Recherche taboue réactive pour mesurer la similarité de graphes multi-étiquetés

Sébastien Sorlin et Christine Solnon
LIRIS, CNRS FRE2672, bât. Nautibus, University of Lyon I
43 Bd du 11 novembre, 69622 Villeurbanne cedex, France
{sebastien.sorlin,christine.solnon}@liris.cnrs.fr

Février 2004

Abstract

Many applications need to measure the similarity of objects, *i.e.*, to identify and to quantify both commonalities and differences of objects. When these objects are represented by graphs, this measure consists in solving a graph matching problem. In this paper, we are interesting in a similarity measure based on the best multivalent mapping of multi-labeled graphs. This measure is interesting in many applications but computing it is a very hard problem. We propose here a local search based algorithm with a reactive Tabu list to solve this problem and we compare this algorithm with his non reactive version.

De nombreuses applications nécessitent de mesurer la *similarité* d'objets, *i.e.*, d'identifier et de quantifier leurs points communs et leurs différences. Lorsque ces objets sont représentés par des graphes, cette mesure se ramène à un problème d'appariement de graphes. Dans ce document, nous nous intéressons à une mesure de similarité basée sur le *meilleur appariement multivoque* de *graphes multi-étiquetés*. Cette mesure de similarité est pertinente dans beaucoup d'applications mais son calcul est un problème très difficile. Pour résoudre ce problème, nous proposons dans cet article un algorithme de recherche locale avec une liste taboue réactive que nous comparons avec sa version non réactive.

Introduction

De la nécessité d'une mesure de similarité. Evaluer la similarité de deux objets, c'est à dire identifier leurs points communs et leurs différences, est un problème qui se pose dans de nombreuses applications. Un corollaire de ce problème est de retrouver, parmi un ensemble d'objets, l'objet le plus similaire à un autre. Par exemple, lorsque l'on recherche des informations sur le Web, il s'agit de localiser, dans le gigantesque réseau que constitue la toile informatique, la partie qui répond au mieux à la requête de l'internaute. De façon plus

générale, de nombreuses autres applications nécessitent de comparer des objets : la recherche d'image par le contenu, la comparaison de protéines, la recherche de cas similaires en raisonnement à partir de cas, l'interrogation de bases de données semi-structurées...

Dans ces différentes applications, la fonction de comparaison doit être capable de faire des approximations dans le sens où l'on cherche l'objet *le plus similaire* à un autre et non pas un objet *identique* : en reconnaissance d'images, les images manipulées sont souvent bruitées ce qui introduit des différences entre des images pourtant identiques à l'origine. En recherche d'information, s'il n'existe pas de document répondant parfaitement à la requête d'un utilisateur, il est alors souhaitable de lui présenter une liste de documents *semblables* au document attendu classés par ordre de similarité décroissante.

De l'utilisation des graphes pour représenter des objets. Les graphes sont un outil puissant et souple pour la représentation d'objets stucturés : les sommets d'un graphe représentent les composantes d'un objet tandis que les arcs décrivent les relations binaires entre ces différentes composantes. Notons que ces graphes sont généralement typés ou étiquetés afin de distinguer les différents types de composantes et de relations. De nombreux domaines scientifiques utilisent les graphes pour représenter des données telles que des molécules [11], des protéines [1, 15] ou des images [3, 14]. Les graphes sont aussi souvent utilisés en ingénierie : conception assistée par ordinateur [8], ingénierie logicielle [13], électronique...

Plan. En partie 1 nous définissons les graphes multi-étiquetés ainsi qu'une nouvelle mesure de similarité pour comparer ces graphes. Nous illustrons ensuite la pertinence de cette mesure dans quelques applications. La partie 2 s'intéresse au problème du calcul de la similarité de deux graphes multi-étiquetés : nous présentons un algorithme glouton capable de calculer rapidement une approximation de la similarité de deux graphes, un algorithme Tabou, améliorant les résultats de cet algorithme glouton et enfin une version réactive (*i.e.,* s'autoparamétrant pendant la recherche) de l'algorithme Tabou. La partie 3 propose des résultats expérimentaux montrant visuellement l'intérêt de la mesure de similarité et comparant d'une part l'algorithme glouton et la recherche locale taboue, et d'autre part la recherche taboue par rapport à la recherche taboue réactive.

1 Graphes multi-étiquetés et similarité

1.1 Graphes et graphes multi-étiquetés

Un graphe orienté est un objet structuré en termes de composantes (appelées sommets), et de relations binaires entre ces composantes (appelées arcs). De façon plus formelle, **un graphe orienté** est défini par un couple $G = (V, E)$, où V est un ensemble fini de sommets et $E \subseteq V \times V$ est un ensemble d'arcs.

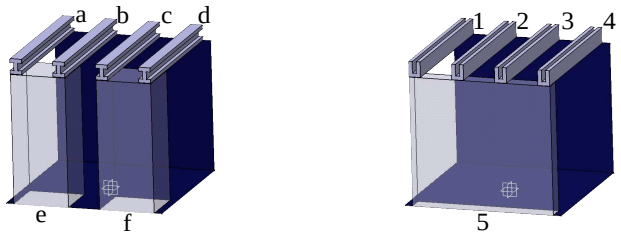
Les objets à représenter sont formés généralement par des composantes (resp. des relations binaires) ayant chacune une ou plusieurs propriétés. On s'intéresse donc plus particulièrement aux graphes multi-étiquetés [9, 8, 20],

i.e., des graphes où un ensemble (non vide) d'étiquettes est associé à chacun de leurs sommets et à chacun de leurs arcs. Chaque étiquette de sommets (resp. d'arcs) symbolise une propriété caractérisant une composante (resp. une relation entre deux composantes). Plus formellement, étant donnés un ensemble L_V d'étiquettes de sommets et un ensemble L_E d'étiquettes d'arcs, **un graphe orienté multi-étiqueté** est défini par un triplet $G = \langle V, r_V, r_E \rangle$ tel que :

- V est un ensemble fini de sommets,
- $r_V \subseteq V \times L_V$ est la relation associant sommets et étiquettes, *i.e.,* r_V est l'ensemble des couples (v_i, l) tels que le sommet v_i est étiqueté par l,
- $r_E \subseteq V \times V \times L_E$ est la relation associant arcs et étiquettes, *i.e.,* r_E est l'ensemble des triplets (v_i, v_j, l) tels que l'arc (v_i, v_j) est étiqueté par l. L'ensemble E des arcs peut être défini par $E = \{(v_i, v_j) | \exists l, (v_i, v_j, l) \in r_E\}$.

