الجمهورية الجزائرية الديمقراطية الشعبية

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

وزارة التعليم العالي و البحث العلمي

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

Université Dr. Tahar Moulay SAIDA — جامعة د الطاهر مولاي سعيدة

Faculté : Technologie — كلية التكنولوجيا

Département : Informatique — قسم : الإعلام

MEMOIRE DE MASTER

Option : MICR

THEME

Etude comparative des méthodes d'appariement de graphes (Graph Matching)

Encadré par : Mr.Rahmani Mohamed

Présenté par : Mlle. Allam Nacera

Remerciements

J'aimerais remercier en premier lieu ma famille et surtout ma mère ma chère qui m'ont soutenue pendant cette année de thèse et mon père. Je remercie très chaleureusement à Mr. HANOUNE MOHAMED pour ses propositions et son aide avec lesquels mon travail à avancer plus rapidement.

***M**erci Dieu le tout puissant pour j'avoir donné la foi et le courage pour accomplir mon travail.*

***Je** tiens à remercier mon professeur et superviseur Mr. RAHMANI MOHAMED pour la confiance qu'il j'ai accordé, les conseils qu'il j'ai donné, l'aide qu'il j'ai apporté et le suivi qu'il a effectué.*

***J'**exprime mon profonde gratitude envers les membres du jury, pour l'intérêt qu'ils ont bien voulu porter à ce projet en acceptant de participer à ce jury.*

***Je** remercie sincèrement tous les enseignants de mon département.*

***C**es remerciements ne seraient pas complets si je ne les associe à ma famille qui j'ai soutenu et supporté pendant ces années d'études.*

***Je** tiens aussi à exprimer mon vif remerciement à tous ceux qui ont contribué au bon déroulement de ce projet.*

Enfin, aux étudiants de notre promotion.

Nesrine

Dédicaces

Je dédie ce modeste travail en priorité à

Mr. HANOUNE MOHAMED Qui m'a toujours assisté durant mes études

Et pendant toutes les phases de réalisation de ce mémoire.

A mes parents toujours présents dans mon cœur

Et qui m'ont soutenue le long de mes études.

A mes charmantes sœurs aicha et fatoma et son époux houari

Et khawla et houria.

A mes frères Nourdine et Boubakra.

A ma Tantes : Souad, mokhtaria.fatma.houria.

Enfin, à tous mes amis (ies) de l'université Moulay Tahar :.(surtout faiza et Nadia et taousse .

MERCI !!!

Nacèra Allam !!!!!!

Tables Des matières

Listes des Figures

INTRODUCTION GENERALE

Ce rapport présente le travail effectué durant notre Projet de Fin d'étude, il a été consacré a l'appariement exacte de graphes et sous graphes. Un état de l'art sur la théorie des graphes et l'isomorphisme de graphes était indispensable pour s'engager dans ce sujet.

Notre mémoire est composé de Quatre chapitres:

Le premier chapitre

Dans cette partie, nous présentons la notion de graphe et nous caractérisons d'abord les différentes classes de graphes afin d'établir la définition formelle du problème traité dans ce mémoire. Les définitions qui suivent sont reproduites principalement du livre Graph Theory and its applications.

.

Le deuxiéme chapitre

- Dans ce chapitre , nous présentons les deferent types des méthodes de comparaison de graphes

•

- Le problème de l'appariement de graphes est de trouver une correspondance entre les sommets d'un graphe et les sommets d'un autre graphe qui satisfasse à certaines contraintes ou critères d'optimalité

Il y a deux grandes catégories de méthodes d'appariement de graphes : les méthodes qui fonctionnent selon une approche symbolique (comme par exemple la recherche d'isomorphisme exact entre graphes ou entre sous-graphes) et les méthodes de comparaison de graphes avec tolérance d'erreur plutôt adaptées aux approches numériques. La plupart du temps, les méthodes symboliques utilisent soit un arbre de recherche avec retour-arrières **«** *backtracking* », soient les matrices d'adjacence des graphes. La deuxième catégorie inclut les méthodes dans lesquelles une fonction d'énergie est minimisée avec quelques heuristiques pour arriver à la solution. Tous les algorithmes d'appariement de graphes de la première

catégorie sont assez performants en termes de résultats fournis mais partagent le même problème d'explosion combinatoire de l'espace de recherche (un problème *NP*-complet au pire). La deuxième catégorie propose des solutions plus approximatives mais en des temps plus raisonnables. **[16]**

Le troixiéme chapitre

Dans cette partie, nous présentons la complexité du problème d'isomorphisme de graphes n'est pas exactement connue : ce problème est **NP** mais on ne sait pas s'il est dans **P** ou s'il est **NP-complet** Cependant, avec certaines topologies particulières de graphes (graphes planaires , des arbres ou graphes de valence bornée , ce problème peut être résolu dans un temps polynomial.

