \in V^2, \beta_{k,G}(\alpha)(u) \neq \beta_{k,G}(\alpha)(v) \Rightarrow \beta_{l,G}(\alpha)(u) \neq \beta_{l,G}(\alpha)(v)$. Le théorème est donc valide. □

Le théorème 3 montre qu'il est possible de réduire le domaine des variables d'un CSP correspondant à un GIP en réétiquetant les sommets des deux graphes avec la fonction β_d et en enlevant du domaine d'une variable x_v correspondant au sommet v d'un graphe G tous les sommets du graphe G' n'ayant pas la même étiquette que v. La fonction β_d permet donc de filtrer le domaine des variables. En outre, plus la distance maximale d est élevée et plus le filtrage est important.

6.3.3 Réétiquetage itératif et point fixe

Réétiqueter les sommets des graphes avec la fonction β permet de réduire le domaine des variables d'un CSP correspondant à un GIP entre les deux graphes. Ces réductions de domaine

peuvent être propagées en réitérant le réétiquetage jusqu'à l'obtention d'un point fixe, c.-à-d. jusqu'à ce qu'un réétiquetage n'augmente plus le nombre d'étiquettes de sommets différentes.

En démarrant d'un étiquetage isomorphe consistant initial α_d^0 (associant par exemple la même étiquette $\emptyset$ à tous les sommets), nous définissons la suite α_d^1, α_d^2, α_d^3... d'étiquetage de sommets telle que chaque rang k de cette suite correspond à un réétiquetage par la fonction β_d basé sur les étiquettes des sommets au rang $k-1$. Plus formellement, étant donné un graphe $G = (V, E)$ et un entier d :

$$\forall v \in V, \alpha_{d,G}^0(v) = \emptyset$$
$$\forall v \in V, \alpha_{d,G}^k(v) = \beta_{d,G}(\alpha^{k-1})(v)$$

Le théorème 2 prouve que chaque rang de la suite α^k est une fonction d'étiquetage isomorphe consistante et le théorème 3 prouve que chaque fonction d'étiquetage α_d^{k+1} est au moins aussi forte que la précédente α_d^k. Finalement, le théorème 4 montre que cette séquence atteint un point fixe, c.-à-d. un rang à partir duquel le nombre d'étiquettes de sommets différentes n'augmente plus.

Théorème 4 (Existence d'un point fixe). Étant donnés un graphe $G = (V, E)$ et un entier d, $\exists k \in \mathbb{N}$ tel que $\forall (u, v) \in V^2, \forall i \geq 0, \alpha_{d,G}^k(u) = \alpha_{d,G}^k(v) \Rightarrow \alpha_{d,G}^{k+i}(u) = \alpha_{d,G}^{k+i}(v)$.

Démonstration. Le théorème 3 montre que quel que soit $d \geq 0$, chaque rang $i+1$ de la suite α est une fonction d'étiquetage au moins aussi forte que la fonction d'étiquetage au rang $i+1$. Le nombre d'étiquettes de sommets différentes ne peut donc qu'augmenter. En outre, comme le nombre de sommets est fini, il existe nécessairement un rang à partir duquel le nombre d'étiquettes ne peut plus augmenter. □

Le théorème 4 montre qu'il existe un rang α_d^k de la suite α_d à partir duquel la suite ne générera pas de fonction d'étiquetage strictement plus forte que α_d^k. Une fois ce point fixe atteint, calculer les rangs suivants de la suite est alors inutile.

Théorème 5 (Identification du point fixe). Étant donnés un graphe $G = (V, E)$, un entier d et un entier k, si $\forall (u, v) \in V^2, \alpha_{d,G}^k(u) = \alpha_{d,G}^k(v) \Rightarrow \alpha_{d,G}^{k+1}(u) = \alpha_{d,G}^{k+1}(v)$, alors $\forall i \geq k, \alpha_{d,G}^k(u) = \alpha_{d,G}^k(v) \Rightarrow \alpha_{d,G}^i(u) = \alpha_{d,G}^i(v)$.

Démonstration. Étant donnée sa définition, nous pouvons voir que la fonction β_d n'utilise pas les étiquettes α elles-mêmes mais uniquement un opérateur d'égalité sur ces étiquettes. Par conséquent, si un réétiquetage des sommets ne change pas les propriétés d'égalité entre les étiquettes de sommets, aucun réétiquetage suivant ne les changera. □

Le théorème 5 montre que, quand un rang de la séquence α_d^k n'augmente pas le nombre d'étiquettes de sommets différentes par rapport au rang précédent, le point fixe est atteint et il n'est pas nécessaire de prolonger le calcul de la suite.

Finalement, nous pouvons trivialement prouver que ce point fixe est atteint en au plus $|V|$ rangs car le nombre d'étiquettes de sommets est strictement croissant : soit le nombre d'étiquettes augmente soit le point fixe est atteint.

6.3.4 d-IDL-consistance et algorithme de filtrage associé

Nous proposons d'utiliser la suite α_d de fonctions de réétiquetage isomorphe-consistantes des sommets pour filtrer le domaine des variables d'un CSP associé à un GIP. La fonction d'étiquetage initiale α_d^0 est définie comme la fonction associant la même étiquette $\emptyset$ à chaque

sommet des graphes. En partant de cette fonction d'étiquetage initiale, la suite α_d^k est calculée jusqu'à l'obtention du point fixe. Le dernier étiquetage obtenu est alors utilisé pour définir une nouvelle consistance partielle pour la contrainte *gip*.

Définition. La contrainte globale $gip(V, E, V', E', L)$ correspondant au problème de l'isomorphisme entre les deux graphes $G = (V, E)$ et $G' = (V', E')$ est d-iterated distance label-consistante (d-IDL-consistante) si et seulement si :

$$\forall (x_u, u) \in L,\ \forall u' \in D(x_u), \forall k \in \mathbb{N} \quad , \quad \alpha^k_{d,G}(u) = \alpha^k_{d,G'}(u')$$

Afin de rendre la contrainte *gip* d-IDL-consistante, il suffit de calculer la suite α_d^1, α_d^2,... d'étiquetage de sommets pour chacun des graphes G et G' jusqu'à l'obtention du point fixe k et de retirer du domaine de chaque variable x_u associé au sommet $u \in V$ les valeurs $u' \in D(x_u)$ telles que $\alpha^k_{d,G}(u) \neq \alpha^k_{d,G'}(u')$.

Trois cas de figure peuvent alors se présenter :

1. le domaine d'une variable est vide, (quand il existe au moins un sommet dont l'étiquette n'est pas représentée dans l'autre graphe), le problème n'admet alors pas de solution et les graphes ne sont pas isomorphes ;
2. le domaine de chaque variable est réduit à un singleton et il est alors possible (en un temps linéaire) de vérifier si la seule affectation possible des valeurs aux variables représente une relation d'isomorphisme entre les deux graphes ;
3. le domaine de certaines variables contient plusieurs valeurs et il est nécessaire de dérouler un arbre de recherche.

Notons que le calcul de la suite peut parfois être interrompu avant l'obtention du point fixe. Chaque rang de la suite α_d peut être utilisé pour filtrer le domaine des variables et, si le cas de figure 1 ou 2 se présente, la consistance globale de la contrainte *gip* peut être aisément vérifiée.

Les étiquettes de sommets sont d'autant plus longues que le rang de la suite est élevé. La comparaison des étiquettes est donc d'autant plus coûteuse en temps et en espace que le rang de la suite est élevé. Néanmoins, comme la preuve du théorème 4 nous le montre, il est possible de renommer les étiquettes à chaque rang de la suite si ce renommage prend garde à conserver les propriétés d'égalité et de différence entre les étiquettes des sommets des deux graphes. À la fin de chaque rang de la suite d'étiquetage, les étiquettes sont renommées par des entiers uniques afin de garder le coût en espace et en temps du calcul constant à chaque rang de la suite.

Complexité. Afin d'établir la d-IDL-consistance, il faut connaître, pour chaque sommet v du graphe, la liste des sommets distants d'au plus d arcs de v. La complexité en temps de ce calcul est de $\mathcal{O}(d \times |E|)$ et sa complexité en espace est de $\mathcal{O}(|V|^2)$. Comme le diamètre d'un graphe est borné par son nombre de sommets, la complexité en temps maximale est bornée par $\mathcal{O}(d \times |E|)$. Chaque rang de la suite α_d est calculable en $\mathcal{O}(|V|^2)$. Le renommage des étiquettes se fait avec une complexité en temps de $\mathcal{O}(|V| \times log(|V|))$ (il faut trier les ensembles de triplets des étiquettes et il y a au plus $|V|$ triplets par ensemble). Il y a au plus $|V|$ rangs dans la suite. Par conséquent, la complexité théorique totale du filtrage par d-IDL-consistance est de $\mathcal{O}(|V|^3)$, quelle que soit la distance maximale d. Sa complexité en espace est de $\mathcal{O}(|V|^2)$.

6.3.5 Propagation des réductions

Établir la d-IDL-consistance ne permet pas toujours de réduire le domaine de toutes les variables à un singleton. Considérons par exemple le graphe de la figure 6.2. Ce graphe comporte plusieurs symétries (il est isomorphe à n'importe quel graphe obtenu par une permutation circulaire de ses sommets). Quel que soit la distance maximale d choisie, tous les sommets ont la

même étiquette. En effet, chaque sommet de ce graphe a exactement 1 sommet à distance 0, 2 sommets à distance 1, 2 sommets à distance 2 et 1 sommet à distance 3. Tous les sommets ont donc la même étiquette après une itération de la suite α_d et la d-IDL-consistance est atteinte sans filtrer le domaine d'aucune variable.

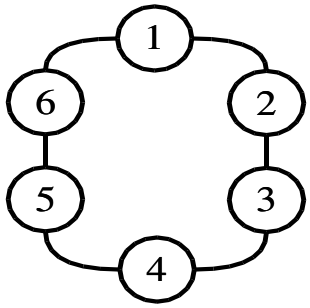


FIG. 6.2 – Un graphe $G = (V, E)$ circulaire. Pour tous les couples de sommets $(u, v) \in V^2$, quels que soient les entiers naturels k et d, $\alpha^k_{d,G}(u) = \alpha^k_{d,G}(v)$.

Quand le filtrage par IDL-consistance ne réduit par le domaine de chaque variable à un singleton, il est nécessaire d'explorer l'espace de recherche composé de toutes les affectations possibles en construisant un arbre de recherche. À chaque nœud de cet arbre, le domaine d'une variable est découpé en deux sous-domaines, puis des techniques de filtrage relatives à des consistances partielles (telles que l'arc-consistance) sont utilisées pour réduire le domaine des variables. Ces techniques de filtrage utilisent les contraintes afin de propager la réduction du domaine d'une variable sur les domaines des autres variables jusqu'à ce qu'un domaine devienne vide (le nœud peut alors être coupé), ou qu'un point fixe soit atteint (soit une solution est trouvée, soit un nouveau nœud doit être développé).

Pour utiliser une contrainte *gip* pour propager les réductions de domaines, une première possibilité consisterait à utiliser l'ensemble des contraintes C_{edge} défini en section 2. Cependant, il est également possible d'utiliser l'IDL-consistance pour propager plus efficacement les réductions de domaines. Nous proposons deux façons d'utiliser l'IDL-consistance pour propager ces réductions : une première utilisable quand le domaine d'une variable est réduit à un singleton et une autre utilisable pour toute réduction de domaine mais moins performante que la première.

Cas 1 : le domaine d'une variable est réduit à un singleton

Lors du développement d'un arbre de recherche, lorsque le domaine d'une variable est réduit à un singleton, le même algorithme de IDL-consistance peut être utilisé.

Affecter une valeur v à une variable x_u exprime le fait que le sommet u "joue le même rôle" dans le graphe G que le sommet v dans le graphe G'. Par conséquent, il est possible de modifier les étiquettes de sommets des graphes en attribuant la même étiquette unique aux sommets u et v (qui partageaient auparavant leur étiquette avec d'autres sommets). Le calcul de la suite α_d peut alors être relancé (jusqu'à l'obtention d'un nouveau point fixe) en prenant comme étiquetage initial cet étiquetage modifié.

Exemple. Considérons par exemple le graphe G de la figure 6.2 et définissons un autre graphe $G' = (V', E')$ isomorphe à G et tel que chaque sommet $u \in V$ est renommé en u' dans G'. Nous notons x_u la variable associée à un sommet $u \in V$. Aucune consistance ne peut filtrer

le domaine des variables de ce CSP car toute affectation d'une valeur à une variable appartient à au moins une affectation complète valide. Par conséquent, l'IDL-consistance associe à tous les sommets la même étiquette l_1.