Nous appelons les tuples de r_V les caractéristiques des sommets de G et ceux de r_E les caractéristiques des arcs de G. On définit le descripteur d'un graphe $G = \langle V, r_V, r_E \rangle$ comme l'ensemble de toutes ses caractéristiques de sommet et d'arc : $descr(G) = r_V \cup r_E$. Cet ensemble décrit entièrement le graphe et sera utilisé pour mesurer la similarité de deux graphes.

1.2 Exemple



Objet 1 Objet 2

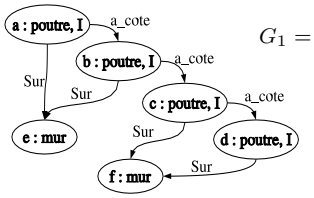
Figure 1: Deux objets de conception similaires

Afin d'illustrer les définitions de graphes multi-étiquetés et de similarité de graphes, voici un exemple issu d'une application de CAO [8]. La figure 1 présente deux objets similaires : les poutres a, b, c et d de l'objet 1 jouent respectivement le même rôle que les poutres 1, 2, 3 et 4 de l'objet 2, tandis que les murs e et f correspondent au mur 5. Ces deux objets sont similaires mais ne sont pas identiques : les poutres n'ont pas la même forme (en I à gauche et en U à droite) et le nombre de murs est différent (le mur 5 joue seul le rôle des murs e et f).

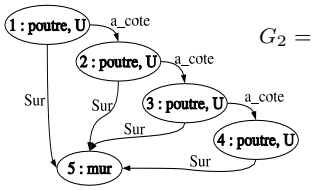
Afin de représenter les objets de la figure 1 par des graphes multi-étiquetés, définissons tout d'abord les ensembles d'étiquettes des sommets et des arcs :

$$L_V = \{ poutre, I, U, mur \}$$
$$L_E = \{ sur, a\_cote \}$$

Etant donnés ces ensembles d'étiquettes, les deux objets peuvent alors être représentés par les deux graphes multi-étiquetés de la figure 2.



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



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

Figure 2: Les graphes étiquetés $G1$ et $G2$ décrivant les objets 1 et 2 de la figure 1 : à gauche, une représentation graphique, à droite leur représentation sous la forme d'un triplet $\langle V, r_V, r_E \rangle$.

1.3 Similarité et appariements de graphes

On s'intéresse maintenant à définir une mesure de similarité entre deux graphes multi-étiquetés $G_1 = \langle V_1, r_{V_1}, r_{E_1} \rangle$ et $G_2 = \langle V_2, r_{V_2}, r_{E_2} \rangle$; on supposera que ces deux graphes sont définis sur le même ensemble d'étiquettes de sommets L_V et le même ensemble d'étiquettes d'arcs L_E et que leurs ensembles de sommets sont disjoints (*i.e.*, $V_1 \cap V_2 = \emptyset$).

1.3.1 Une mesure "générique" de la similarité

Afin de mesurer la similarité de deux objets, il est courant et assez intuitif de mettre en rapport les caractéristiques communes à ces deux objets avec l'ensemble de toutes leurs caractéristiques [17]. Tversky [21] définit la similarité de deux objets a et b, décrits respectivement par les deux ensembles A et B de leurs caractéristiques, par la formule [1] :

$$sim_{\text{Tversky}}(a, b) = \frac{f(A \cap B)}{f(A \cup B)}$$

[1] Il s'agit d'une version simplifiée de la formule (obtenue en mettant les deux paramètres α et β à 0) ; la version complète, permettant des mesures asymétriques est la suivante :

$$sim_{\text{Tversky}}(a, b) = \frac{f(A \cap B)}{f(A \cup B) - \alpha f(A - B) - \beta f(B - A)}$$

où f est une fonction positive monotone croissante par rapport à l'inclusion, permettant de pondérer l'importance des différentes caractéristiques. Lorsque la fonction f est définie comme la fonction cardinalité, cela revient à compter le nombre de caractéristiques communes par rapport au nombre total de caractéristiques.

1.3.2 Similarité de graphes et appariements

L'ensemble $descr(G)$ contient toutes les caractéristiques d'un graphe G. Par conséquent, il serait logique de comparer les deux graphes G_1 et G_2 en faisant l'intersection des ensembles $descr(G_1)$ et $descr(G_2)$ les décrivant. Le problème est que, comme $V_1 \cap V_2 = \emptyset$, les ensembles des descripteurs de deux graphes sont systématiquement disjoints ($descr(G_1) \cap descr(G_2) = \emptyset$). Pour pouvoir comparer deux graphes, il est donc nécessaire d'établir une correspondance (un *appariement*) entre les sommets de ces graphes mettant en évidence leurs points communs. Une mesure de similarité de deux graphes s'appuie alors sur un *meilleur appariement* de ces deux graphes.