IL existe certains algorithmes pour résoudre le problème d'appariement de graphe tel que Ullmann, Schmidt and Druffel (SD), VF, VF2, et Nauty. Dans ce chapitre mais détaillant algorithme Ullmann ainsi que VF2

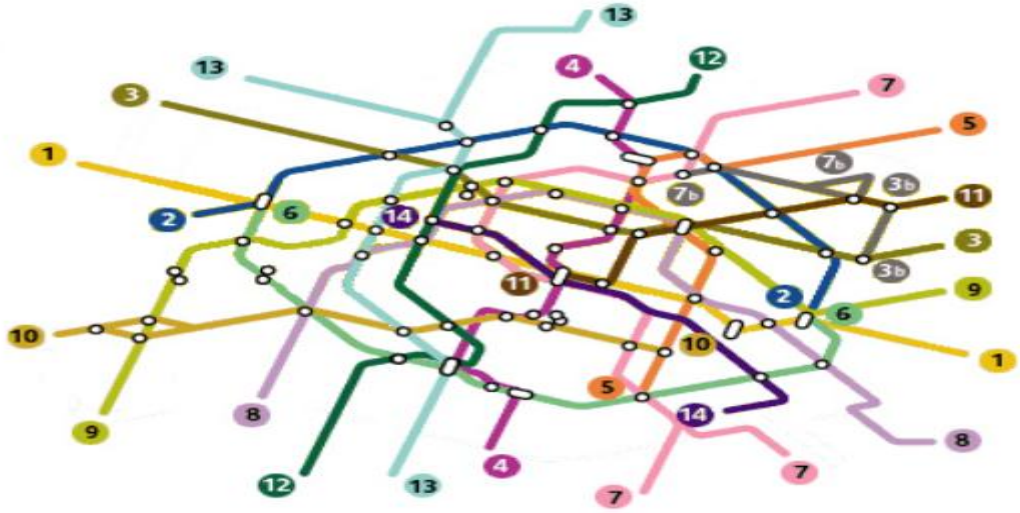
L'algorithme d'Ullmann cherche à trouver tous les isomorphismes de sous-graphes possibles du graphe $\boldsymbol{G_D}$ à partir du graphe $\boldsymbol{G_M}$.

Le quatriéme chapitre

- Ce dernier chapitre présente notre implémentation étude comparative des deux méthodes (Ullmann, VF2) à travers un qu'un ensemble de tests sur différents graphe l'origine et sous-graphe, tout en discutant et commentant les résultats obtenus.

Chapitre I

Etat de l'art sur la théorie des graphes



Sommaire

Introduction :

Les graphes sont des outils mathématiques utilisés dans de nombreux domaines pour modéliser les problèmes à résoudre. Depuis, la théorie des graphes a donné naissance à de multiples techniques dans de nombreux domaines, en mathématique, automatique, algorithmique, biologie, chimie, etc. et plus récemment en reconnaissance des formes.

Cette théorie a deux principaux objectifs. Le premier est d'offrir un modèle pour représenter le problème, généralement l'ensemble des possibilités. Dans notre contexte, le problème est la mise en correspondance d'éléments dans deux ensembles différents. Dans ce cas,le graphe est une représentation des différents éléments et de leurs relations binaires entre eux. Le deuxième objectif de cette théorie est d'offrir des outils pour permettre de trouver les meilleures possibilités dans le but de résoudre un problème. Dans notre contexte, nous nous intéressons aux outils qui permettent de comparer deux ensembles de possibilités (i.e. les graphes).

Dans ce chapitre nous introduisons quelques notions sommaires de la théorie des graphes qui vont nous être utiles dans les chapitres suivants. Pour plus de détails nous nous référons à **[22]** et **[23].**

I-1. Généralités sur les graphes

Le plus souvent, les graphes sont un moyen commode pour représenter, modéliser, visualiser certaines situations et problèmes, et pour contribuer à les résoudre. La théorie des graphes est néanmoins une théorie autonome, dans la mesure où elle a ses propres problèmes, et où elle a su développer des outils et concepts, qui lui sont propres, pour les résoudre. Ces instruments fondamentaux ont de multiples applications dans des domaines très divers. L'exemple le plus classique est la représentation d'un réseau de communication : réseau de routes représenté par une carte routière, réseau de chemin de fer, de téléphone ou de relais de télévision, réseaux électriques, réseaux d'informations dans une organisation, etc…

Définition 1.1

Un graphe orienté $G = (N, A)$ est déterminé par un ensemble N non vide dont les éléments sont appelés sommets ou nœuds et un ensemble A dont les éléments sont des couples ordonnés de sommets appelés arcs. Les sommets seront numérotés $1,....,n$ (on dit alors que G est d'ordre n), et les arcs $1....m$.