Afin de trouver une solution à ce CSP, il est nécessaire de développer un arbre de recherche en choisissant arbitrairement une affectation d'une valeur à une variable. Supposons que la valeur $1'$ soit affecté à la variable x_1. Le calcul de l'IDL-consistance peut être relancé en associant la même étiquette unique l_2 aux sommets 1 et $1'$. Établir la IDL-consistance (par exemple la 3-IDL-consistance mais le résultat est le même pour n'importe qu'elle valeur de $d > 0$) permet de réduire le domaine de x_4 au singleton $\{4'\}$ car le sommet 4 est à distance de 3 d'un sommet ayant l'étiquette l_2 et que seul le sommet $4'$ du graphe G' vérifie cette propriété. De la même façon, le domaine des variables x_2 et x_6 est réduit à l'ensemble $\{2', 6'\}$ et celui des variables x_3 et x_5 à l'ensemble $\{3', 5'\}$. Un forward-checking sur les contraintes C_{edge} n'aurait pas pu filtrer autant le domaine des variables. Le même filtrage est obtenu par arc consistance mais le coût de ce filtrage est plus élevé. En effet, la consistance d'arcs avec AC2001 sur un CSP décrivant le problème de l'isomorphisme nécessite $\mathcal{O}(ed^2)$ opérations [BC93] où e est le nombre de contraintes, c.-à-d. $e = n(n-1)/2$, et d est la taille du plus grand domaine, c.-à-d. $d = n$. Établir la d-IDL-consistance est donc d'un ordre moins coûteux que la consistance d'arcs sur les contraintes binaires traditionnellement utilisées.

Cas 2 : autre cas de réduction

La propagation recalculant la séquence α_d n'est utilisable que dans le cas où le domaine d'une variable est réduit à un singleton. Par conséquent, si une valeur est retirée du domaine d'une variable sans que ce domaine ne soit réduit à un singleton, il n'est pas possible d'utiliser ce type de propagation et seule une consistance partielle "classique" (telle l'arc consistance) sur les contraintes C_{edge} peut être utilisée.

Néanmoins, dans [SS04b], nous montrons qu'il est possible de tirer partie des résultats obtenus lors du filtrage par d-IDL-consistance pour définir un ensemble de contraintes plus "fortes", c.-à-d. des contraintes définies par un nombre plus petit (ou égal) de couples de valeurs autorisées. La propagation de ces contraintes permet de réduire plus fortement les domaines des variables.

L'idée est de contraindre chaque paire (x_u, x_v) de variables associée à la paire (u, v) de sommets du premier graphe à prendre leurs valeurs parmi l'ensemble des paires de sommets (u', v') du second graphe telles que la distance entre u et v est égale à la distance entre u' et v' (ou est supérieure à d). Comme le prouve le théorème 1, une fonction bijective entre les sommets de deux graphes est une fonction d'isomorphisme si et seulement si cette fonction préserve les distances entre chaque paire de sommets des deux graphes. La contrainte globale $gip(V, E, V', E', L)$ est donc sémantiquement équivalente à l'ensemble des "*contraintes de distance*" défini par : pour tout $((x_u, u), (x_v, v)) \in L \times L$ tel que $u \neq v$,

$$\begin{aligned} C_{distance,d}(x_u, x_v) &= \{(u', v') \in V' \times V' \mid \delta_{(V,E)}(u, v) = \delta_{(V',E')}(u', v')\} \text{ si } \delta(V, E)(u, v) \leq d \\ C_{distance,d}(x_u, x_v) &= \{(u', v') \in V' \times V' \mid \delta_{(V',E')}(u', v') > d\} \text{ si } \delta(V, E)(u, v) > d \end{aligned}$$

Nous pouvons facilement montrer que chaque contrainte binaire $C_{distance,d}(x_u, x_v)$ est au moins aussi forte que la contrainte binaire $C_{edge}(x_u, x_v)$ correspondante quand $d \geq 1$, c.-à-d. qu'elle est vérifiée pour moins de tuples :

- si les sommets de G associés aux variables x_u et x_v sont reliés par une arête, alors $C_{distance,d}(x_u, x_v) = C_{edge}(x_u, x_v) = E'$;
- sinon, $C_{distance,d}(x_u, x_v) \subseteq C_{edge}(x_u, x_v)$ car $C_{edge}(x_u, x_v)$ contient tous les couples de sommets de G' qui ne sont pas connectés par un arc alors que $C_{distance,d}(x_u, x_v)$ ne contient

que les couples de sommets de G' tels que la distance entre leurs sommets est égale à la distance entre les sommets de G associés à x_u et x_v (si cette distance est plus petite ou égale à d) ou les couples de sommets de G' ayant une distance supérieure à d (si la distance entre u et v est plus grande que d).

Par conséquent, quelle que soit la consistance locale considérée, propager une contrainte $C_{distance,d}$ permettra de réduire au moins autant les domaines que la propagation de la contrainte C_{edge} correspondante, et, dans certains cas, réduira plus fortement les domaines.

Exemple. Considérons par exemple le graphe G de la figure 6.2 et définissons un autre graphe $G' = (V', E')$ isomorphe à G et tel que chaque sommet $u \in V$ est renommé en u' dans G'. Nous notons x_u la variable associée à un sommet $u \in V$. La contrainte d'arc entre x_1 et x_4 contient tous les couples de sommets de G' qui ne sont pas reliés par un arc, c.-à-d. :

$$\begin{aligned} C_{edge}(x_1, x_4) = \{ \quad & (1',3'), (1',4'), (1',5'), (2',4'), (2',5'), (2',6'), \\ & (3',5'), (3',6'), (3',1'), (4',6'), (4',1'), (4',2'), \\ & (5',1'), (5',2'), (5',3'), (6',2'), (6',3'), (6',4')\} \end{aligned}$$

alors que la contrainte de distance $C_{distance,+\infty}$ entre x_1 et x_4 contient seulement les couples de sommets de G' qui sont à une distance de 3 l'un de l'autre car la distance entre les sommets 1 et 4 est égale à 3, c.-à-d. :

$$C_{distance}(x_1, x_4) = \{(1',4'), (2',5'), (3',6'), (4',1'), (5',2'), (6',3')\}$$

La contrainte de distance entre x_1 et x_4 étant plus stricte que la contrainte d'arc correspondante, elle permet une meilleure propagation des réductions de domaines. Par exemple, si la valeur $1'$ est affectée à x_1, une propagation par forward-checking de la contrainte $C_{distance,+\infty}(x_1, x_4)$ réduit le domaine de x_4 au singleton $\{4'\}$ alors que la propagation par forward-checking de $C_{edge}(x_1, x_4)$ ne réduit le domaine de x_4 qu'à $\{3', 4', 5'\}$, l'ensemble des sommets qui ne sont pas connectés à $1'$.

De même, lorsque la valeur $1'$ est supprimée du domaine de x_1, une propagation par consistance d'arc de la contrainte $C_{distance,+\infty}(x_1, x_4)$ permet d'enlever la valeur $4'$ au domaine de x_4 alors que la même propagation de la contrainte $C_{edge}(x_1, x_4)$ ne permet d'enlever aucune valeur.

6.4 Label-consistance et ILL-consistance

Dans [SS04b] et [SS06], nous proposons deux consistances partielles pour la contrainte globale *gip* : la label-consistance et l'iterated local-label-consistance (ILL-consistance). Nous montrons dans cette section que chacune de ces deux consistances partielles est un cas particulier de l'IDL-consistance. La label-consistance correspond à la $+\infty$-IDL-consistance pour laquelle le calcul de la suite α est interrompue au rang 2 et l'ILL-consistance correspond à la 1-IDL-consistance. Pour chacune de ces consistances, nous donnons un exemple complet sur le graphe de la figure 6.1.

6.4.1 Label-consistance

Définition et exemple

Dans [SS04b], nous proposons une consistance partielle pour la contrainte *gip* : la label-consistance. Cette consistance est basée sur le calcul, pour chaque sommet u des graphes, d'une étiquette qui caractérise les relations en termes de plus courte chaîne entre u et tous les autres sommets du graphe.

De façon plus formelle, étant donnés un graphe $G = (V, E)$, un sommet $u \in V$ et une distance $i \in [0, |V| - 1]$, nous notons $\Delta_G(u, i)$ l'ensemble des sommets à une distance de i du sommet u et $\#\Delta_G(u, i)$ le nombre de ces sommets, c.-à-d. :

$$\Delta_G(u, i) = \{v \in V | \delta_G(u, v) = i\} \quad \text{et} \quad \#\Delta_G(u, i) = |\Delta_G(u, i)|$$

Par exemple, pour le graphe G de la figure 6.1, nous avons :

$$\begin{array}{ll} \Delta_G(A, 0) = \{A\} & \#\Delta_G(A, 0) = 1 \\ \Delta_G(A, 1) = \{B, C, D, F\} & \#\Delta_G(A, 1) = 4 \\ \Delta_G(A, 2) = \{E, G, H, J\} & \#\Delta_G(A, 2) = 4 \\ \Delta_G(A, 3) = \{I\} & \#\Delta_G(A, 3) = 1 \\ \Delta_G(A, i) = \emptyset & \#\Delta_G(A, i) = 0, \quad \forall i \geq 4 \end{array}$$

Étant donnés un graphe $G = (V, E)$ et un sommet $u \in V$, nous notons $\#\Delta_G(u)$ la séquence composée de $|V|$ nombres correspondant respectivement aux nombres de sommets à une distance de 0, 1, ... ,$|V| - 1$ de u, c.-à-d. :

$$\#\Delta_G(u) = < \#\Delta_G(u, 0), \#\Delta_G(u, 1), ..., \#\Delta_G(u, |V| - 1) >$$

Nous omettrons les zéros présents à la fin de la séquence (un zéro ne peut être suivi que d'autres zéros).
Par exemple, les séquences des sommets du graphe G de la figure 6.1 sont :
$\#\Delta_G(A) = \#\Delta_G(D) = \#\Delta_G(F) = < 1, 4, 4, 1 >$
$\#\Delta_G(B) = \#\Delta_G(C) = \#\Delta_G(E) = \#\Delta_G(G) = \#\Delta_G(I) = \ < 1, 3, 4, 2 >$
$\#\Delta_G(H) = < 1, 4, 5 >$
$\#\Delta_G(J) = < 1, 3, 3, 3 >$

Chaque séquence $\#\Delta_G(u)$ caractérise les relations en termes de distance entre le sommet u et les autres sommets de G. L'étiquetage des sommets par leur séquence $\#\Delta_G(u)$ est donc isomorphe consistant. Plusieurs sommets du même graphe peuvent cependant avoir la même séquence, ce critère n'est donc pas toujours suffisant pour élaguer efficacement l'espace de recherche. Par exemple, dans le graphe de la figure 6.1, cinq sommets ont pour séquence $< 1, 3, 4, 2 >$. À partir de ces séquences $\#\Delta_G$, une second étiquetage des sommets est donc défini : l'étiquetage *label*.

Soit un graphe $G = (V, E)$, nous notons $\#\Delta_G$ l'ensemble de toutes les séquences associées aux sommets de G :

$$\#\Delta_G = \{s | \exists u \in V, s = \#\Delta_G(u)\}$$

Par exemple, l'ensemble de toutes les séquences présentes dans le graphe G de la figure 6.1 est :

$$\#\Delta_G = \{< 1, 3, 3, 3 >, < 1, 3, 4, 2 >, < 1, 4, 4, 1 >, < 1, 4, 5 >\}$$

Soit un sommet $u \in V$, nous notons $label_G(u)$ l'ensemble de tous les triplets (i, s, k) tels que i est une distance, s est une séquence, et k est le nombre de sommets distants de i du sommet u et dont la séquence est s :

$$\begin{array}{rl} label_G(u) = \{(i, s, k) \mid & i \in [0, |V| - 1], \\ & s \in \#\Delta_G, \text{ and} \\ & k = |\{v \in \Delta_G(u, i) / \#\Delta_G(v) = s\}|\} \end{array}$$

Nous ne considérerons que les triplets (i, s, k) tels que $k > 0$.
Par exemple, pour le graphe G de la figure 1, nous avons :

$$\begin{aligned} label_G(A) = \{ \quad & (0, < 1,4,4,1 >, 1), \\ & (1, < 1,3,4,2 >, 2), (1, < 1,4,4,1 >, 2), \\ & (2, < 1,3,3,3 >, 1), (2, < 1,3,4,2 >, 2), (2, < 1,4,5 >, 1), \\ & (3, < 1,3,4,2 >, 1)\} \end{aligned}$$

car il y a un sommet (A) distant de 0 de A et dont la séquence est $< 1,4,4,1 >$, deux sommets (B et C) distants de 1 de A dont la séquence est $< 1,3,4,2 >$, deux autres sommets (D et F) distants eux aussi de 1 du sommet A mais dont la séquence est $< 1,4,4,1 >$, etc...