Le meilleur appariement de deux graphes est souvent défini comme l'appariement identifiant le plus grand sous graphe commun aux deux graphes [7, 6, 5]. Le problème de ce type d'appariement est qu'il est nécessairement univoque : chaque sommet d'un graphe est associé à au plus un sommet de l'autre graphe. [9, 8, 20, 2] montrent que dans certaines applications, la comparaison est beaucoup plus pertinente si des appariement multivoques (*i.e.,* des appariements pouvant associer un sommet à plusieurs autres) sont utilisés.

Un appariement multivoque de deux graphes $G_1 = \langle V_1, r_{V_1}, r_{E_1} \rangle$ et $G_2 = \langle V_2, r_{V_2}, r_{E_2} \rangle$, tels que $V_1 \cap V_2 = \emptyset$, est une relation $m \subseteq V_1 \times V_2$, *i.e.,* un ensemble de couples de sommets. Un tel appariement associe à chaque sommet d'un graphe 0, 1 ou plusieurs sommets de l'autre graphe. Par extension, l'ensemble des sommets associés au sommet v dans l'appariement m est noté $m(v)$:

$$\begin{array}{lll} \forall v_1 \in V_1, & m(v_1) & \doteq \{v_2 \in V_2 |\ (v_1, v_2) \in m\} \\ \forall v_2 \in V_2, & m(v_2) & \doteq \{v_1 \in V_1 |\ (v_1, v_2) \in m\} \end{array}$$

Les caractéristiques communes $descr(G_1) \sqcap_m descr(G_2)$ à deux graphes G_1 et G_2 est alors définie par rapport à un appariement m de la façon suivante :

$$\begin{aligned} descr(G_1) \sqcap_m descr(G_2) \doteq & \{(v, l) \in r_{V_1} |\ \exists v' \in m(v),\ (v', l) \in r_{V_2}\} \\ \cup\ & \{(v, l) \in r_{V_2} |\ \exists v' \in m(v),\ (v', l) \in r_{V_1}\} \\ \cup\ & \{(v_i, v_j, l) \in r_{E_1} |\ \exists v'_i \in m(v_i), \exists v'_j \in m(v_j)\ (v'_i, v'_j, l) \in r_{E_2}\} \\ \cup\ & \{(v_i, v_j, l) \in r_{E_2} |\ \exists v'_i \in m(v_i), \exists v'_j \in m(v_j)\ (v'_i, v'_j, l) \in r_{E_1}\} \end{aligned}$$

Cet ensemble contient toutes les caractéristiques des éléments de G_1 (resp. G_2) retrouvées au moins une fois dans G_2 (resp. G_1) via l'appariement m, il est alors possible d'adapter la fomule de Tversky pour définir la similarité entre G_1 et G_2 par rapport à l'appariement m:

$$sim1_m(G_1, G_2) \quad = \quad \frac{f(descr(G_1) \sqcap_m descr(G_2))}{f(descr(G_1) \cup descr(G_2))} \qquad (1)$$

La définition de la similarité $sim1_m$ de l'équation 1 n'est pas entièrement satisfaisante dans le sens où elle ne tient pas compte du fait qu'un sommet peut être "éclaté", *i.e.,* apparié à plusieurs sommets. Par exemple, dans les deux graphes étiquetés de la figure 2, l'appariement $m_A = \{(a,1),(b,2),(c,3),(d,4),(e,5),(f,5)\}$ apparie le sommet 5 avec les deux sommets e et f. Pour prendre en compte ces éclatements de sommets, la fonction *splits* qui retourne l'ensemble des sommets "éclatés" avec l'ensemble des sommets auxquels ils sont reliés est introduite :

$$splits(m) \doteq \{(v, s_v) |\ v \in V_1 \cup V_2,\ s_v = m(v),\ |m(v)| \geq 2\}$$

et on définit un opérateur d'inclusion sur ces ensembles de sommets éclatés :

$$splits(m_1) \sqsubseteq splits(m_2) \Leftrightarrow \forall (v, s_v) \in splits(m_1),\ \exists (v, s'_v) \in splits(m_2),\ s_v \subseteq s'_v$$

L'équation (1) est alors modifiée pour prendre en compte ces ***splits*** :

$$sim_m(G_1, G_2) \quad = \quad \frac{f(descr(G_1) \sqcap_m descr(G_2)) - g(splits(m))}{f(descr(G_1) \cup descr(G_2))}$$

où la fonction g possède les mêmes propriétés que f : elle est positive et monotone croissante par rapport à la relation d'inclusion $\sqsubseteq$ de sorte que la similarité ne peut que décroître lorsque le nombre d'éclatements de sommets dans l'appariement m croît.

La ***similarité maximum*** $sim(G_1, G_2)$ de deux graphes G_1 et G_2 est la plus grande similarité par rapport à tous les appariements possibles :

$$sim(G_1, G_2) \quad = \quad \max_{m \subseteq V_1 \times V_2} \frac{f(descr(G_1) \sqcap_m descr(G_2)) - g(splits(m))}{f(descr(G_1) \cup descr(G_2))}$$

Notons que c'est le choix des fonctions f et g qui permet de déterminer le meilleur appariement. f et g sont choisies en fonction du domaine d'application. Dans la majorité des cas, f et g sont définies comme des fonctions de pondérations : f est la somme des poids des étiquettes retrouvées par l'appariement et g la somme des poids des splits générés par l'appariement.

1.4 Applications de cette similarité

La mesure de similarité présentée en section 1.3.2 est générique dans le sens où elle est entièrement paramétrable par le choix des fonctions f et g. En particulier, si on définit f par la fonction cardinalité et si g associe une valeur infinie à tous les *splits*, alors l'appariement maximisant $sim(G_1, G_2)$ correspond au plus grand graphe commun à G_1 et G_2. Dans cette section, nous illustrons l'intérêt d'utiliser une mesure de similarité de graphes basée sur des appariements multivoques à travers deux applications : la reconnaissance d'image et le raisonnement à partir de cas (RàPC) sur des objets de conception assistée par ordinateur (CAO).