Définition 1.2

Etant donné un graphe $G = (N, A)$, pour $X \subset N$, nous désignons par $\omega(X)$ l'ensemble des arcs ayant leur extrémité initiale dans X et leur extrémité terminale dans $(N-X)$. Nous avons $\omega^-(X) = \omega(N-X)$.

Définition 1.3

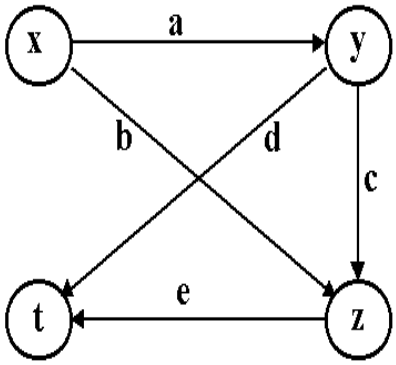
Le degré d'un sommet $x \in N$ est l'entier $d(x) = |\omega(x)| + |\omega^-(x)|$.

Définition 1.4

La matrice d'incidence sommets-arcs d'un graphe G est la matrice

$M = (m_{ij})$ de taille n×m, définie par (figure I. 1):

$$m_{ij} = \begin{cases} 1 \text{ si } i \text{ est le sommet initoal de } j, \\ -1 \text{ si } i \text{ est le sommet terminal de } j, \\ 0 \text{ dans les autres cas.} \end{cases}$$



Représentation sagittale

	a	b	c	d	e
x	+1	+1	0	0	0
y	-1	0	+1	+1	0
z	0	-1	-1	0	+1
t	0	0	0	-1	-1

Représentation par une matrice d'incidence sommets-arcs.

FIG. I.1 - Représentation d'un graphe simple orienté.

Définition 1.5

Une arête est un couple non ordonné de nœuds. Tandis que pour un arc l'ordre du couple de nœuds qui le définit est important [$(x, y) \neq (y, x)$].

Définition 1.6

Une chaîne dans G est une suite d'arcs $x_1, \ldots, x_p$ telle que chaque arc u_i ($1 \leq i \leq p-1$) a une extrémité commune avec l'arc u_{l+1}. Une chaîne est simple si elle ne comporte pas plusieurs fois le même arc dans la suite qui la construit. On dit qu'une chaîne est élémentaire si tous les sommets sont au plus de degré 2.

Définition 1.7

Un cycle est une chaîne dont les extrémités coïncident. Un cycle est dit élémentaire s'il ne contient strictement aucun autre cycle.

Définition 1.8

Un chemin est une chaîne dont tous les arcs sont orientés dans le même sens. Un chemin élémentaire est un chemin dont tous les sommets sont au plus de degré 2.

Définition 1.9

Un circuit est un chemin dont les extrémités coïncident. Un circuit élémentaire est un circuit ne contenant strictement aucun autre circuit.

Définition 1.10

Un graphe *G=(N,A)* est dit complet si, pour toute paire de nœuds *(x, y)*, il existe au moins un arc de la forme *(x, y)* ou *(y, x)*.

Définition 1.11

Un graphe est dit connexe si, pour tout couple de sommets *x* et *y*, il existe une chaîne joignant *x* et *y*.

Définition 1.12

Un graphe est dit fortement connexe si, étant donnés deux sommets quelconques *x* et *y* (dans cet ordre), il existe un chemin d'extrémité initiale x et d'extrémité terminale *y*.

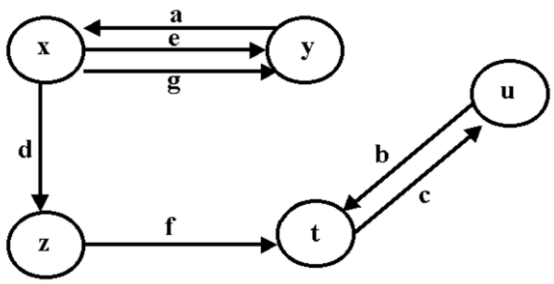
Définition 1.13

Un multi-graphe est un couple *G = (N,A)* dans lequel *N* est un ensemble de sommets, et *A* une famille d'arcs $A = (u_i, u_2, ... u_m)$. Chaque arc est défini comme un couple de sommets *(x, y)*, qui sont respectivement son origine et son extrémité.

Plus précisément, on pourrait définir la famille d'arcs par deux applications, *f* et g de *{1,2, ... m}* dans *N*, telles que pour chaque *i*, on ait *f*(i) =origine (u_i), et g(i):= extrémité (u_i).

On voit que cette définition permet de traiter des graphes dont plusieurs arcs auraient la même origine et la même extrémité. D'où le nom de multigraphe.