Dans [SS04b], il est montré que l'étiquetage des sommets par leur label est isomorphe-consistant et peut donc être utilisé pour filtrer le domaine des variables d'un CSP correspondant à un GIP.

Théorème 6 (L'étiquetage par les labels est isomorphe-consistant). Soient deux graphes $G = (V, E)$ et $G' = (V', E')$. S'il existe une fonction d'isomorphisme $f : V \to V'$ entre les deux graphes G et G', alors, pour chaque sommet $u \in V$, $label_G(u) = label_{G'}(f(u))$.

Démonstration. f est une bijection et la distance entre deux sommets u et v de G est égale à la distance entre leur images par f (voir le théorème 1), c.-à-d. que $\delta_G(u, v) = \delta_{G'}(f(u), f(v))$. Par conséquent, le nombre de sommets de G distants de i du sommet u est égal au nombre de sommets de G' distants de i du sommet $f(u)$, donc $\#\Delta_G(u) = \#\Delta_{G'}(f(u))$. Par conséquent, les ensembles $\#\Delta$ de séquences des deux graphes sont égaux, c.-à-d. que $\#\Delta_G = \#\Delta_{G'}$. Pour chaque séquence $s \in \#\Delta_G$, et pour chaque sommet $u \in V$, le nombre de sommets distants de i du sommet u et dont la séquence est s est égal au nombre de sommets distants de i du sommet $f(u)$ ayant également pour séquence s. Nous avons donc $label_G(u) = label_{G'}(f(u))$. □

Le filtrage par labels réduit souvent très fortement les domaines des variables. Considérons par exemple le graphe G de la figure 6.1. Les trois premiers triplets des labels de chaque sommet (triés par distance, puis par séquence croissante) sont :

$$\begin{aligned} label_G(A) = \quad & \{(0, < 1,4,4,1 >, 1), (1, < 1,3,4,2 >, 2), (1, < 1,4,4,1 >, 2), ...\} \\ label_G(B) = \quad & \{(0, < 1,3,4,2 >, 1), (1, < 1,3,3,3 >, 1), (1, < 1,4,4,1 >, 2), ...\} \\ label_G(C) = \quad & \{(0, < 1,3,4,2 >, 1), (1, < 1,3,4,2 >, 1), (1, < 1,4,4,1 >, 2), ...\} \\ label_G(D) = \quad & \{(0, < 1,4,4,1 >, 1), (1, < 1,3,4,2 >, 2), (1, < 1,4,4,1 >, 1), ...\} \\ label_G(E) = \quad & \{(0, < 1,3,4,2 >, 1), (1, < 1,3,4,2 >, 2), (1, < 1,4,5 >, 1), ...\} \\ label_G(F) = \quad & \{(0, < 1,4,4,1 >, 1), (1, < 1,3,3,3 >, 1), (1, < 1,3,4,2 >, 1), ...\} \\ label_G(G) = \quad & \{(0, < 1,3,4,2 >, 1), (1, < 1,3,4,2 >, 2), (1, < 1,4,4,1 >, 1), ...\} \\ label_G(H) = \quad & \{(0, < 1,4,5 >, 1), (1, < 1,3,4,2 >, 2), (1, < 1,4,4,1 >, 2), ...\} \\ label_G(I) = \quad & \{(0, < 1,3,4,2 >, 1), (1, < 1,3,3,3 >, 1), (1, < 1,3,4,2 >, 1), ...\} \\ label_G(J) = \quad & \{(0, < 1,3,3,3 >, 1), (1, < 1,3,4,2 >, 2), (1, < 1,4,4,1 >, 1), ...\} \end{aligned}$$

Tous les sommets de G ont des labels différents. Par conséquent, pour toute contrainte *gip* entre G et un autre graphe G', le filtrage par labels détectera une inconsistance (si des labels de G ne sont pas présents dans G') ou réduira le domaine de chaque variable à un singleton rendant alors la consistance globale du problème facile à vérifier.

Lien avec l'IDL-consistance

La label-consistance est un cas particulier de la $+\infty$-IDL-consistance.

En effet, les séquences $\#\Delta_G(u)$ contiennent exactement la même information que la fonction $\alpha^1_{+\infty}$: l'ensemble des triplets (j, k, l) renvoyés par la fonction $\alpha^1_{+\infty}$ a simplement été remplacé par

une séquence de nombre de sommets ordonnés par la distance. Étant donné que les sommets ne sont initialement pas étiquetés, cette transformation se fait sans perte d'information. En outre, les labels sont identiques aux étiquettes renvoyées par la fonction $\alpha^2_{+\infty}$ si $\alpha^2_{+\infty} = \#\Delta_G$

Par conséquent, la label-consistance est identique à la $+\infty$-IDL-consistance pour laquelle la suite α est interrompue au rang 2.

6.4.2 ILL-consistance

Définition et lien avec l'IDL-consistance

La label-consistance permet en général de réduire le domaine des variables à des singletons et permet donc de résoudre le problème sans développer un arbre de recherche. Néanmoins, ce filtrage nécessite un temps de calcul assez élevé : il nécessite de calculer la distance entre chaque couple de sommets et ce calcul est généralement coûteux en pratique. Parallèlement à ce temps de calcul élevé, nous avons pu observer que les labels des sommets étaient le plus souvent très différents les uns des autres et que par conséquent, il n'était généralement pas nécessaire de prendre en compte l'intégralité de l'information contenue dans ces labels pour montrer que deux sommets ne peuvent être associés par une fonction d'isomorphisme de graphe.

Dans [SSJ06a] nous avons proposé une autre consistance partielle pour la contrainte globale *gip* : l'iterated-local-label-consistance (ILL-consistance). Cette consistance permet de filtrer le domaine des variables du CSP associé à un GIP sans devoir calculer la matrice des distances entre tous les couples de sommets des graphes. Le filtrage est basé sur les propriétés d'un sommet et des sommets qui lui sont adjacents. Ce filtrage est inefficace sur les graphes réguliers mais permet un filtrage aussi efficace (sinon meilleur) que la label-consistance sur des graphes quelconques tout en étant beaucoup plus rapide.

L'ILL-consistance correspond exactement à la 1-IDL-consistance. Autrement dit, seuls les voisins directs d'un sommet sont pris en compte pour le calcul d'un rang de la suite α. La propagation de l'étiquetage d'un sommet u vers les sommets qui ne lui sont pas adjacents se fait au fur et à mesure du calcul des différents rangs de la suite α.

Exemple complet

Nous proposons un exemple complet de la séquence α de réétiquetage par ILL-consistance (ou autrement dit de 1-IDL-consistance) sur le graphe de la figure 6.1. À chaque rang de la suite α, les étiquettes sont renommées par des étiquettes $l_{i,j}$. Pour des raisons de place, nous noterons $\alpha^k_{d,G}(S)$ quand $\forall (u,v) \in S^2, \alpha^k_{d,G}(u) = \alpha^k_{d,G}(v)$.

Au rang 1 de la suite :

$$\begin{array}{lcll} \alpha^1_{1,G}(\{A,D,F,H\}) & = & \{(0,1,\emptyset),(1,4,\emptyset)\} & \Rightarrow \quad l_{1,1} \\ \alpha^1_{1,G}(\{B,C,E,G,I,J\}) & = & \{(0,1,\emptyset),(1,3,\emptyset)\} & \Rightarrow \quad l_{1,2} \end{array}$$

Par conséquent, pour les deux rangs suivants :

$$
\begin{array}{lcrcl}
\alpha^2_{1,G}(\{A,D,F,H\}) & = & \{(0,1,l_{1,1}),(1,2,l_{1,1}),(1,2,l_{1,2})\} & \Rightarrow & l_{2,1} \\
\alpha^2_{1,G}(\{B,C\}) & = & \{(0,1,l_{1,2}),(1,1,l_{1,2}),(1,2,l_{1,1})\} & \Rightarrow & l_{2,2} \\
\alpha^2_{G}(\{E,G,I,J\}) & = & \{(0,1,l_{1,2}),(1,1,l_{1,1}),(1,2,l_{1,2})\} & \Rightarrow & l_{2,3}
\end{array}
$$

$$
\begin{array}{lcrcl}
\alpha^3_{1,G}(A) & = & \{(0,1,l_{2,1}),(1,2,l_{2,1}),(1,2,l_{2,2})\} & \Rightarrow & l_{3,1} \\
\alpha^3_{1,G}(\{B,C\}) & = & \{(0,1,l_{2,2}),(1,2,l_{2,1}),(1,1,l_{2,3})\} & \Rightarrow & l_{3,2} \\
\alpha^3_{1,G}(\{D,F\}) & = & \{(0,1,l_{2,1}),(1,1,l_{2,2}),(1,1,l_{2,3}),(1,2,l_{2,1})\} & \Rightarrow & l_{3,3} \\
\alpha^3_{1,G}(\{E,J\}) & = & \{(0,1,l_{2,3}),(1,1,l_{2,1}),(1,1,l_{2,2}),(1,1,l_{2,3})\} & \Rightarrow & l_{3,4} \\
\alpha^3_{1,G}(\{G,I\}) & = & \{(0,1,l_{2,3}),(1,1,l_{2,1}),(1,2,l_{2,3})\} & \Rightarrow & l_{3,5} \\
\alpha^3_{1,G}(H) & = & \{(0,1,l_{2,1}),(1,2,l_{2,1}),(1,2,l_{2,3})\} & \Rightarrow & l_{3,6}
\end{array}
$$

Notons qu'à partir du rang 3, deux sommets (les sommets A et H) ont des étiquettes uniques. Par conséquent, il n'est plus nécessaire de calculer les rangs suivants de la suite α pour ces sommets. Le sommet A (resp. H) garde l'étiquette $l_{3,1}$ (resp. $l_{3,6}$).

$$
\begin{array}{lcrcl}
\alpha^4_{1,G}(\{B,C\}) & = & \{(0,1,l_{3,2}),(1,1,l_{3,1}),(1,1,l_{3,3}),(1,1,l_{3,4})\} & \Rightarrow & l_{4,1} \\
\alpha^4_{1,G}(D) & = & \{(0,1,l_{3,3}),(1,1,l_{3,1}),(1,1,l_{3,2}),(1,1,l_{3,5}),(1,1,l_{3,6})\} & \Rightarrow & l_{4,2} \\
\alpha^4_{1,G}(E) & = & \{(0,1,l_{3,4}),(1,1,l_{3,2}),(1,1,l_{3,5}),(1,1,l_{3,6})\} & \Rightarrow & l_{4,3} \\
\alpha^4_{1,G}(F) & = & \{(0,1,l_{3,3}),(1,1,l_{3,1}),(1,1,l_{3,2}),(1,1,l_{3,4}),(1,1,l_{3,6})\} & \Rightarrow & l_{4,4} \\
\alpha^4_{1,G}(G) & = & \{(0,1,l_{3,5}),(1,1,l_{3,3}),(1,1,l_{3,4}),(1,1,l_{3,5})\} & \Rightarrow & l_{4,5} \\
\alpha^4_{1,G}(I) & = & \{(0,1,l_{3,5}),(1,1,l_{3,4}),(1,1,l_{3,5}),(1,1,l_{3,6})\} & \Rightarrow & l_{4,6} \\
\alpha^4_{1,G}(J) & = & \{(0,1,l_{3,4}),(1,1,l_{3,2}),(1,1,l_{3,3}),(1,1,l_{3,5})\} & \Rightarrow & l_{4,7}
\end{array}
$$

Au rang 4, seulement deux sommets (B et C) ont la même étiquette.

$$
\begin{array}{lcrcl}
\alpha^5_{1,G}(B) & = & \{(0,1,l_{4,1}),(1,1,l_{3,1}),(1,1,l_{4,4}),(1,1,l_{4,7})\} & \Rightarrow & l_{5,1} \\
\alpha^5_{1,G}(C) & = & \{(0,1,l_{4,1}),(1,1,l_{3,1}),(1,1,l_{4,2}),(1,1,l_{4,3})\} & \Rightarrow & l_{5,2}
\end{array}
$$

À partir du rang 5, tous les sommets ont des étiquettes différentes. Par conséquent, n'importe quel problème d'isomorphisme de graphe impliquant le graphe G de la figure 6.1 est résolu par le filtrage par 1-IDL-consistance.