Reconnaissance d'image. [2] utilise une mesure de similarité de graphes autorisant des appariements multivoques pour comparer des images. Les images sont représentées par des graphes : un sommet est associé à chaque région d'une image et les arcs décrivent les relations spatiales entre deux régions. Des étiquettes décrivant les textures des régions sont ajoutées aux sommets des graphes. La segmentation d'une image (*i.e.,* son découpage en régions) est une étape très complexe où de nombreuses erreurs peuvent survenir. Les deux erreurs les plus courantes sont la sur et la sous segmentation, *i.e.*, le fait de sur ou sous découper une image. Lors d'une comparaison de deux images, il est donc parfois nécessaire de faire correspondre une région d'une image avec plusieurs régions d'une autre image ce que seul l'utilisation d'appariements multivoques permet.
RàPC en CAO. L'exemple de la figure 1 est issu de [8] qui propose d'appliquer le raisonnement à partir de cas en conception assistée par ordinateur. Cette application nécessite de comparer des objets de conception représentés par des graphes multi-étiquetés. Comme le montre l'exemple de la figure 1, une composante (le mur du premier objet de conception) peut jouer le rôle de plusieurs composantes (les deux murs du second objet) et le meilleur appariement pour ces deux graphes consiste à apparier les sommets e et f au sommet 5.

2 Calcul de la similarité de deux graphes

Dans cette section, nous nous intéressons au problème du calcul de la similarité de deux graphes multi-étiquetés telle que définie en 1.3.2. Nous étudions en section 2.1 la complexité de ce problème et les limites des approches complètes pour la résolution de ce problème. Nous proposons ensuite trois algorithmes de recherche incomplète : un algorithme glouton, un algorithme de recherche locale guidée par une liste taboue et une version réactive de cet algorithme.

2.1 Complexité et approches complètes

Calculer la similarité de deux graphes consiste à trouver l'appariement m qui maximise la formule sim de la section 1.3.2. Le dénominateur de cette formule ne dépendant pas de l'appariement m, le problème revient à trouver l'appariement m qui maximise la fonction $score$ suivante :

$$score(m) = f(descr(G_1) \sqcap_m descr(G_2)) - g(splits(m))$$

L'espace de recherche du problème de la similarité maximum entre deux graphes étiquetés $G_1 = \langle V_1, r_{V_1}, r_{E_1} \rangle$ et $G_2 = \langle V_2, r_{V_2}, r_{E_2} \rangle$ est composé de tous les appariements possibles de ces deux graphes, *i.e.,* de tous les sous-ensembles de l'ensemble $V_1 \times V_2$. Il existe $2^{|V_1|*|V_2|}$ états possibles dans cet espace. Trouver le meilleur appariement multivoque entre deux graphes est un problème NP-difficile[2].

[2]Il a été montré en 1.4 qu'en choisissant correctement les fonctions f et g, mesurer la similarité entre G et G' permet de trouver le plus sous graphe commun à G et G'. Notre

[20, 9] proposent un algorithme de résolution du problème par séparation et évaluation (*"branch and bound"*). L'espace de recherche est structuré en arbre et est exploré de façon exhaustive. Une fonction d'évaluation de la qualité d'un ensemble d'appariements est utilisée pour élaguer les branches de l'arbre et accélérer ainsi la recherche. Le problème réside dans le fait qu'il n'existe pas de *bonne* fonction d'évaluation permettant un élagage efficace de l'arbre de recherche. En pratique, cette approche ne permet la comparaison que de très petits graphes (10 à 15 sommets pour les instances difficiles).

2.2 Algorithme glouton

Une approche gloutonne de résolution d'un problème consiste à choisir à chaque étape de l'élaboration d'une solution, l'élément de la solution semblant le plus intéressant au regard d'une heuristique. Les algorithmes gloutons, généralement de complexité faiblement polynomiale, permettent l'élaboration rapide de solutions de bonne qualité (mais sans garantie d'optimalité). [20, 9] proposent un algorithme glouton pour le problème de la recherche du meilleur appariement. L'algorithme démarre avec un appariement m vide et ajoute, à chaque itération, un couple de sommets dans m selon un principe glouton. Le couple à insérer est choisi parmi l'ensemble *cand* des couples qui maximisent la fonction *score*. Cet ensemble comportant généralement plus d'un candidat, les ex-æquo sont départagés en anticipant le potentiel des candidats : l'ensemble $look\_ahead$ des propriétés d'arcs (départs ou arrivées d'arcs) partagées par les deux sommets du couple candidat (u_1, u_2) et n'appartenant pas à $descr(G_1) \sqcap_{m \cup (u_1,u_2)} descr(G_2)$ est calculé de la façon suivante :

$$\begin{aligned} look\_ahead(u_1,u_2) \quad = \quad & \{(u_1,v_1,l) \in r_{E_1} \mid \exists v_2 \in V_2, (u_2,v_2,l) \in r_{E_2}\} \\ & \cup \{(u_2,v_2,l) \in r_{E_2} \mid \exists v_1 \in V_1, (u_1,v_1,l) \in r_{E_1}\} \\ & \cup \{(v_1,u_1,l) \in r_{E_1} \mid \exists v_2 \in V_2, (v_2,u_2,l) \in r_{E_2}\} \\ & \cup \{(v_2,u_2,l) \in r_{E_2} \mid \exists v_1 \in V_1, (v_1,u_1,l) \in r_{E_1}\} \\ & - descr(G_1) \sqcap_{m \cup \{(u_1,u_2)\}} descr(G_2) \end{aligned}$$

Le prochain sommet à insérer est alors choisi aléatoirement parmi ceux qui maximisent $f(look\_ahead)$.