On dit qu'un multi-graphe est un *p*-graphe, si quels que soient les sommets *x, y*, ce graphe a au plus *p* arcs d'origine *x*, et l'extrémité *y*. En particulier les *1*-graphes sont des graphes simples orientés.



G=(N,A) ; N={x, y, z, t, u} ;

A={a, b, c, d, e, f, g} ;

Origine(a)=y ; extrémité(a)=x ;

Origine(b)=u ; extrémité(b)=t ;

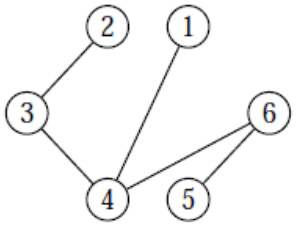
Origine(c)=t ; extrémité(c)=u ;

FIG. I.2 - Un multi-graphe orienté.

Définition 1.14

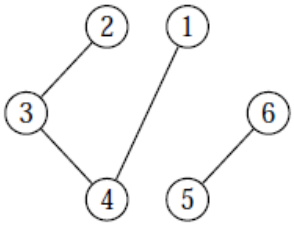
On appelle **arbre** tout graphe connexe sans cycle. Un graphe sans cycle mais non connexe est appelé une **forêt**.

Une **feuille** ou **sommet pendant** est un sommet de degré 1.



Arbre

Les sommets 1, 2 et 5 sont les feuilles



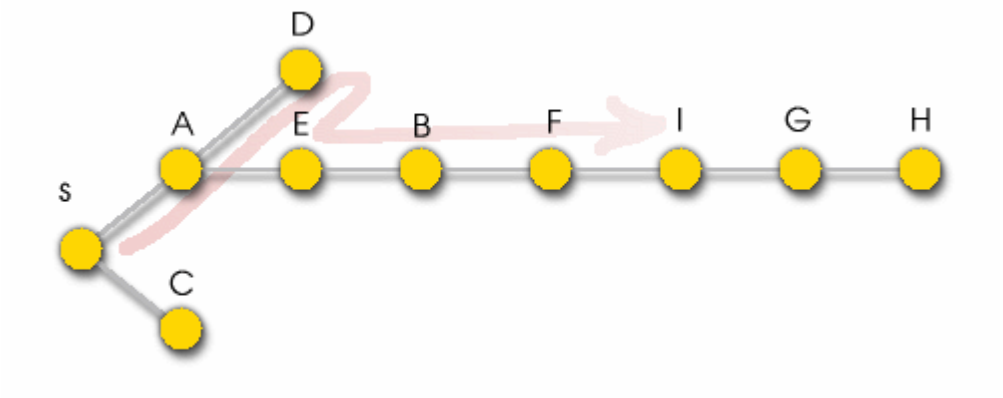
Forêt

Les sommets 1, 2, 5 et 6 sont les feuilles

FIG. I.3 - arbre et foret

I-1-14-1-la recherche en profondeur DFS (*Depth First Search*)

Dans l'exploration, l'algorithme cherche à aller très vite "profondément" dans le graphe, en s'éloignant du sommet ***s*** de départ. La recherche sélectionne à chaque étape un sommet voisin du sommet marqué à l'étape précédente.



Les sommets sont visités depuis ***s*** dans l'ordre ***A, D, E, B, F, I, G, H, C***

FIG. I.4- recherche en profondeur

I-2-Isomorphisme de graphes

Définition 2.1.

Soient $G_1 = (V_1,E_1)$ et $G_2 = (V_2,E_2)$ deux digraphes (resp. deux graphes non orientés). Une application $f : V_1 \rightarrow V_2$ est un **homomorphisme** si :

$$(x, y) \in E_1 \Rightarrow (f(x), f(y)) \in E_2$$

$$(\text{resp. } \{x, y\} \in E_1 \Rightarrow \{f(x), f(y)\} \in E_2).$$

On parlera alors d'homomorphisme de G_1 dans G_2. Il est clair que la composée d'homomorphismes est encore un homomorphisme.

Exemple : Avec les graphes G et H de la figure 1.3, on voit facilement qu'on a un homomorphisme de G dans H mais pas de H dans G. A la figure 1.4, on donne un autre exemple d'homomorphisme entre deux graphes G et H. Cela montre que $f : V_1 \rightarrow V_2$ n'est pas n´nécessairement injectif.