6.5 Extension aux graphes étiquetés, aux graphes orientés

6.5.1 Graphes étiquetés

Dans de nombreuses applications, les sommets et les arcs des graphes à comparer sont étiquetés et la fonction d'isomorphisme f recherchée doit respecter ces étiquettes.

De façon plus formelle, étant donnés deux ensembles L_V et L_E d'étiquettes de sommets et d'arcs, un graphe étiqueté est un tuple $G = (V, E, \alpha, \beta)$ où (V, E) est un graphe, $\alpha : V \rightarrow L_V$ une application attribuant une étiquette à chaque sommet du graphe et $\beta : E \rightarrow L_E$ une application attribuant une étiquette à chaque arc du graphe.

Étant donnés deux graphes étiquetés $G = (V, E, \alpha, \beta)$ et $G' = (V', E', \alpha', \beta')$ tels que $|V| = |V'|$, les graphes étiquetés G et G' sont isomorphes si et seulement s'il existe une application bijective $f : V \rightarrow V'$ telle que $(u, v) \in E \Leftrightarrow (f(u), f(v)) \in E'$ et $\forall u \in V, \alpha(u) = \alpha'(f(u))$ et $\forall (u, v) \in E, \beta((u, v)) = \beta'((f(u), f(v)))$.

Étant donnés deux graphes étiquetés $G = (V, E, \alpha, \beta)$ et $G' = (V', E', \alpha', \beta')$, le CSP (X, D, C) correspondant au problème de l'isomorphisme de G et de G' est le suivant :

- une variable x_u est associée à chaque sommet $u \in V$, c.-à-d. que $X = \{x_u / u \in V\}$;
- le domaine de chaque variable x_u est l'ensemble des sommets de G' ayant le même nombre d'arcs entrants, le même nombre d'arcs sortants et la même étiquette que u :

$$D(x_u) = \{u' \in V' \mid \begin{array}{l} |\{(u,v) \in E\}| = |\{(u',v') \in E'\}| \text{ et} \\ |\{(v,u) \in E\}| = |\{(v',u') \in E'\}| \text{ et} \\ \alpha(u) = \alpha(u')\} \end{array}$$

- il existe une contrainte binaire $C_{edge}(x_u, x_v)$ entre chaque paire de variables (x_u, x_v) exprimant le fait que les sommets de G' affectés à x_u et x_v doivent être connectés par un arc si et seulement si $(u,v) \in E$:

$$\begin{array}{ll} \text{si } (u,v) \in E, & C_{edge}(x_u, x_v) = \{(u',v') \in E' | \beta(u,v) = \beta(u',v')\} \\ \text{sinon} & C_{edge}(x_u, x_v) = \{(u',v') \in V'^2 \mid u' \neq v' \text{ et } (u',v') \notin E'\} \end{array}$$

Le filtrage que nous proposons est parfaitement applicable à des graphes étiquetés. Néanmoins, il n'utilise pas l'information portée par les étiquettes des sommets et des arcs et l'utilisation de cette information permettrait de mieux filtrer le domaine des variables.

Prise en compte des étiquettes de sommets. Prendre en compte les étiquettes de sommets dans notre travail est simple. Il suffit pour cela de définir le premier étiquetage α^0 de la suite α par les étiquettes de sommets des graphes.

Prise en comptes des étiquettes d'arcs. Il est également possible de prendre en compte les étiquettes d'arcs. Nous avons, dans le cas des graphes non-étiquetés, pris en compte le fait que toute chaîne dans un graphe devait exister dans l'autre graphe. Il en est de même pour les graphes étiquetés où les chaînes doivent en plus avoir la même succession d'étiquettes d'arc. Par conséquent, en définissant un ordre total sur les chaînes prenant en compte d'une part la longueur des chaînes et d'autre part les étiquettes (en définissant un ordre total sur les étiquettes et en utilisant alors l'ordre lexicographique pour les chaînes) il est possible d'adapter la fonction α^{i+1} afin qu'elle retourne des triplets $(< l_1, l_2, ... >, k, l)$ où k est le nombre de sommets étiquetés par l et accessibles par une plus courte chaîne étiquetée par $< l_1, l_2, ... >$.

6.5.2 Graphes orientés

Les définitions et les théorèmes introduits en section 3 restent valables dans le cas des graphes orientés. Cependant, contrairement aux graphes non-orientés, deux sommets d'un graphe orienté ne sont pas nécessairement connectés par un chemin. Sans aucune modification de notre algorithme de filtrage, le réétiquetage d'un sommet n'est donc pas propagé sur tous les sommets du graphe. Par exemple, deux sommets qui n'ont que des arcs entrants ont nécessairement la même étiquette et ce, quelle que soit leur relation avec les autres sommets du graphe. Dans le cas général, le filtrage par d-IDL-consistance ne réduit donc que faiblement le domaine des variables.

Une façon d'étendre notre travail aux graphes orientés consiste à générer le graphe non-orienté correspondant (en ignorant le sens des arcs) et à calculer les étiquettes α des sommets de ce graphe non-orienté et de transposer les résultats sur les sommets du graphe orienté. Bien que le filtrage obtenu puisse être efficace, il apparaît dommage de ne pas tenir compte du sens des arcs dans le filtrage.

Une seconde façon d'étendre notre travail consiste à adapter la fonction β afin de prendre en compte le sens des arcs. Par exemple, dans le cas de la 1-IDL-consistance, il est possible d'adapter

la fonction β afin qu'elle distingue d'une part les sommets v connectés à un sommet u par un arc (u, v) et d'autre part ceux connectés par un arc (v, u).

6.6 Résultats expérimentaux

Nous testons dans cette section l'efficacité de nos consistances et le temps d'exécutions des algorithmes permettant d'établir ces consistances sur des graphes non-étiquetés générés aléatoirement et nous comparons ces résultats à *Nauty*, le meilleur algorithme connu pour le problème de l'isomorphisme de deux graphes. Le protocole expérimental est le suivant : sur chaque graphe considéré, nous utilisons *Nauty* pour établir sa représentation canonique, nous calculons et dénombrons les labels de la label-consistance des sommets de ce graphe et nous établissons la 1-IDL-consistance. Nous ne cherchons donc pas à résoudre un problème d'isomorphisme de graphes en particulier mais nous cherchons à montrer que tout problème d'isomorphisme de graphes impliquant le graphe considéré peut être résolu avec notre filtrage.

6.6.1 Jeu de tests 1

Nous considérons les graphes générés aléatoirement issus du benchmark de Foggia *et al.* [FSV01]. Comme le nombre de sommets des graphes de ce benchmark ne dépasse pas 1000, nous avons généré des graphes plus gros avec Nauty (Nauty permet également la génération de graphes). Nous considérons au final des graphes ayant un nombre de sommets entre 200 et 9800 avec un pas de 200 sommets. Nous testons nos algorithmes avec des graphes ayant 3 densités d'arcs différentes : 1%, 5% et 10%. Afin de mesurer l'influence de la densité des graphes sur le temps d'exécution, nous avons également générés des graphes de 1000 sommets ayant une densité entre 1% et 50% (au delà, il est plus intéressant de prendre le graphe complémentaire).

Aucun des graphes $G = (V, E)$ considéré n'est automorphe : il n'existe pas de fonction d'isomorphisme entre G et lui-même autre que la fonction identité. Par conséquent, un filtrage parfait de ces graphes est tel que le domaine de chacune des variables est réduit à un singleton, autrement dit, que chaque sommet ait une étiquette différente. Nous comparons donc nos algorithmes de deux façons : le nombre d'étiquettes différentes que notre méthode a permis d'obtenir et le temps nécessaire à l'étiquetage des sommets.

Pour chaque type de graphe, les résultats donnés ont été obtenus en faisant une moyenne sur 100 graphes.

Puissance de filtrage

Nauty est un algorithme complet. Par conséquent, il trouve à coup sûr une représentation canonique des graphes et nous ne testons donc pas sa puissance de filtrage.

A contrario, nos algorithmes de filtrage définissent une consistance partielle et ne trouvent donc pas systématiquement un étiquetage unique de chaque sommet. Néanmoins, à l'exception des graphes de petite taille et de densité faible (*i.e.,* moins de 400 sommets et une densité de 1%), la label-consistance et la 1-IDL-consistance permettent toutes les deux un étiquetage unique des sommets. Pour les graphes de plus de 800 sommets, le point fixe de la 1-IDL-consistance est atteint au rang 2 de la suite α. Pour tous les graphes de taille supérieure à 600 sommets, nos deux consistances trouvent un étiquetage unique des sommets. Par conséquent, elles permettent la résolution du problème de l'isomorphisme de graphes sans déroulement d'un arbre de recherche.

Les résultats des petits graphes à faible densité sont présentés au tableau 6.1. Ces résultats montrent que la 1-IDL-consistance filtre mieux et beaucoup plus rapidement que la

$\|V\|$	1-IDL-consistance			label-consistance	
	#L	t	#iter	#L	t
200	199,64	0	3,40	191,92	0,01
400	400,00	0	2,88	399,87	0,07
600	600,00	0	2,14	600,00	0,19
800	800,00	0,01	2,01	800,00	0,36

TAB. 6.1 – Résultats obtenus par nos filtrages pour les graphes de densité 1%. Pour chaque type de graphe $G = (V, E)$ sont donnés le nombre de sommets $|V|$. Pour chaque consistance sont donnés le nombre moyen d'étiquettes de sommets #L obtenues et le temps nécessaire t du calcul. Le nombre moyen #iter de rangs nécessaires pour établir la 1-IDL-consistance est également donné.

label-consistance. Notons que la label-consistance n'est pas totalement identique à la $+\infty$-IDL-consistance car elle n'effectue le calcul que des deux premiers rangs de la suite α. Par conséquent, elle obtient de moins bons résultats que la $+\infty$-IDL-consistance théoriquement plus forte que la 1-IDL-consistance.

Temps d'exécution

Pour les graphes où le filtrage est parfait, nous comparons les temps d'exécutions de nos algorithmes avec ceux obtenus par *Nauty*. Ces mesures de temps d'exécutions ont été réalisées avec un PC à 1,7GHz et 512Mo de RAM sous linux (noyau 2.6).

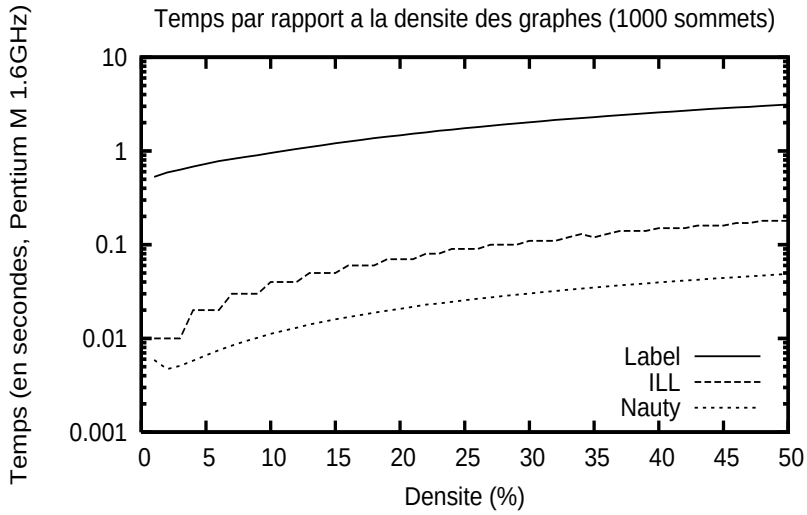


FIG. 6.3 – Évolution du temps d'exécution en fonction de la densité des graphes (1000 sommets)

Les courbes des figure 6.3, 6.4, 6.5 et 6.6 montrent que pour chacun des algorithmes, le temps d'exécution est d'autant plus important que la densité des graphes est élevée. Toutefois,

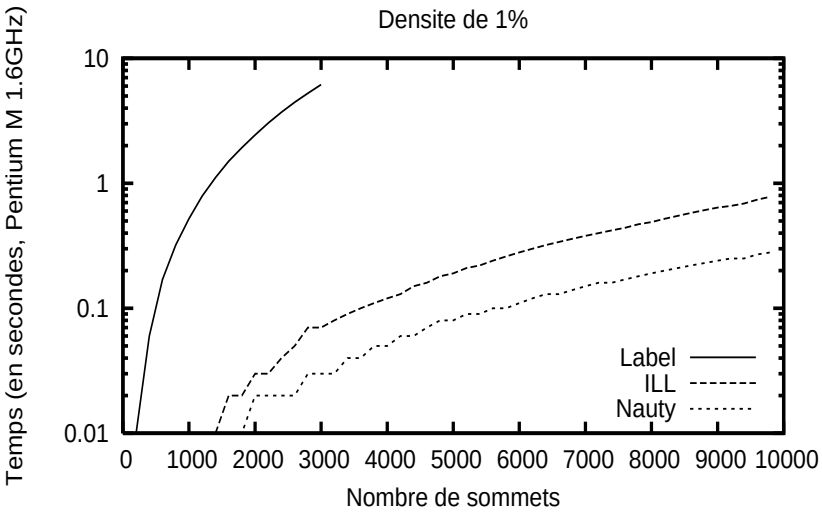


FIG. 6.4 – Évolution du temps d'exécution en fonction de la taille des graphes pour les graphes de densité 1%.