2.3 Recherche locale Tabou

Recherche locale. L'algorithme glouton décrit en 2.2 retourne un appariement "localement optimal" dans le sens où ajouter ou supprimer *un seul* couple de sommets à cet appariement ne peut pas améliorer la qualité de celui-ci. Cela ne signifie pas pour autant qu'il n'existe pas d'appariement meilleur (induisant une plus grande similarité) qu'il serait possible d'obtenir en ajoutant et/ou en supprimant *plusieurs* couples de sommets à cet appariement.

problème, plus général qu'un problème NP-difficile et est donc NP-difficile.

Le principe d'une recherche locale [12, 16] se base sur le fait que la solution optimale d'un problème fortement combinatoire peut ne pas être *éloignée* d'une bonne solution. L'espace de recherche est tout d'abord structuré en terme de voisinage : les voisins d'un état e sont définis comme les états pouvant être obtenus en appliquant une transformation élémentaire à e. Dans notre cas, l'espace de recherche est composé de tous les appariements, *i.e.,* l'ensemble $\wp(V_1 \times V_2)$. Les transformations élémentaires définissant le voisinage d'un appariement m sont le retrait d'un couple de sommets à m ou l'ajout d'un nouveau couple de sommets à m. L'ensemble des voisins d'un appariement m est donc défini par :

$$\begin{aligned} \forall m \in \wp(V_1 \times V_2), voisinage(m) &= \{m \cup \{(u,v)\}/(u,v) \in (V_1 \times V_2) - m\} \\ &\cup \{m - \{(u,v)\}/(u,v) \in m\} \end{aligned}$$

A partir d'un état initial de bonne qualité (obtenu par un algorithme glouton), l'espace de recherche est exploré de proche en proche jusqu'à l'obtention de la solution optimale du problème (lorsque la qualité de celle-ci est connue) ou jusqu'à ce que le nombre maximum d'itérations soit atteint. L'heuristique sélectionnant à chaque étape le prochain voisin d'un état distingue les différentes recherches locales existantes.

Heuristique Tabou. Une des meilleures heuristiques connues pour choisir le prochain voisin est la méthode Tabou [12, 10, 18]. Elle consiste à choisir à chaque itération le meilleur voisin mais, afin de diversifier la recherche (*i.e.,* éviter de *tourner en rond* lorsqu'on se trouve sur des maxima locaux), une liste taboue est introduite. Cette liste, de longueur finie k, mémorise les k dernières transformations élémentaires appliquées sur l'état courant (*i.e.,* les k derniers couples de sommets ajoutés ou supprimés). Durant un nombre d'itérations égal à k, il est interdit d'effectuer une transformation inverse à une transformation récemment effectuée (*i.e.,* ajouter un couple récemment supprimé ou supprimer un couple récemment ajouté). Une exception au principe Tabou, appelée *aspiration*, est souvent ajoutée. Si une transformation interdite par la liste Taboue permet d'obtenir une solution de qualité supérieure à la meilleure solution rencontrée depuis le début de la recherche, l'interdiction d'exécuter cette transformation est levée et la transformation est exécutée. La méthode Tabou guide la recherche vers de bonnes solutions (en choisissant les meilleurs voisins) tout en permettant une diversification de la recherche en évitant de *"tourner en rond"* (grâce à la liste taboue).

Détail de l'algorithme. La figure 3 décrit le schéma général de l'algorithme Tabou. On commence par construire un premier appariement m avec l'algorithme glouton décrit en 2.2 puis, à chaque itération, on sélectionne un appariement m' dans le voisinage de m. Cette sélection s'effectue en deux temps : tout d'abord, on construit l'ensemble *candidats* des voisins de m dont le *score* (défini en section 2.1) est supérieur au meilleur appariement rencontré au cours de la recherche (c'est le critère d'aspiration) ; si *candidats* est vide, alors on définit *candidats* comme l'ensemble des voisins non Tabous de m. La deuxième étape de cette sélection consiste à choisir aléatoirement un appariement

de l'ensemble *candidats* parmi ceux qui maximisent la fonction *score* puis la fonction *look_ahead* (définie en section 2.2).

fonction Tabou($G_1 = \langle V_1, r_{V1}, r_{E1}\rangle, \langle V_2, r_{V2}, r_{E2}\rangle, f, g, nbMaxIter, tabuSize$)
retourne un appariement $m \in \wp(V_1 \times V_2)$
  $m \leftarrow Glouton(G_1, G_2, f, g)$
  $best_m \leftarrow m$
  $nbIter \leftarrow 0$
  tant que $score(m) < score(descr(G_1) \cup descr(G_2))$ et
      $nbIter < nbMaxIter$ **faire**
    / test du critère aspiration */*
    $candidats \leftarrow \{m' \in voisinage(m) / score(m') > score(best_m)\}$
    si $candidats = \emptyset$ **alors**/* *pas d'aspiration* */
      $candidats \leftarrow \{m' \in voisinage(m) /$ non $estTabou(m, m', tabouSize)$
    fin si
    $candidats \leftarrow \{m' \in candidats / score(m')$ puis $look\_ahead(m')$ maximaux$\}$
    choisir aléatoirement $m' \in candidats$
    $rendreTabou(m, m', tabouSize)$
    $m \leftarrow m'$
    si $score(m') > score(best_m)$ **alors** $best_m \leftarrow m'$ **fin si**
    $nbIter \leftarrow nbIter + 1$
  fin tant que
  retourner $best_m$

Figure 3: Algorithme Tabou de recherche du meilleur appariement.