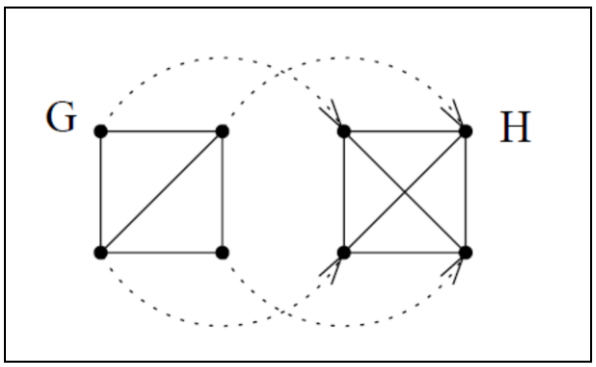


FIG. I.5 - Homomorphisme De G dans H.
.

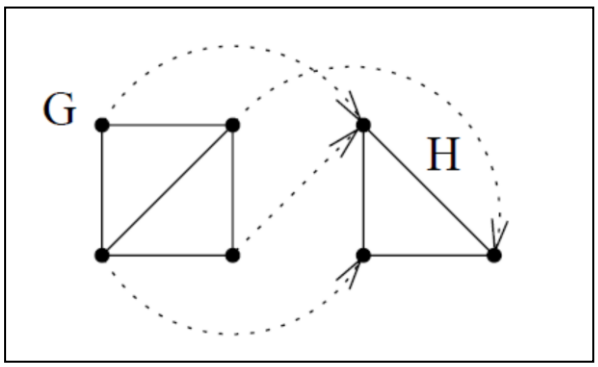


FIG. I.6 - Homomorphisme De G dans H.

Définition 2.2.

Deux graphes (resp. deux graphes non orientés) $G_1 = (V_1, E_1)$ et $G_2 = (V_2, E_2)$ sont **isomorphes** s'il existe une bijection $f : V_1 \rightarrow V_2$ qui est telle que :

$(x, y) \in E_1 \Leftrightarrow (f(x), f(y)) \in E_2$

(resp. telle que $\{x, y\} \in E_1 \Leftrightarrow \{f(x), f(y)\} \in E_2$).

Cette définition s'adapte au cas de multi-graphes orientés. Deux multi-graphes$G_1 = (V_1, E_1)$ et $G_2 = (V_2,E_2)$ sont isomorphes s'il existe une bijection $f : V_1 \rightarrow V_2$ telle que (x, y) est un arc de multiplicité k de $\mathbf{G_1}$ si et seulement si (f(x), f(y)) est un arc de multiplicité k de G_2. Bien évidemment, une telle application f est qualifiée d'isomorphisme de graphes. Bien sûr, si f est un isomorphisme, il en va de même pour f^{-1}.

Exemple

Voici deux graphes isomorphes représentés à **la figure 1.5.** On a un isomorphisme donné par$\phi : a \rightarrow 4, b \rightarrow 5, c \rightarrow 6, d \rightarrow 1, e \rightarrow 2, f \rightarrow 1$.

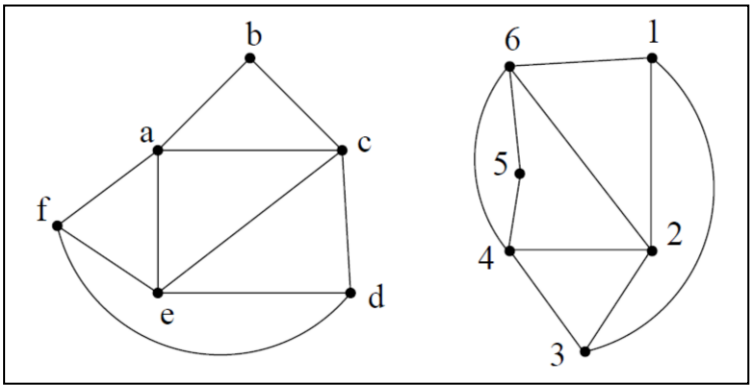


FIG. I.7 - Deux graphes isomorphes.

Définition 2.3. Soit G = (V, E) un graphe (orienté ou non). Un automorphisme de G est un isomorphisme de G dans G.

Conclusion

Les Graphes modélisent de nombreuses situations concrètes où interviennent des objets en interaction. Par exemple

♦ Les interconnections routières, ferroviaires, aériennes, chimie, biologie.

Ce domaine qui est la théorie de graphe campâtes énormément d'outillés (les algorithmes) très utiles dans la modélisation, explication, et démonstration dans beaucoup de domaines dans de la théorie à la pratique

Chapitre II

l'appariement des graphes

Sommaire

Introduction

L'appariement entre graphes est le processus qui consiste à rechercher d'une correspondance entre les nœuds et les arêtes de deux graphes, tout en assurant la satisfaction des certaines contraintes.

Généralement, les méthodes d'appariement sont divisées en deux grandes catégories : la première contient les méthodes d'appariement exact qui nécessitent une correspondance stricte entre les deux graphes ou au moins entre des sous-parties. La deuxième catégorie définit les méthodes d'appariement inexact, où une mise en correspondance peut se produire entre deux graphes même s'ils sont différents dans une certaine mesure.