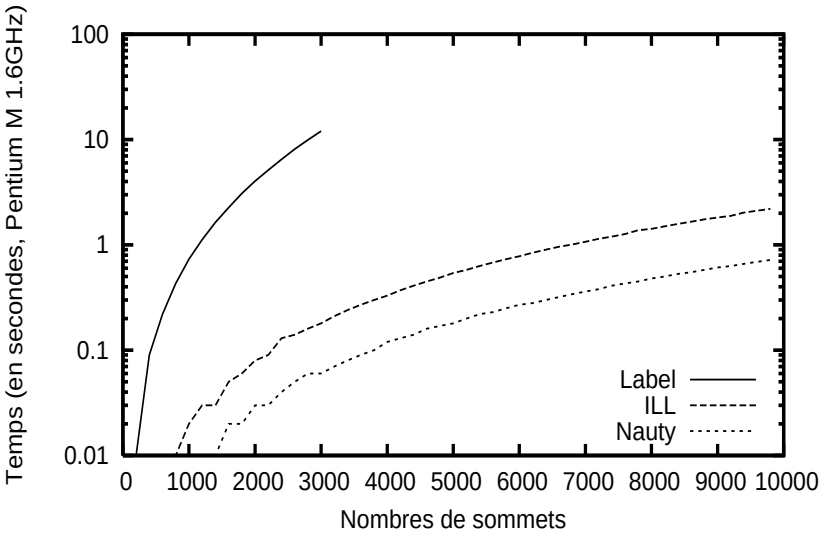


FIG. 6.5 – Évolution du temps d'exécution en fonction de la taille des graphes pour les graphes de densité 5%.

les comportements des trois algorithmes considérés sont similaires quelle que soit la densité des graphes.

D'autre part, les courbes montrent que la label-consistance est très nettement plus coûteuse que les deux autres méthodes (nous avons interrompu les courbes à partir de 3000 sommets).

Si nous comparons la 1-IDC consistance à *Nauty*, nous pouvons voir que ces deux algorithmes ont un comportement très similaire. Néanmoins, *Nauty* est en moyenne 3 fois plus rapide que la 1-IDC consistance et reste donc le meilleur algorithme connu pour le problème de l'isomorphisme de graphe. Notons finalement que nous avons dû interrompre les expérimentations avec des graphes de 9800 sommets car au delà, l'algorithme qui établit la 1-IDC consistance commençait à swapper (avec 512Mo de RAM).

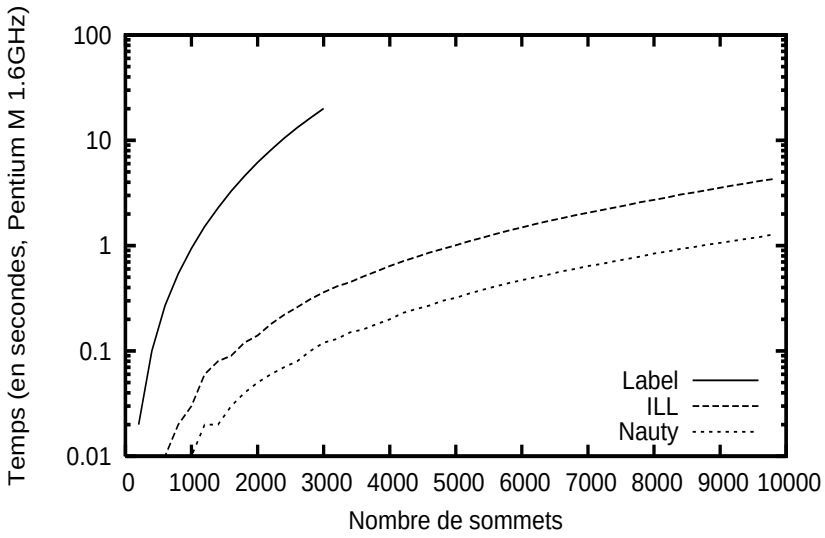


FIG. 6.6 – Évolution du temps d'exécution en fonction de la taille des graphes pour les graphes de densité 10%.

6.6.2 Jeu de tests 2

Nauty permet de générer tous les graphes non-isomorphes d'une certaine taille. Nous l'avons utilisé pour générer tous les graphes de 7 sommets. Il existe 853 graphes de 7 sommets non-isomorphes. Nous avons testé nos deux consistances sur les 363378 paires de graphes différentes résultantes.

Les graphes n'étant pas isomorphes, nos filtrages permettent de résoudre les instances lorsque les étiquettes des 7 sommets des deux graphes ne correspondent pas, c.-à-d. quand une étiquette n'est présente que dans un seul graphe ou n'apparaît pas le même nombre de fois dans chacun des deux graphes.

	Label-consistance	**1-IDL-consistance**	**$+\infty$-IDL-consistance**
Résolues	363337	363361	363366
Non résolues	41	17	12

Sur les 363378 instances différentes, la label-consistance (pour laquelle le calcul de la suite α est interrompu au rang 2) permet de résoudre toutes les instances à l'exception de 41 problèmes. L'ILL-consistance permet de résoudre 363361 instances et échoue sur seulement 17 instances. Finalement, la $+\infty$-IDL-consistance (pour laquelle, contrairement à la label-consistance, le calcul de la suite α n'est pas interrompu au rang 2), permet de résoudre toutes les instances sauf 12. Notons que ces 12 instances correspondent exactement à l'intersection des problèmes non résolus par l'ILL-consistance et par la label-consistance.

Ces résultats illustrent l'efficacité de la d-IDL-consistance : dans presque tous les cas, la d-IDL-consistance permet la résolution du problème. En outre, ces résultats montrent que, si la $+\infty$-IDL-permet un meilleur filtrage que la 1-IDL-consistance, le gain obtenu en terme de filtrage (5 instances de plus) n'est sans doute pas suffisant pour justifier le coût supplémentaire du calcul de la matrice des distances des sommets des graphes.

La figure 6.7 donne un exemple de paire de graphes non-isomorphes dont l'étiquetage des sommets par la suite α ne permet pas de détecter l'inconsistance du problème d'isomorphisme.

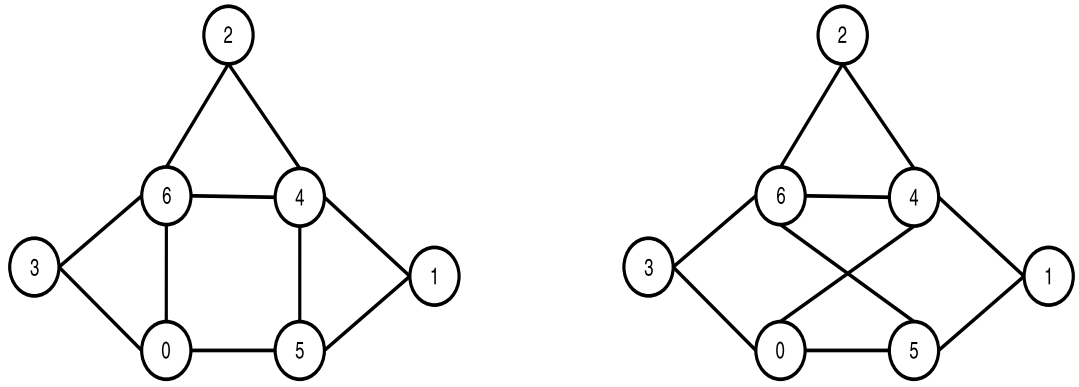


FIG. 6.7 – Deux graphes non-isomorphes dont l'étiquetage des sommets par la suite α ne permet pas de détecter l'inconsistance du problème d'isomorphisme.

6.7 Discussion

Contrainte *gip* et d-IDL-consistance. Nous introduisons dans ce chapitre une nouvelle contrainte globale pour les problèmes d'isomorphisme de graphes, la contrainte *gip*. Pour une utilisation efficace de cette contrainte, nous proposons une consistance partielle, la d-iterated-distance-label-consistance (d-IDL-consistance) et un algorithme permettant de l'établir. Cette consistance est basée sur le calcul, pour chaque sommet u, d'une étiquette qui caractérise les relations en termes de plus courte chaîne entre u et les autres sommets du graphe distants d'au plus d arcs du sommet u. Nous montrons que la d-IDL-consistance est d'autant plus forte que la distance d choisie est élevée.

Label consistance et ILL-consistance. Nous déclinons cette consistance en deux consistances particulières : la label-consistance (correspondant à la $+\infty$-IDL-consistance pour laquelle le calcul de la suite α est interrompu au rang 2) et l'iterated-local-label-consistance (l'ILL-consistance correspondant à la 1-IDL-consistance). Nous montrons expérimentalement que dans de nombreux cas, ces deux consistances permettent à un solveur de contraintes de détecter une inconsistance ou de réduire les domaines des variables d'un CSP modélisant un GIP à des singletons rendant alors la consistance globale du problème facile à vérifier.

Propagation des réductions. Pour les cas où la d-IDL-consistance ne permet pas de résoudre à elle seule le problème de l'isomorphisme de graphes, nous définissons un ensemble de contraintes de distance, sémantiquement équivalent à la contrainte globale *gip*, et permettant de propager les réductions de domaines. Nous montrons que ces contraintes de distances sont plus "strictes" que les contraintes d'arcs traditionnellement utilisées. Ces contraintes de distance permettent de réduire plus fortement les domaines des variables lors d'une propagation de contraintes. Les contraintes de distance peuvent être combinées avec une contrainte globale *allDiff* pour propager encore mieux les réductions.

Complexité. Le filtrage par d-IDL-consistance et la génération de l'ensemble des contraintes de distance peuvent être réalisés en $\mathcal{O}(np)$ opérations pour les graphes composés de n sommets et p arcs ($n-1 \leq p \leq n^2$). En comparaison, la consistance d'arcs avec AC2001 sur un CSP décrivant le problème de l'isomorphisme avec des contraintes d'arcs nécessite $\mathcal{O}(ed^2)$ opérations [BC93] où e est le nombre de contraintes, c.-à-d. $e = n(n-1)/2$, et d est la taille du plus grand domaine, c.-à-d. $d = n$. Établir la d-IDL-consistance sur notre contrainte globale est donc d'un ordre moins coûteux que la consistance d'arcs sur les contraintes binaires traditionnellement utilisées. Notons cependant que ces consistances ne sont pas comparables : pour certains graphes,

tel que le graphe de la figure 6.1, la d-IDL-consistance permet de résoudre le problème alors que AC sur les contraintes d'arcs ne réduit aucun domaine.

Performance. La consistance de labels (ou $+\infty$-IDL-consistance) nécessite le calcul de la matrice des distance entre chaque couple de sommets du graphe. Ce calcul est très coûteux, et, bien que leur complexité théorique dans le pire des cas soit la même, il est généralement plus intéressant d'utiliser l'ILL-consistance (1-IDL-consistance) qui ne nécessite pas le calcul de cette matrice. Nous montrons que, sur des graphes générés aléatoirement, ce filtrage est plus efficace que la label-consistance et surtout beaucoup plus rapide. L'utilisation de ce filtrage rend la programmation par contraintes compétitive avec Nauty, l'algorithme de référence pour le problème de l'isomorphisme de deux graphes.

Nous n'avons pas jugé nécessaire de tester la d-IDL-consistance avec des valeurs de d différentes de 1 et de $+\infty$. En effet, les performances de la 1-IDL-consistance sont suffisantes pour résoudre presque toutes les instances considérées.