2.4 Recherche Taboue réactive

Un paramètre important dans une recherche de type Taboue est la longueur de la liste : elle permet de réguler la diversification de la recherche par rapport à son intensification. Si la liste est trop longue, la diversification est trop forte, l'algorithme convergera trop doucement et ne trouvera pas de bonnes solutions en un temps raisonnable. A contrario, si la liste est trop courte, l'intensification est trop forte, l'algorithme ne pourra pas *sortir* des maxima locaux, il *tournera en rond* et ne sera plus capable d'améliorer la solution courante.

La longueur optimale de la liste taboue est délicate à régler (voir la section 3). Afin de remédier à ce problème de paramétrage, [4] introduit l'idée d'une recherche taboue réactive, où la longueur de la liste taboue s'auto-adapte dynamiquement : lorsque la recherche semble se cantonner autour d'un maximum local, la longueur est augmentée afin de permettre de s'*échapper* de ce sous-espace de recherche ; a contrario, quand l'exploration de l'espace de recherche semble efficace, la longueur de la liste est réduite afin de ne pas contraindre trop fortement la recherche et ainsi explorer tous les maxima locaux rencontrés.

Rendre réactif un algorithme Tabou nécessite d'évaluer le besoin de diversification de la recherche. Lorsque l'algorithme retombe sur un appariement

déjà visité auparavant, il devient nécessaire de diversifier la recherche. Afin de détecter de telles redondances lors de l'exploration de l'espace de recherche on mémorise les clés de hachage correspondant aux appariements visités. Une collision dans la table de hachage est synonyme d'un manque de diversification et il devient alors nécessaire d'allonger la liste taboue. A contrario, lorsqu'il n'y a pas eu de collision depuis un certain nombre d'itérations de l'algorithme, la diversification est suffisante, et il devient alors possible de diminuer la longueur de la liste. Le calcul du code de hachage d'un appariement est incrémental : les coûts relatifs à la mémorisation des états sont donc négligeables.

L'algorithme Tabou réactif nécessite un plus grand nombre de paramètres que sa version non réactive. Il faut en effet définir la longueur de la liste au début de la recherche, définir de combien la liste doit être allongée et de combien elle doit être réduite lors d'une mise à jour de sa longueur ainsi que définir les longueurs extrêmes (minimum et maximum) de cette liste. Des expérimentations sur l'algorithme Tabou non réactif permettent de définir certains de ces paramètres : les bornes de la longueur de la liste sont obtenues en mesurant l'efficacité de Tabou sur des tailles de listes extrêmes. Les pas d'incrémentation et de décrémentation de la longueur de la liste sont obtenus expérimentalement, la longueur initiale de liste est toujours fixée au minimum.

3 Résultats expérimentaux

Benchmark. Il n'existe pas à notre connaissance de benchmark de problèmes de comparaison de graphes autorisant les fusions et les éclatements de sommets. Afin de tester les algorithmes présentés en section 2, nous avons utilisé un générateur aléatoire de paires de graphes similaires. Cet outil génère un premier graphe puis lui applique une suite de transformations (fusions et éclatements de sommets, ajouts et suppressions d'arcs, de sommets et d'étiquettes) afin d'obtenir un deuxième graphe "similaire". Afin d'évaluer la qualité des résultats trouvés par nos algorithmes, une borne inférieure de la similarité des deux graphes est calculée à partir de la suite de transformations appliquées.
La comparaison de deux graphes est simple lorsque les éléments des graphes sont fortement étiquetés : peu de couples de sommet sont similaires, et la solution est alors triviale. Pour tester nos algorithmes sur des instances difficiles, nous avons donc générés des graphes de 80 à 100 sommets et de 200 à 360 arcs ayant tous la même étiquette. Chaque suite de transformations est composée de 5 fusions ou éclatements de sommets et de 10 ajouts ou suppressions d'arcs ou de sommets. Sur 100 problèmes générés, une cinquantaine sont simples à résoudre (*i.e.,* l'algorithme glouton trouve souvent la solution et une recherche locale trouve tout le temps la solution optimale en un petit nombre d'itérations), 35 instances semblent plus difficiles (*i.e.,* glouton peine à trouver la solution) et 14 instances sont très difficiles (*i.e.,* glouton ne peut pas trouver la solution et la recherche locale peine à la trouver).
Réglage des paramètres. De nombreuses expérimentations ont été effectuées afin de déterminer les paramètres optimaux des recherches locales. Ces tests

montrent que la recherche Tabou non réactive est efficace lorsque la liste taboue a une longueur lgL comprise entre 10 et 20. Pour Tabou réactif, la longueur minimale de la liste $lMin$ doit être autour de 10, le pas Pas d'incrémentation ou décrémentation de la liste (*i.e.,* l'allongement de la liste lorsque la recherche n'est pas suffisemment diversifiée ou la réduction de la liste lorsque la recherche est suffisamment diversifiée) doit avoir une valeur proche de 10 ou 15. La fréquence de décrémentation $fDec$ de la liste, *i.e.,* le nombre d'itérations attendues avant de raccourcir la liste quand la recherche est correctement diversifiée, doit être fixée à 1000 itérations. Enfin, la longueur maximum $lMax$ de la liste doit être fixée à 50.

Glouton vs. recherche locale. Quelques itérations de l'algorithme glouton (une demi-seconde environ par itération pour les problèmes étudiés) suffisent à résoudre les instances faciles ou à trouver une très bonne approximation de la similarité des deux graphes. Les instances plus difficiles sont rarement résolues par Glouton alors que quelques itérations d'un algorithme Tabou suffisent à trouver la meilleure solution. Enfin, l'algorithme glouton ne trouve que de très mauvaises solutions pour les instances les plus difficiles : la similarité de deux graphes identiques à 95% est parfois estimée à moins de 50%!