Dans le présent chapitre, nous présentons respectivement la définition de chacune de ces catégories et un état de l'art des méthodes associées regroupées selon leurs algorithmes de base.

II-1-Appariement exact

II-1-1-Définition

L'appariement exact est caractérisé par le fait que la connexité entre les nœuds doit être préservée : si deux nœuds (du premier graphe) sont liés par une arête, ils correspondent à deux nœuds (du deuxième graphe) qui sont aussi liés par une arête. **[17]**

La plupart des algorithmes d'appariement exact sont basés sur la notion d'arbre de recherche et de retour-arrière. D'autres techniques sont également utilisées comme la théorie des groupes, l'arbre de décision, etc. Dans la suite nous présentons les travaux les plus connus qui ont abordé l'appariement exact. Ils sont regroupés en deux catégories : la première pour les algorithmes qui sont basés sur l'arbre de recherche et la seconde pour les autres techniques **[17].** Dans ce qui suit nous détaillons les méthodes d'appariement exact de la forme la plus stricte à la plus relâchée.

Un isomorphisme existe entre les deux graphes $\boldsymbol{G_D}$ et $\boldsymbol{G_M}$en entrée, trois solutions sont alors possibles : **[16].**

*$|\boldsymbol{G_D}| = |\boldsymbol{G_M}|$ Alors un isomorphisme de graphes existe entre $\boldsymbol{G_D}$ **et** $\boldsymbol{G_M}$
***$|\boldsymbol{G_D}| < |\boldsymbol{G_M}|$** Alors $\boldsymbol{G_D}$ est isomorphe d'un sous-graphe de $\boldsymbol{G_M}$
***$|\boldsymbol{G_M}| > |\boldsymbol{G_D}|$** Alors $\boldsymbol{G_D}$ est un isomorphe d'un sous-graphe de $\boldsymbol{G_M}$

II-1-2. Arbre de recherche

L'idée de base de ces techniques est qu'un appariement partiel (initialement vide) est itérativement élargi en y ajoutant de nouvelles paires de nœuds appariés. Les nouvelles paires doivent :

- assurer leur compatibilité avec les contraintes imposées par le type d'appariement souhaité ;
- prendre en compte les paires déjà appariées dans l'appariement partiel ;

- élaguer le plus tôt possible les chemins infructueux.

En fin de compte, soit l'algorithme trouve un appariement complet, soit il atteint un point où l'appariement partiel ne peut plus être élargi. Dans le dernier cas, l'algorithme fait un marche arrière « retour sur trace » (en anglais **« backtracking »**) : annule les derniers ajouts jusqu'à trouver un appariement partiel pour lequel une extension alternative est possible.
Si toutes les possibilités qui satisfaisons les contraintes ont été déjà essayées, l'algorithme s'arrête (l'appariement entre ces graphes n'est pas possible).
Plusieurs stratégies de mise en œuvre de ce type d'algorithme ont été utilisées. La plus simple est **« la recherche en profondeur d'abord »** qui requiert moins de mémoire que d'autres. De plus, il convient très bien à une formulation récursive. Une propriété intéressante de cette stratégie est qu'il peut facilement prendre en compte les attributs des nœuds et des arêtes. Ceci est très important lorsque les attributs jouent un rôle dans la réduction du temps d'appariement**.[16]**

II-1-2-1-L'algorithme d'Ullmann

Est le premier et le plus connu de cette famille.
En dépit de son ancienneté, il est encore couramment employé. Cet algorithme aborde les problèmes d'isomorphe, du sous-graphe isomorphe et du monomorphe. L'auteur a proposé une procédure de raffinement qui utilise une matrice des futures paires possibles de nœuds à apparier afin d'élaguer les chemins infructueux. A chaque étape, les paires qui ne sont pas compatibles avec l'appariement partiel seront supprimées. **[17]**

Ullmann propose un algorithme reconnu pour être un des plus efficaces en terme de réduction de la taille de l'arbre de recherche. L'idée de base consiste à ne pas tester certains appariements dont on sait qu'ils ne fourniront pas d'isomorphisme. Dans un premier temps, le problème est reformulé en utilisant les matrices d'adjacence **[16]**

II-1-2--2- Les algorithmes VF et VF2

Ces algorithmes permettent de travailler sur des graphes de grande taille dans la

mesure où ses besoins en mémoire sont moindres que la plupart des autres algorithmes déterministes grâce à une fonction dite de faisabilité qui optimise l'arbre de recherche en se basant sur l'analyse des nœuds adjacents à ceux déjà appariés. Cette heuristique est facile à calculer dans la plupart des cas et améliore de manière significative les performances de

l'algorithme d'Ullmann et de ses dérivés permettant d'être efficace sur des graphes de grandes tailles.