Extension à l'isomorphisme de sous-graphe. Le problème de l'isomorphisme de sous-graphe (SGIP) consiste à montrer qu'un graphe est inclus dans un autre graphe (à un renommage de ses sommets près). Si la complexité théorique du GIP n'est pas encore parfaitement déterminée, SGIP est connu comme étant un problème NP-complet [GJ79]. De fait, SGIP est un problème plus difficile et nombreuses instances ne peuvent être résolues en un temps raisonnable.

La modélisation des SGIPs en CSPs est très similaire à celle des GIPs. Comme pour les GIPs, il est possible de tirer partie de la sémantique globale du problème pour définir des algorithmes de filtrage puissants. Cependant, contrairement aux fonctions d'isomorphisme de graphes (théorème 1), une fonction d'isomorphisme de sous-graphe $f : V \rightarrow V'$ ne préserve pas les distances entre les sommets des graphes : pour chaque chaîne $< v_1, ..., v_n >$ de G, il existe une chaîne $< f(v_1), ..., f(v_n) >$ dans G' mais l'opposé n'est pas toujours vrai (f n'est pas une bijection et il peut donc exister des sommets de V' qui ne sont reliés à aucun sommet de V). Par conséquent, la distance $\delta_G(u, v)$ entre deux sommets u et v de G peut être plus grande que la distance $\delta_{G'}(f(u), f(v))$ entre $f(u)$ et $f(v)$.

Les techniques de filtrage permettant de résoudre efficacement un SGIP peuvent être basées sur la propriété suivante : étant donnés deux graphes non-orientés $G = (V, E)$ et $G' = (V', E')$, si f est une fonction d'isomorphisme de sous-graphe entre G et un sous-graphe de G', alors :

$$\forall (u, v) \in V \times V,\ \delta_G(u, v) \geq \delta_{G'}(f(u), f(v))$$

Cette propriété peut être utilisée pour définir une consistance partielle et l'algorithme de filtrage qui lui est associé. L'idée consiste à vérifier que pour tous les sommets u de G, le domaine de la variable x_u associée à u ne contient que les sommets u' tels que, pour toutes les distances $k \in 1..|V| - 1$, le nombre de sommets $v \in G$ pour lesquels $\delta_G(u, v) \leq k$ est inférieur ou égal aux nombre de sommets $v' \in G'$ pour lesquels $\delta_{G'}(u', v') \leq k$.

Cette propriété peut aussi être utilisée pour définir des contraintes binaires $C_{distance}$ et propager les réductions de domaines pendant l'exploration d'un arbre de recherche.

Nous aimerions également intégrer ce travail dans le formalisme CP(Graph) proposé par Dooms *et al.* [DDD05b]. CP(Graph) est un nouveau domaine de calcul en programmation par contraintes permettant de définir des variables de type graphe et définissant un ensemble de contraintes applicables sur les graphes (appartenance d'un ensemble de sommets ou d'arcs à un graphe, inclusion d'un graphe dans un autre, existence d'un chemin entre deux sommets...). CP(Graph) a été introduit afin de faciliter la modélisation et la résolution des problèmes combinatoires sur les graphes en programmation par contraintes, en particulier ceux rencontrés lors de l'analyse de réseaux biochimiques.

Expérimentations sur des graphes particuliers. Nos expérimentations ont porté sur des graphes générés aléatoirement. Nous aimerions maintenant comparer Nauty, la label-consistance et l'ILL-consistance sur des graphes plus réguliers. En effet, certains graphes (très réguliers) sont tels que tous leurs sommets ont le même nombre de sommets voisins (et ou le même nombre de sommets à distance 2,3...). Sur ces graphes, la consistance que nous proposons ne permet pas de filtrer le domaine des variables avant le déroulement d'un arbre de recherche et Nauty est parfois très long à s'exécuter. Nous aimerions donc intégrer notre algorithme de filtrage (et ses propagations de réductions) dans un solveur de contraintes (par exemple CHOCO [Lt00]) afin d'évaluer les performances de ce filtrage sur les graphes réguliers. Nous pensons également que, bien que sur les instances que nous avons considéré la 1-IDL-consistance offrait le meilleur compromis temps d'exécution/efficacité, sur des graphes réguliers, la 2-IDL ou la 3-IDL-consistance peuvent s'avérer meilleures.

Conclusions et perspectives

1 Conclusions

1.1 Mesure de distance de graphes

Il existe de nombreuses mesures de distance de graphes. Toutes ces mesures reposent sur la recherche d'un meilleur appariement des sommets des deux graphes, c.-à-d. une relation entre leurs sommets. Néanmoins, ces distances peuvent différer les unes des autres sur de nombreux points :

- L'appariement recherché peut respecter des contraintes dures (imposant aux graphes à avoir des structures identiques) ou des préférences (imposant alors aux graphes à avoir des structures similaires sans nécessairement être identiques) ;
- L'appariement recherché peut être univoque (chaque sommet devant alors être mis en relation avec au plus un autre sommet) ou multivoque (les sommets pouvant alors être mis en relation avec un ensemble de sommets de l'autre graphe) ;
- Les graphes manipulés peuvent ou non être valués, des valeurs numériques ou symboliques pouvant alors être ajoutées sur chacun de leurs sommets ou de leurs arcs. L'appariement recherché devant alors prendre en compte ces valeurs en respectant des contraintes dures ou des préférences sur les valeurs des sommets et des arcs mis en correspondance.

Nous proposons une mesure générique de la distance de deux graphes. Cette mesure à trois caractéristiques principales :

- elle est basée sur la recherche d'un meilleur appariement multivoque des sommets des graphes à comparer ;
- elle permet d'exprimer des contraintes dures et des préférences (individuelles) entre les arcs et les sommets appariés, la distance entre deux graphes étant alors une agrégation de ces préférences ;
- elle permet de comparer des graphes de tout type : étiquetés, valués, multi-étiquetés...

Nous montrons que notre mesure de la distance de deux graphes est générique. En effet, cette mesure est paramétrable et permet de modéliser les mesures de distance ou de similarité de graphes existantes. En particulier, nous montrons comment l'utiliser pour modéliser le problème de l'isomorphisme de (sous-) graphe, celui du calcul de la distance d'édition de graphe (étendue ou non), celui de la recherche du plus grand sous-graphe commun (partiel ou induit) entre deux graphes, la mesure de similarité de Boeres *et al.* [BRB04], la mesure de Champin et Solnon [CS03]...

Les mesures de distance ou de similarité de graphes existantes utilisent des formalismes très différents les uns des autres ce qui rend la comparaison de ces mesures difficiles. Avoir une mesure de distance générique rend la comparaison des mesures existantes plus simple. Elle offre également un cadre uniformisé permettant une définition plus aisée de nouvelles mesures de distance de graphes. Enfin, un même algorithme, capable de calculer cette distance générique,

peut être utilisé pour calculer des mesures de similarité ou de distance de graphes très différentes.

1.2 Calcul de la distance de deux graphes

Recherche locale. Calculer la distance entre deux graphes est, dans le cas général, un problème difficile en théorie comme en pratique. Nous proposons deux algorithmes de calcul de cette mesure générique : une recherche locale taboue et une version réactive de cet algorithme capable de s'auto-paramétrer dynamiquement afin de réguler intensification et diversification de la recherche. Ces algorithmes sont des recherches incomplètes ne garantissant pas l'optimalité de la solution retournée mais ayant une complexité polynomiale.

Résultats expérimentaux. Ces algorithmes étant dédiés au calcul de notre mesure de distance générique, nous avons pu les expérimenter sur des problèmes d'appariements de graphes très différents les uns des autres. Nos tests montrent que nos algorithmes sont utilisables mais pas compétitifs avec des algorithmes dédiés sur des problèmes d'isomorphisme de sous-graphe. Nos algorithmes se montrent cependant très compétitifs sur des problèmes plus difficiles. En particulier, sur la recherche d'un plus grand sous-graphe commun entre deux graphes, nos algorithmes permettent de résoudre en un temps très court des instances qui ne peuvent pas être résolues en un temps raisonnable par une recherche complète arborescente.

Nous comparons également nos algorithmes à l'algorithme `ANT-GM'06` [SSSG06a] sur des problèmes d'appariement non-bijectifs de Boeres *et al.* [BRB04] et sur des problèmes de recherche d'un meilleur appariement multivoque. `ANT-GM'06` et nos algorithmes obtiennent des résultats complémentaires : nos algorithmes obtiennent des résultats parfois légèrement moins bons que ceux de `ANT-GM'06` mais sont plus rapides que ce dernier.

Paramétrage, réactivité. L'algorithme Tabou que nous proposons est difficile à paramétrer. Il admet un seul paramètre (régulant l'intensification et la diversification de la recherche) mais ce paramètre est fragile et critique : le paramétrage optimal varie d'une instance d'un problème à une autre et un mauvais paramétrage dégrade fortement les résultats. Nous proposons donc d'insérer un processus de réactivité pour que la recherche s'auto-paramètre dynamiquement en fonction des besoins de diversification ou d'intensification de la recherche. Les résultats obtenus montrent que cet algorithme réactif est parfois plus simple à paramétrer et plus efficace que la version non réactive. Néanmoins, sur des problèmes très difficiles, il reste nécessaire de trouver un bon paramétrage de cet algorithme pour obtenir des résultats compétitifs.

Programmation par contraintes et isomorphisme de graphes. Nous proposons une contrainte globale dédiée à la modélisation des problèmes d'isomorphisme de graphes (la contrainte *gip*), une consistance partielle pour cette contrainte (la d-IDL consistance) et l'algorithme de filtrage permettant de l'établir. Nous montrons que ce filtrage est très efficace et rend la programmation par contraintes compétitive avec Nauty, le meilleur algorithme connu dédié à ce problème. Des instances à 10,000 sommets peuvent être résolues en quelques secondes seulement grâce à cette consistance partielle.

2 Perspectives

2.1 Distance générique de graphe et modélisation de problèmes

Notre mesure générique de distance de graphe permet de modéliser de nombreuses mesures de distance ou de similarité existantes. Bien que nous ayons restreint notre attention à ces problèmes, notre mesure permet également de représenter de façon élégante de nombreux autres problèmes. En particulier, le problème de la recherche de clique maximum dans un graphe G peut être très

facilement modélisé comme le calcul de la distance entre G et un graphe complet. Le problème du Sudoku peut également être modélisé comme un calcul de la distance de deux graphes. Le premier graphe modélise la grille de Sudoku : c'est un graphe de 81 sommets représentant les 81 cases de la grille où un arc est présent entre deux cases si et seulement si ces cases appartiennent à la même ligne, la même colonne ou le même carré 3×3 de la grille. Le second graphe modélise les symboles : c'est un graphe complet (hors boucles) de 9 sommets où chaque sommet représente un symbole. La distance d'une case doit être nulle quand cette case est appariée à un et un seul symbole et $+\infty$ sinon. La distance d'un arc de la grille est nulle quand l'arc est apparié à un et un seul arc du graphe symboles et $+\infty$ sinon. Si une case c de la grille est pré-remplie par un symbole s, la fonction de distance des sommets retourne 0 si s est apparié à c et $+\infty$ sinon. Il est alors facile de montrer que la distance entre les deux graphes est nulle si et seulement si l'appariement représente une grille de Sudoku complète et valide car la distance est nulle si et seulement si chaque case est associée à un symbole et que toutes les contraintes de différences entre les cases d'une même ligne, d'une même colonne ou d'un même carré sont respectées.

De façon générale, de nombreux problèmes de satisfaction de contraintes peuvent être exprimés comme le calcul de la distance entre deux graphes. Nous aimerions approfondir les liens existants entre la PPC et notre distance de deux graphes. En particulier, nous aimerions pouvoir comparer l'expressivité de notre mesure et celle de la PPC.

Afin de rendre la résolution d'un problème plus facile, la PPC utilise des contrainte globales permettant l'utilisation d'algorithmes de filtrage efficaces. Nous aimerions également explorer les moyens d'introduire dans notre mesure des connaissances supplémentaires permettant une résolution plus simple des problèmes par notre algorithme de calcul de la distance de deux graphes. Par exemple, l'idée à l'origine de notre algorithme de filtrage pour le problème de l'isomorphisme de graphe consiste à transformer les graphes à apparier en graphes complets où chaque arc est étiqueté par la distance séparant les deux extrémités de l'arc. C'est modélisation permet d'introduire de façon élégante les contraintes de distance $C_{distance}$ que nous avons définies.