Recherche locale Tabou. 500 itérations de recherche locale (effectuées en 3 ou 4 secondes sur les problèmes étudiés) suffisent à trouver la solution optimale de la majorité des problèmes. Une dizaine de milliers d'itérations de Tabou permet de résoudre presque toutes les instances. La longueur de la liste taboue est cependant un paramètre très important : si elle est trop courte (moins de 10), l'algorithme ne sort d'aucun maximum local et ne peut alors pas trouver la solution, si elle est trop longue (plus de 20), l'algorithme explore de façon quasi aléatoire l'espace de recherche et ne trouve pas de bonnes solutions en un temps raisonnable. Notons que le résultat d'une recherche locale est meilleur lorsque la solution de départ est de bonne qualité : il est préférable de commencer la recherche locale à partir du meilleur appariement trouvé lors de plusieurs exécutions de Glouton.

Pour être résolues, les instances les plus difficiles nécessitent un grand nombre d'itérations (plusieurs dizaines de milliers) et un paramétrage précis de la longueur de la liste Tabou. La tableau 1 résume les résultats moyens de 10 exécutions de Tabou (non réactif) sur les 14 instances les plus difficiles. Le nombre maximum d'itérations a été fixé à 25000 et est réévalué à $max(25000, 2*t)$ lorsque l'algorithme trouve une meilleure solution au temps t. La colonne lgL précise la longueur de liste taboue utilisée, $TReussi$ précise le pourcentage d'exécutions qui ont trouvé la borne minimum (ou une valeur meilleure), $TpsMoy$ précise le temps d'exécution moyen en secondes pour trouver la meilleure solution, $TpsET$ précise l'écart type de ces temps d'exécution et $nbIter$ le nombre moyen d'itérations effectuées.

Le tableau 1 montre que la longueur de la liste taboue est un paramètre très difficile à régler : les résultats obtenus avec une liste de taille 10 sont très différents de ceux obtenus avec une taille 12 (34% de problèmes résolus avec une taille 10 contre 54% avec une taille 12). D'une façon générale, les instances difficiles nécessitent une liste taboue plus longue mais nous avons cependant

lgL	TReussi	TpsMoy	TpsET	nbIter
10	34%	265	250	46056
12	54%	281	316	50396
16	63%	250	251	43913
20	64%	263	271	45766

Table 1: Résultats de Tabou sur les instances les plus difficiles

limité la longueur de la liste à 20 car, au delà, ce sont les instances plus simples qui ne sont pas résolues en un temps raisonnable. Une instance sur les 14 instances considérées n'a jamais été résolues par aucun des algorithmes Tabou.

Recherche Tabou reactive. Tabou et Tabou réactif résolvent les instances moyennement difficiles (*i.e.,* souvent résolues en moins de 10000 mouvements de Tabou) de façon similaires. En effet, en 10000 itérations, l'algorithme ne boucle presque jamais, l'effet de réaction ne se déclenche pas et la version réactive a le même comportement que la version non réactive de Tabou.

La tableau 2 résume les résultats moyens de 10 exécutions de Tabou réactif sur les 14 instances les plus difficiles. Comme pour Tabou, le nombre maximum d'itérations a été fixé à 25000 et est réévalué à $max(25000, 2 * t)$ lorsque l'algorithme trouve une meilleure solution au temps t. La colonne $lMin$ précise la longueur minimale de le liste, Pas précise le pas d'incrémentation et de décrémentation de la liste, $fDec$ la fréquence de décrémentation. La longueur maximum $lMax$ de la liste à toujours été fixée à 50.

lMin	Pas	fDec	TReussi	TpsMoy	TpsET	nbIter
10	10	1000	76%	233	116	44141
12	10	1000	73%	208	106	38873
10	15	1000	79%	218	115	41174

Table 2: Résultats de Tabou réactif sur les instances les plus difficiles

Le tableau 2 montre que la version réactive de Tabou donne de meilleurs résultats que sa version non réactive : le meilleur Tabou (longueur de liste 20) obtient 64% de réussite alors que les résultats de Tabou réactif sont tous supérieurs à 73% de réussite en un temps plus court (233s au pire contre 263s). L'algorithme réactif semble aussi plus robuste : quelque soit le paramétrage, les résultats obtenus sont relativement proches. De plus, les écarts types des temps d'exécution pour la version réactive sont bien moindre que ceux de la version non réactive. Les paramètres optimaux de Tabou réactif sont les mêmes quelque soit la difficulté de l'instance : les problèmes simples sont résolus avec les mêmes paramètres que les problèmes difficiles alors que Tabou donne de meilleurs résultats avec une liste courte (10) sur les instances faciles et avec une liste longue (20) pour les instances difficiles. Enfin, il est intéressant de noter que l'instance la plus difficile, celle qui n'a jamais été résolue par la version non réactive de Tabou, a été résolue plusieurs fois par chacun des algorithmes Tabou réactif.

Tabou réactif est donc plus efficace et plus simple à paramétrer que sa version non réactive même s'il comporte plus de paramètres. La recherche locale taboue réactive n'a cependant d'intérêt que pour les instances les plus difficiles du

problème de la mesure de similarité de graphes multi-étiquetés car les instances plus simples sont résolues trop rapidement pour que la réaction s'enclenche.