Un algorithme plus récent pour le cas de l'isomorphe et du sous-graphe isomorphe est l'algorithme de VF. Les auteurs définissent une méthode basée sur l'analyse des ensembles de nœuds qui sont adjacents à ceux déjà pris dans l'appariement partiel.Cette méthode est assez rapide, ce qui a mené à une amélioration significative du temps d'appariement (dans des nombreux cas) par rapport aux autres algorithmes. les auteurs ont proposé une modification de cet algorithme afin de réduire l'espace mémoire nécessaire. Cette amélioration rend cet algorithme intéressant pour travailler avec les grands graphes. **[16]**

II-1-3. Autres techniques

II-1-3-1-Entités canoniques

II-1-3-1-a-Algorithme Nauty

Est probablement l'algorithme pour les graphes isomorphes le plus intéressant qui n'est pas basé sur l'arbre de recherche. Il est considéré par beaucoup d'auteurs comme l'algorithme pour les graphes isomorphes le plus rapide disponible jusqu'à présent. Il est basé sur la théorie des groupes. **[17]**

D'autres techniques de comparaison de graphes se basent sur des outils mathématiques, comme la théorie des groupes. Un algorithme très populaire pour sa rapidité est celui de **McKay**. Cet algorithme, qui permet de déterminer l'isomorphisme entre deux graphes, est basé sur un autre algorithme qui permet d'obtenir une représentation canonique des graphes. Déterminer si deux graphes sont isomorphes revient donc à comparer les deux représentations canoniques.

Ce type de méthode est par conséquent très intéressant pour comparer un nouveau graphe avec un grand nombre de graphes dont les représentations canoniques ont été pré-calculées**. [18]**

Actuellement, la méthode utilisant le concept d'entité canonique la plus performante, et donc la plus populaire est **nauty**. L'algorithme utilisé par **nauty** (et par la plupart des

méthodes similaires) est un algorithme à retour-arrière qui parcourt le graphe pour rechercher les entités canoniques. Les nœuds de chaque graphe sont ordonnés en se basant sur un étiquetage canonique. Cet étiquetage canonique permet à l'algorithme **nauty** d'être encore à ce jour un des plus performants de sa catégorie et reste une référence pour la recherche d'isomorphisme avec VF2 suivant la topologie des graphes à analyser **[16]**

II-1-3-1-b-Arbre de décision

Un autre algorithme qui s'intéresse aux graphes isomorphes et au sous-graphe isomorphe a été présenté en 1995 par Messmer. Cette méthode est conçue afin de faire l'appariement entre un graphe et une base de graphes. Elle est basée sur une décomposition récursive de chaque graphe de la base dans des sous graphes plus petits, jusqu'à arriver à un niveau trivial (un noeud).

Les auteurs ont proposé un algorithme plus performant pour l'isomorphe et le sous-graphe isomorphe entre un graphe et une base de graphes. En phase de prétraitement, un arbre de décision est construit à partir des graphes de la base. **[17]**

II-2. Appariement inexact

L'objectif des méthodes de comparaison inexacte est de trouver une solution satisfaisante pour un critère donné. Le critère le plus utilisé est naturellement l'erreur commise vis à vis de la solution optimale, mais d'autres critères sont aussi utilisés comme un temps de calcul maximal, ou des combinaisons de différents critères. **[18]**

Cette approche a deux principaux intérêts.

1- Le premier est qu'elle permet de trouver une solution intéressante en un temps raisonnable, et par conséquent de traiter des graphes de plus grande taille.

2- Le deuxième est qu'elle permet de résoudre des problèmes bruités, dont la solution exacte ne peut être identifiée clairement. Ces méthodes sont plus souples, et permettent de trouver des correspondances partielles là où les méthodes exactes ne trouvent aucune solution. **[18]**

II. 2. 1. Arbre de recherche

Deux catégories de tels algorithmes existent.

II-2-1-1-Les méthodes optimales

Les algorithmes estiment le coût global d'appariement et restituent les graphes dont

ce coût est minimal. Cela implique que si la solution exacte existe, elle sera retrouvée par de tels algorithmes. Ce type d'algorithmes (qui sont appelés optimaux) permet de résoudre le problème de déformation mais ils n'assurent pas une amélioration en temps de calcul. Ils sont généralement plus coûteux que leurs homologues exacts. **[17]**

II-2-1-2-Les méthodes approximatives

L'autre type d'algorithmes d'appariement inexact (qui sont appelés approximatifs) cherche seulement à minimiser les coûts locaux (au niveau de chaque noeud) dans le processus d'appariement. Les résultats obtenus ne sont pas généralement très différents de ceux obtenus par les algorithmes optimaux. **[17]**

L'inconvénient majeur de ces algorithmes est l'incapacité de garantir de trouver la solution exacte si elle existe. Par contre, cette perturbation dans le résultat est compensée par un gain du temps qui est généralement polynomial.