2.2 Calculer la distance de deux graphes

Nous aimerions améliorer les résultats de nos algorithmes. En particulier, le processus de réactivité que nous avons mis en place ne nous paraît pas pleinement satisfaisant : les résultats obtenus par notre algorithme réactif sont globalement meilleurs mais, sur certains problèmes, cet algorithme nécessite un paramétrage parfois difficile pour donner de bons résultats. En outre, nos algorithmes n'utilisent aucune procédure de filtrage pendant la recherche ce qui les handicape fortement sur certains problèmes (notamment l'isomorphisme de sous-graphe).

Nous aimerions également tester nos algorithmes sur d'autres problèmes d'appariement multivoque de graphes. Il n'existe pas à notre connaissance de benchmark de ce type de problèmes ce qui nous a contraint à générer nous même de tels problèmes. Nous aimerions proposer à la communauté un benchmark de problèmes d'appariements multivoques de graphes.

2.3 Programmation par contraintes

Nous aimerions intégrer notre algorithme de filtrage pour l'isomorphisme de graphes dans un solveur de contraintes (par exemple CHOCO) afin de tester l'efficacité de la PPC sur des graphes de différent types (par exemple des graphes réguliers, des grilles...).

Nous aimerions également voir dans quelle mesure l'idée d'utiliser un filtrage basé sur les distances entre les sommets des graphes à apparier peut être transposée au problème de l'isomorphisme de sous-graphe ou à la recherche du plus grand sous-graphe commun à deux graphes.

2.4 Applications

Les travaux que nous présentons dans cette thèse sont assez théoriques et nous ne proposons pas un exemple concret d'application utilisant notre mesure générique de la distance de deux graphes. Nous aimerions nous associer avec des spécialistes de la reconnaissance d'images afin de proposer une application de classification sémantique d'images basée sur la comparaison de graphes. La biologie ou la chimie seraient aussi des domaines vers lesquels nous aimerions nous pencher.

Bibliographie

[AACGJ01] Hervé Albin-Amiot, Pierre Cointe, Yann-Gaël Guéhéneuc, and Narendra Jussien. Instantiating and detecting design patterns : Putting bits and pieces together. In Debra Richardson, Martin Feather, and Michael Goedicke, editors, *Proceedings of the 16th ASE conference*, pages 166 – 173. IEEE Computer Society Press, November 2001.

[AB93] Abderrahmane Aggoun and Nicolas Beldiceanu. Overview of the chip compiler system. *Constraint Logic Programming. Selected Research*, pages 421–435, 1993.

[AFB03] Roman Ambauen, Stefan Fischer, and Horst Bunke. Graph edit distance with node splitting and merging and its application to diatom identification. *4th IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition*, LNCS 2726-Springer :95–106, August 2003.

[AFC98] Giuliano Antoniol, Roberto Fiutem, and Lucas Cristoforetti. Design pattern recovery in object-oriented software. In *Proceedings of the 6th International Workshop on Program Comprehension*, pages 153 – 160. IEEE Computer Society, Washington, DC, USA, June 24 – 26 1998.

[AHU74] Alfred V. Aho, John .E. Hopcroft, and Jeff .D. Ullman. *The design and analysis of computer algorithms*. Addison Wesley, 1974.

[BC93] Christian Bessiere and Marie-Odile Cordier. Arc-consistency and arc-consistency again. In *AAAI-93 : Proceedings 11th National Conference on Artificial Intelligence*, 1993.

[Ben02] Endika Bengoetxea. *Inexact Graph Matching Using Estimation of Distribution Algorithms*. PhD thesis, Ecole Nationale Supérieure des Télécommunications (Paris), Département Traitement du Signal et des Images, 2002.

[BFG+02] Horst Bunke, Pasquale Foggia, Corrado Guidobaldi, Carlo Sansone, and Mario Vento. A comparison of algorithms for maximum common subgraph on randomly connected graphs. In *Proc. IAPR Workshop on Structural and Syntactic Pattern Recognition*, 2002.

[BFM+96] Stefano Bistarelli, Hélène Fargier, Ugo Montanari, Francesca Rossi, Thomas Schiex, and Gérard Verfaillie. Semiring-based csps and valued csps : Basic properties and comparison. In *Over-Constrained Systems (Selected papers from the Workshop on Over-Constrained Systems at CP'95, reprints and background papers)*, 1996.

[BFR99] Christian Bessière, Eugene C. Freuder, and Jean-Charles Régin. Using constraint metaknowledge to reduce arc consistency computation. *Artificial Intelligence*, 107 :125–148, 1999.

[BH99] Mériéma Belaidouni and Jin-Kao Hao. Landscapes of the maximal constraint satisfaction problem. In *Evolution Artificielle 99 (EA'99)*, volume 1829 of *LNCS*, pages 244–255, 1999.

[BH03a] Christian Bessiere and Pascal Van Hentenryck. Etre ou ne pas être ... une contrainte globale. In *9ème Journées Nationales sur la résolution pratique de problèmes NP-Complet (JNPC'03)*, 2003.

[BH03b] Christian Bessière and Pascal Van Hentenryck. To be or not to be a global constraint. In Francesca Rossi, editor, *Principles and Practice of Constraint Programming ? CP 2003 : 9th International Conference, CP 2003*, pages 789–794. Springer, September 2003.

[BJ00] Horst Bunke and Xiaoyi Jiang. *H.-N. Teodorescu and D. Mlynek and A. Kandeland H.-J. Zimmermann : Intelligent Systems and Interfaces*, chapter Chapter 1 - Graph matching and similarity, page Chapter 1. Kluwer Academic Publishers, 2000.

[BLB+02] Endika Bengoetxea, Pedro Larrañaga, Isabelle Bloch, Aymeric Perchant, and Claudia Boeres. Inexact graph matching by means of estimation of distribution algorithms. *Pattern Recognition*, 35(12) :2867–2880, 2002.

[BP01] Roberto Battiti and Marco Protasi. Reactive local search for the maximum clique problem. *Algorithmica*, 29(4) :610–637, April 2001.

[BRB04] Maria Boeres, Celso Ribeiro, and Isabelle Bloch. A randomized heuristic for scene recognition by graph matching. In *Workshop on Experimental and Efficient Algorithms (WEA 2004)*, pages 100–113, 2004.

[BS98] Horst Bunke and Kim Shearer. A graph distance metric based on the maximal common subgraph. *Pattern recognition letters*, 19(3–4) :255–259, 1998.

[BT94] Roberto Battiti and Giampietro Tecchiolli. The reactive tabu search. *ORSA Journal on Computing*, 6(2) :126–140, 1994.

[Bun97] Horst Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters*, 18 :689 – 694, August 1997.

[Bun00] Horst Bunke. Graph matching : Theoretical foundations, algorithms, and applications. In *Proc. Vision Interface 2000, Montreal*, 2000.

[Cer85] V. Cerny. A thermodynamical approach to the travelling salesman problem : an efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45 :41–51, 1985.

[CFCS01] Luigi P. Cordella, Pasquale Foggia, and Mario Vento Carlo Sansone. An improved algorithm for matching large graphs. *3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition*, 2001.

[CFSV99] Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. Performance evaluation of the vf graph matching algorithm. In *Proceedings of the 10th International Conference on Image Analysis and Processing (ICIAP'99)*, page 1172. IEEE, 1999.

[CFSV00] Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. Fast graph matching for detecting cad image components. In *15th International Conference on Pattern Recognition (ICPR'00)*, volume 2, page 6034, 2000.

[Cha02] Pierre-Antoine Champin. *Modéliser l'expérience pour en assister la réutilisation : de la Conception Assistée par Ordinateur au Web Sémantique*. PhD thesis, Université Claude Bernard - Lyon 1, 2002.

[CS03] Pierre-Antoine Champin and Christine Solnon. Measuring the similarity of labeled graphs. In *5th International Conference on Case-Based Reasoning (ICCBR 2003)*, volume 2689 of *LNCS*, pages 80–95. Springer-Verlag, 2003.

[dBSL04] Davi de Castro Reis, Paulo Braz Golgher, Altigran Soares da Silva, and Alberto H. F. Laender. Automatic web news extraction using tree edit distance. In *Proceedings of the 13th international conference on World Wide Web*, pages 502–511. ACM Press, 2004.

[DD99] Marco Dorigo and Gianni Di Caro. The ant colony optimization meta-heuristic. *New Ideas in Optimization*, pages 11–32, 1999.

[DDD05a] Grégoire Dooms, Yves Deville, and Pierre Dupont. Constrained metabolic network analysis : discovering pathways using cp(graph). In *Workshop on Constraint Based Methods for Bioinformatics, CP2005*, 2005.

[DDD05b] Grégoire Dooms, Yves Deville, and Pierre Dupont. Cp(graph) : Introducing a graph computation domain in constraint programming. In *Proceedings on Principles and Practice on Constraint Programming (CP2005)*, 2005.

[DHLJ05] Aline Deruyver, Yann Hodé, Eric Leammer, and Jean-Michel Jolion. Adaptive pyramid and semantic graph : Knowledge driven segmentation. In Luc Brun and Mario Vento, editors, *Graph-Based Representations in Pattern Recognition : 5th IAPR International Workshop, GbRPR 2005, Poitiers, France, April 11-13, 2005. Proceedings*, volume 3434 of *LNCS*, page 213. Springer, 2005.

[DLSM04] Paul T. Darga, Mark H. Liffiton, Karem A. Sakallah, and Igor L. Markov. Exploiting structure in symmetry detection for cnf. *DAC*, pages 530–554, 2004.

[DS05] Marco Dorigo and Thomas Stützle. *Ant Colony Optimization*. MIT Press, 2005.

[Epp99] David Eppstein. Subgraph isomorphism in planar graphs and related problems. *Journal of Graph Algorithms and Applications*, 3(3) :1–27, 1999.

[For96] Scott Fortin. The graph isomorphism problem. Technical report, Dept. of Computer Science, University of Alberta, 1996.

[FSV01] Pasquale Foggia, Carlo Sansone, and Mario Vento. A database of graphs for isomorphism and sub-graph isomorphism benchmarking. *3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition*, pages 176 –187, 2001.

[GH97] Philippe Galinier and Jin-Kao Hao. Tabu search for maximal constraint satisfaction problems. In *Proceedings of Principles and Practice of Constraint Programming (CP'97)*, volume 1330 of *LNCS*, pages 196–208. Springer Verlag, 1997.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns, Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[Glo89] Fred Glover. Tabu search - part i. *Journal on Computing*, pages 190–260, 1989.

[GW93] Ian P. Gent and Toby Walsh. An empirical analysis of search in gsat. *Journal of Artificial Intelligence Research*, 1 :47–59, 1993.

[HW74] John E. Hopcroft and Jin-Kue Wong. Linear time algorithm for isomorphism of planar graphs. In *Proc. 6th Annual ACM Symposium on Theory of Computing*, pages 172–184, 1974.

[ILO00] ILOG,S.A. *ILOG Solver 5.0 User's Manual and Reference Manual, 2000*, 2000.

[JL02] Narendra Jussien and Olivier Lhomme. Local search with constraint propagation and conflict-based heuristics. *Artificial Intelligence*, 139(1) :21–45, 2002.

[Jol04] Jean-Michel Jolion. On the deviation of a set of strings. *Pattern Analysis and Applications*, 6(3) :224–231, 2004.

[Jon72] Sparck K. Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28 :11–21, 1972.

[KGV83] Scott Kirkpatrick, Junior D. Gelatt, and Mario P. Vecchi. Optimisation by simulated annealing. *Science*, 220 :671 – 680, 1983.

[Lev65] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals (in russian). *Doklady Akademii Nauk SSSR*, 163(4) :845–848, 1965.

[Lin98] Dekang Lin. An information-theoretic definition of similarity. In *Proc. 15th International Conf. on Machine Learning*, pages 296–304. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 1998.

[LMS02] Helena Ramalhino Lourenco, Olivier Martin, and Thomas Stützle. *Handbook of Metaheuristics*, chapter 11, pages 321–353. Kluwer Academic Publishers, 2002.

[LNML03] Florence Le Ber, Amedeo Napoli, Jean-Luc Metzger, and Sylvie Lardon. Modeling and comparing farm maps using graphs and case-based reasoning. *Journal of Universal Computer Science*, 9(9) :1073–1095, 2003.

[Lt00] François Laburthe and the OCRE project team. CHOCO : implementing a CP kernel. In *Proc. of the CP'2000 workshop on techniques for implementing constraint programming systems*, 2000.

[Luk82] Eugene M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer System Science*, pages 42–65, 1982.

[LV97] Javier Larrosa and Gabriel Valiente. Graph pattern matching using constraint satisfaction. In *Proc. Fourth South American Workshop on String Processing*, volume 8 of *International Informatics Series,*, pages 180–197. Carleton University Press, 1997.