Conclusion

Dans cet article, nous introduisons une mesure de similarité de graphes ainsi que des algorithmes permettant le calcul de cette similarité. L'apport de ce travail par rapport aux approches existantes concerne les points suivants :

- La mesure se similarité est basée sur la notion d'*appariement multivoque*, ce qui permet de prendre en compte le fait qu'un composant d'un objet puisse jouer le même rôle que plusieurs composants d'un autre objet.
- Les algorithmes que nous proposons pour mesurer la similarité de graphes sont de complexité polynomiale et permettent de mesurer la similarité de graphes comportant une centaine de sommets en un temps de l'ordre de la minute. En contrepartie, nos algorithmes ne garantissent pas l'optimalité : ils peuvent, dans certains cas, retourner une valeur de similarité inférieure à la valeur optimale. Les expérimentations montrent cependant que, si l'algorithme glouton ne trouve généralement pas la solution optimale des instances difficiles du problème, la recherche locale trouve bien souvent la solution optimale.
- L'utilisation d'une liste taboue *réactive*, *i.e.,* capable d'auto-ajuster sa longueur au cours de la recherche, nous a permis d'améliorer les résultats obtenus par une recherche locale non réactive.

Ces travaux ouvrent de nombreuses perspectives : nous aimerions dans un premier temps, en collaboration avec des spécialistes du traitement de l'image, améliorer l'application de recherche d'images similaires afin de pouvoir utiliser des images plus réalistes. Nous aimerions ensuite utiliser notre mesure de similarité dans d'autres applications, en particulier pour la recherche de réponses approximatives à des requêtes sur des bases de données semi-structurées. Un autre de nos objectifs est l'amélioration de l'algorithme Tabou réactif. Il s'agira de proposer d'une part une méthode permettant de détecter plus rapidement que la recherche tourne autour d'un minimum local en utilisant la méthode de comparaison des états de l'espace de recherche proposée par [19] et d'autre part de rendre l'algorithme plus robuste en faisant en sorte que les valeurs paramètrant la réactivité (pas d'incrémentation/décrémentation et fréquence de décrémentation de la longueur de la liste) puissent, tout comme la longueur de la liste taboue, s'auto-paramétrer.

References

[1] T. Akutsu. Protein structure alignment using a graph matching technique, 1995.

[2] R. Ambauen, S. Fischer, and H. Bunke. Graph Edit Distance with Node Splitting and Merging, and Its Application to Diatom Identification. In *IAPR-TC15 Workshop on Graph-based Representation in Pattern Recognition*, pages 95–106, 2003.

[3] R. Baeza-Yates and G. Valiente. An image similarity measure based on graph matching. In *Proceedings of 7th Int. Symp. String Processing and Information Retrieval*, pages 28–38. IEEE Computer Science Press, 2000.

[4] R. Battiti and M. Protasi. Reactive local search for the maximum clique problem. In Inc. Springer-Verlag New York, editor, *Algorithmica*, volume 29, pages 610–637, 2001.

[5] H. Bunke. On a relation between graph edit distance and maximum common subgraph. *PRL: Pattern Recognition Letters*, 18, 1997.

[6] H. Bunke. Graph matching : Theoretical foundations, algorithms, and applications. In *Proc. Vision Interface 2000, Montreal*, pages 82–88, 2000.

[7] H. Bunke and X. Jiang. *Graph Matching and Similarity*, volume Teodorescu, H.-N., Mlynek, D., Kandel, A., Zimmermann, H.-J. (eds.): Intelligent Systems and Interfaces, chapter 1. Kluwer Academic Publishers, 2000.

[8] P.-A. Champin. *Modéliser l'expérience pour en assister la réutilisation : de la Conception Assistée par Ordinateur au Web Sémantique.* Thèse de doctorat en informatique, Université Claude Bernard, Lyon (FR), 2002.

[9] P.-A. Champin and C. Solnon. Measuring the similarity of labeled graphs. In *5th International Conference on Case-Based Reasoning (ICCBR 2003)*, volume Lecture Notes in Artificial Intelligence Number 2689 - Springer-Verlag, pages 80–95, 2003.

[10] R. Dorne and J. Hao. *Tabu Search for graph coloring, T-coloring and Set T-colorings*, chapter 3. I.H. Osman et al. (Eds.), Kluver Academic Publishers, 1998.

[11] E. J. Gardiner, P. J. Artymiuk, and P. Willett. Clique-detection algorithms for matching three-dimensional molecular structures. *Journal of Molecular Graphics and Modelling 15*, pages 245–253, 1997.

[12] F. Glover. Tabu search - part I. *Journal on Computing*, pages 190–260, 1989.

[13] Y.-G. Guéhéneuc and N. Jussien. Quelques explications pour les patrons – Une application de la PPC avec explications pour l'identification de patrons de conception. In Bertrand Neveu, editor, *Actes des 7e JNPC*, pages 111–122. ONERA, juin 2001.

[14] A. Hlaoui and S. Wang. A new algorithm for graph matching with application to content-based image retrieval. *Lecture Notes in Computer Science*, Volume 2396, 2002.

[15] L. Holm and C. Sander. Mapping the protein universe. *Science 273*, pages 595–602, 1996.

[16] S. Kirkpatrick, S. Gelatt, and M. Vecchi. Optimisation by simulated annealing. In *Science*, volume 220, pages 671–680, 1983.

[17] D. Lin. An Information-Theoretic Definition of Similarity. In *proceeding of ICML 1998, 15th International Conference on Machine Learning*, pages 296–304. M. Kaufmann, 1998.

[18] S. Petrovic, G. Kendall, and Y. Yang. A Tabu Search Approach for Graph-Structured Case Retrieval. In *STAIRS 2002*, pages 55–64, 2002.

[19] A. Sidaner, O. Bailleux, and J.-J. Chabrier. Measuring the spatial dispersion of evolutionist search processes : application to walksat. In *Artificial Evolution*, pages 77–90, 2001.

[20] S. Sorlin, P.-A. Champin, and C. Solnon. Mesurer la similarité de graphes étiquetés. In *9ème Journées Nat. sur la résolution pratique de problèmes NP-Complet (JNPC)*, 2003.

[21] A. Tversky. Features of Similarity. In *Psychological Review*, volume 84, pages 327–352. American Psychological Association Inc., 1977.