[Mac77] Alan K. Mackworth. Consistency in networks of relations. *Artificial intelligence*, 8 :99–118, 1977.

[McG79] James J. McGregor. Relational consistency algorithms and their applications in finding subgraph and graph isomorphisms. *Information Science*, 19 :229–250, 1979.

[McG82] James J. McGregor. Backtrack search algorithms and the maximal common subgraph problem. *Software-Practice and Experience*, 12 :23 – 34, 1982.

[McK81] Brendan D. McKay. Practical graph isomorphism. *Congressus Numerantium*, 1981. Nauty.

[MF85] Alan K. Mackworth and Eugene C. Freuder. The complexity of some polynomial network consistency algorithms for some constraint satisfaction problems. *Artificial intelligence*, 25 :65–74, 1985.

[MGMR02] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity flooding : A versatile graph matching algorithm and its application to schema matching. In *18th International Conference on Data Engineering (ICDE'02)*, 2002.

[MH86] Roger Mohr and Thomas C. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28 :225–233, 1986.

[NB04] Michel Neuhaus and Horst Bunke. An error-tolerant approximate matching algorithm for attributed planar graphs and its application to fingerprint classification. In Ana L. N. Fred, Terry Caelli, Robert P. W. Duin, Aurélio C. Campilho, and Dick de Ridder, editors, *Structural, Syntactic, and Statistical Pattern Recognition (SSPR'04)*, number 3138 in LNCS, pages 180–189. Springer, 2004.

[NRB06] Michel Neuhaus, Kaspar Riesen, and Horst Bunke. Fast suboptimal algorithms for the computation of graph edit distance. In *Structural and Syntactic Pattern Recognition (SSPR 2006)*, 2006.

[PKY02] Sanja Petrovic, Graham Kendall, and Yong Yang. A tabu search approach for graph-structured case retrieval. In IOS Press, editor, *Proceedings of the STarting Artificial Intelligence Researchers Symposium STAIRS 2002, Lyon, France*, pages 55–64, 2002.

[Pug05] Jean-François Puget. Automatic detection of variable and value symmetries. In *Principles and Practice of Constraint Programming - CP 2005*, volume 3709, pages 475–489, 2005.

[PV04] Cédric Pralet and Gérard Verfaillie. Travelling in the world of local searches in the space of partial assignments. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems : First International Conference, CPAIOR 2004*, 2004.

[RLJS05] Julien Ros, Christophe Laurent, Jean-Michel Jolion, and Isabelle Simand. Comparing string representations and distances in a natural images classification task. In *Graph based Representation in Pattern Recognition*, pages 72–81, 2005.

[Rég95] Jean-Charles Régin. *Développement d'Outils Algorithmiques pour l'Intelligence Artificielle. Application à la Chimie Organique.* PhD thesis, Université Montpellier II, 1995.

[SB91] Michael J. Swain and Dana H. Ballard. Color indexing. *International Journal of Computer Vision*, 7 :11–32, 1991.

[SBV00] Kim Shearer, Horst Bunke, and Svetha Venkatesh. Video indexing and similarity retrieval by largest common subgraph detection using decision trees. *Pattern Recognition*, pages 1075–1091, 2000.

[SCS03] Sébastien Sorlin, Pierre-Antoine Champin, and Christine Solnon. Mesurer la similarité de graphes étiquetés. In *9ème Journées Nationales sur la résolution pratique de problèmes NP-Complet (JNPC)*, pages 325–339, 2003.

[SD76] Douglas Schmidt and Larry Druffel. A fast backtracking algorithm to test directed graphs for isomorphism using distance matrices. *Journal of the ACM (JACM)*, 23(3) :433–445, July 1976.

[Sel77] Stanley M. Selkow. The tree-to-tree editing problem. *Information Processing Letters*, 6 :184–186, 1977.

[SH00] Thomas Stützle and Holger H. Hoos. Max-min Ant System. *Journal of Future Generation Computer Systems*, 16 :889–914, 2000.

[SKC93] Bart Selman, Henry A. Kautz, and Bram Cohen. Local search strategies for satisfiability testing. In Michael Trick and David Stifler Johnson, editors, *Proceedings of the Second DIMACS Challenge on Cliques, Coloring, and Satisfiability*, Providence RI, 1993.

[SLBK04] Adam Schenker, Mark Last, Horst Bunke, and Abraham Kandel. Classification of web documents using graph matching. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(3) :475–496, 2004.

[SLM92] Bart Selman, Hector J. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In Paul Rosenbloom and Peter Szolovits, editors, *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 440–446. AAAI Press, 1992.

[SS04a] Sébastien Sorlin and Christine Solnon. A global constraint for graph isomorphism problems. In *the 1st International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR 2004)*, number 3011 in LNCS, pages 287–301. Springer-Verlag, Avril 2004.

[SS04b] Sébastien Sorlin and Christine Solnon. Une contrainte globale pour le problème de l'isomorphisme de graphes. In Hermes, editor, *13ème Journées Francophones de Programmation en Logique par Contraintes, JFPLC 2004, Angers*, pages 91–107, Juin 2004.

[SS05a] Sébastien Sorlin and Christine Solnon. Reactive tabu search for measuring graph similarity. In Luc Brun and Mario Vento, editors, *5th IAPR-TC-15 workshop on Graph-based Representation in Pattern Recognition*, pages 172–182. Springer Verlag, 2005.

[SS05b] Sébastien Sorlin and Christine Solnon. Similarité de graphes : une mesure générique et un algorithme tabou réactif. In Emmanuel Guéré, editor, *7èmes rencontres jeunes chercheurs en intelligence artificielle RJCIA'05*, pages 253–266. PUG, 2005.

[SS06] Sébastien Sorlin and Christine Solnon. A new filtering algorithm for the graph isomorphism problem. In *(CPAI'06) CP'06 workshop on Constraint Propagation And Implementation*, 2006.

[SSJ06a] Sébastien Sorlin, Christine Solnon, and Jean-Michel Jolion. *Applied Graph Theory*, chapter A Generic Graph Distance Measure Based on Multivalent Matching. (a paraître chez Springer en 2007, 30 pages), 2006.

[SSJ06b] Sébastien Sorlin, Christine Solnon, and Jean-Michel Jolion. Mesurer la similarité de graphes. In Nicole Vincent et Nicolas Lomenie, editor, *Extraction de COnnaissance à partir d'Images, Atelier de Extraction et Gestion de Connaissances (EGC06)*, pages 21–30. ed. Lille, 2006.

[SSSG06a] Olfa Sammoud, Sébastien Sorlin, Christine Solnon, and Khaled Ghédira. A comparative study of ant colony optimization and reactive search for graph matching problems. In Jens Gottlieb et Günther Raidl, editor, *6th European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP 2006)*, number 3906 in LNCS, pages 234–246. Springer, April 2006.

[SSSG06b] Sébastien Sorlin, Olfa Sammoud, Christine Solnon, and Khaled Ghedira. Etude comparative de aco et de tabou réactif sur des problèmes d'appariement de graphes. In Laurent Henocque, editor, *2èmes Journées Francophones de Programmation par Contraintes*, pages 317–326, Juin 2006.

[Tve77] Amos Tversky. Features of similarity. *Psychological Review*, 84 :327–352, 1977.

[Ull76] Jeff .R. Ullman. An algorithm for subgraph isomorphism. *Journal of the Association of Computing Machinery*, pages 31–42, 1976.

[ZDD05] Stéphane Zampelli, Yves Deville, and Pierre Dupont. Approximate constrained subgraph matching. In *11th International Conference on Principles and Practice of Constraint Programming*, number 3709 in Lecture Notes in Computer Science, pages 832–836. Springer, 2005.

[ZDD06] Stéphane Zampelli, Yves Deville, and Pierre Dupont. Elimination des symétries pour l'appariement de graphes. In Laurent Henocque, editor, *JFPC'06, Deuxièmes Journées Francophones de Programmation par Contraintes*, pages 357–367, 2006.

Résumé

De nombreuses applications, comme par exemple la recherche ou la classification d'informations, nécessitent de mesurer la distance ou la similarité entre deux graphes, *i.e.*, apparier —mettre en correspondance— les sommets des graphes afin d'identifier leurs points communs et leurs différences.

Il existe différents types d'appariements de graphes donnant chacun lieu à une définition différente de la distance entre deux graphes. Les appariements exacts (isomorphisme de graphes ou de sous-graphe) permettent de montrer que deux graphes sont identiques ou qu'un graphe est inclus dans un autre graphe. Cependant, dans de nombreuses applications, supposer l'existence d'un tel appariement est une hypothèse trop forte. Par conséquent, des appariements de graphes à tolérance d'erreurs tels que la recherche du plus grand sous-graphe commun à deux graphes ou la distance d'édition de graphes ont été proposés. L'appariement recherché est alors un "meilleur" appariement, *i.e.*, un appariement devant préserver le plus grand nombre de sommets et d'arcs des graphes sans pour autant nécessairement tous les préserver. Plus récemment, trois différentes approches ont proposé d'aller un cran plus loin en introduisant la notion d'appariment multivoque où le sommet d'un graphe peut être apparié à un ensemble de sommets de l'autre graphe. Ce type d'appariement permet de prendre en compte le fait que le composant d'un objet modélisé par un graphe peut "jouer le même rôle" que plusieurs composants d'un autre objet modélisé par un autre graphe.

Un premier objectif de cette thèse est de définir une nouvelle distance de graphe basée sur la recherche d'un meilleur appariement entre les sommets de deux graphes, *i.e.*, un appariement qui minimise des fonctions de distance de sommets et d'arcs. Cette distance de graphe est générique dans le sens où elle permet des appariements univoques ou multivoques et où elle est paramétrable en fonction de l'application considérée. Nous montrons comment utiliser ce cadre générique de définition de la distance entre deux graphes pour modéliser les mesures de distance ou de similarité de graphes existantes.

Un second objectif de cette thèse est de proposer une solution algorithmique permettant le calcul de notre mesure générique de la distance de deux graphes. Nous proposons et expérimentons un algorithme de recherche locale taboue capable de résoudre de nombreux problèmes différents d'appariement de graphes.

Nous nous intéressons ensuite plus spécifiquement à la résolution du problème de l'isomorphisme de deux graphes à l'aide de la programmation par contraintes. Nous proposons une contrainte globale dédiée à ce problème, une consistance partielle pour cette contrainte et l'algorithme permettant de l'établir. Nous montrons alors que l'utilisation de cette contrainte globale permet à la programmation par contraintes de devenir compétitive avec des approches dédiées à ce problème.

Mots-clés: Graphe, distance, similarité, recherche locale, tabou réactif, PPC, isomorphisme de graphe, contrainte globale

Abstract

Many applications such as *e.g.*, information retrieval and classification, involve measuring graph distance or similarity, *i.e.,* matching graphs to identify and quantify their common features.

Different kinds of graph matchings have been proposed, giving rise to different graph similarity or distance measures. Exact graph matchings such as graph or subgraph isomorphism can be used in order to show graph equivalence or inclusion. However, in many applications, the assumption of the existence of such an "exact" matching is too strong. As a consequence, error-tolerant graph matchings such as maximum common subgraph and graph edit distance have been proposed. Such matchings drop the condition that the matching must preserve all vertices and edges and look for a "best" matching, *i.e.,* one which preserves a maximum number of vertices and edges. Most recently, three different approaches proposed to go one step further by introducing multivalent matchings where a vertex may be matched with a set of vertices. This kind of matching handles the fact that, due to different description granularity levels, one object component may "play the same role" than a set of components of another object.

Un first goal of this work is to define a new graph distance based on the search of a best matching between the graph vertices, *i.e.*, a matching that minimizes vertex and edge distance functions. This distance is generic in the sense that it allows both univalent and multivalent matchings and it is parameterized by vertex and edge distance functions defined by the user depending on the considered application. We show how to use this graph distance generic framework to model existing graph distance or similarity measure.

A second goal of this work is to propose an algorithm to compute this generic graph distance. We propose a reactive tabu local search ables to solve many different graph matching problems and give some experimental results.

We then focus our attention on solving the graph isomorphism problem with constraint programming. We propose a global constraint dedicated to this problem, a partial consistency for this constraint and an algorithm to establish this consistency. We show that using this consistency makes the constraint programming competitive with algorithms dedicated to this problem.

Keywords: Graph, distance, similarity, local search, reactive tabu search, constraint programming, graph isomorphism, global constraint