Les méthodes d'appariement inexact approximatif qui est basée sur la décomposition de deux graphes en des graphes élémentaires. Un graphe élémentaire est composé d'un noeud, de ses voisins ainsi que des arêtes qui le relient à ces derniers. **[17]**

II-2-1-2-1-L'algorithme A*

Parmi les travaux fondés sur l'arbre de recherche, nous pouvons citer aussi la méthode d'appariement inexact approximative, où l'algorithme de recherche A* a été utilisé. Le même algorithme a été employé plus récemment. Dans ce dernier, les futurs coûts d'appariement partiel sont estimés en se basant sur une méthode d'appariement entre graphes bipartis. L'idée de base consiste à trouver le plus grand appariement entre deux ensembles de noeuds formant un graphe biparti. Les auteurs ont prouvé que l'appariement selon cette manière peut être fait en temps polynomial. **[17]**

II. 2. 2. Optimisation continue

Les méthodes d'appariement mentionnées ci-dessus se basent sur la formulation du problème en termes de graphes (c'est-à-dire en nœuds et en arêtes). **[17]**

II-2-2-1-La première famille de ces méthodes utilise la technique de « relâchement d'étiquetage » (en anglais « relaxation labeling »). L'un des travaux pionniers de cette approche est présenté dans [Fischler et al. 1973].

L'idée de base consiste à assigner à chaque nœud d'un graphe une étiquette parmi un ensemble discret d'étiquettes possibles.

Les étiquettes sont utilisées pour déterminer les correspondances des nœuds dans l'autre graphe.

Un inconvénient majeur de cette famille de méthodes concerne le fait que la correspondance entre les nœuds et les étiquettes est fait dans un seul sens : à la fin de l'algorithme, chaque nœud a une seule étiquette, mais il n'y a aucune garantie que chaque étiquette soit attribuée à un seul nœud. Pour certaines applications, ceci ne constitue pas un gros problème. **[17]**

Les algorithmes d'optimisation continue forment un second grand groupe. Le problème est transformé dans le domaine continu et devient un problème d'optimisation non-linéaire.

Les méthodes dans cette catégorie sont en général approchées et elles nécessitent la transformation de la solution dans le domaine d'origine. Il existe des méthodes basées sur la relaxation probabiliste, **[16]**

II-2-2-2-Une autre famille de méthodes est basée sur la formulation du problème comme un Appariement des Graphes Pondérés (AGP en anglais WGM). Cela consiste à trouver une correspondance entre deux sous-ensembles de nœuds des deux graphes (généralement exprimée par une matrice d'appariement *M*) **[17]**

II-2-2-2-1-de décomposition lagrangienne

Une autre méthode dans cette famille basée sur l'approche de décomposition lagrangienne [Guignard et al. 1987]. Dans cette méthode, les contraintes sur les lignes et sur les colonnes de la matrice *M* sont résolues séparément, puis fusionnées par le biais d'un multiplicateur de Lagrange **[17]**

II-2-2-2-2- AFG et AGP

Plusieurs autres méthodes d'appariement inexact basées sur l'optimisation continue ont été proposées ces dernières années. Entre autres, il existe l'Appariement Flou de Graphes (AFG en anglais FGM), qui est une version simplifiée d'AGP basée sur la logique floue. En fait, le calcul de coût d'appariement des deux nœuds proposé dans cette méthode ne dépend pas des autres nœuds, ce qui la rend plus simple qu'AGP. **[17]**

II. 2. 3. Méthodes spectrales

Les méthodes spectacles sont basées sur l'observation suivante : les valeurs propres et les vecteurs propres de la matrice d'adjacence d'un graphe sont invariants par rapport aux permutations des nœuds. Par conséquent, si deux graphes sont isomorphes, leurs matrices d'adjacence auront les mêmes valeurs/vecteurs propres. Ces méthodes ne sont pas en mesure d'exploiter les attributs des nœuds et des arêtes, ce qui est souvent nécessaire dans le processus d'appariement. **[17]**

II-2-3-1-Le premier algorithme de cette catégorie est présenté dans [Umeyama 1988].

Une des premières méthodes est proposée par Umeyama **[16]**

L'auteur a utilisé les valeurs/vecteurs propres des matrices d'adjacence des graphes pour dériver une expression de la matrice orthogonale

Une méthode combinant l'approche spectrale avec l'idée de regroupement a été présentée dans [Kosinov et al. 2002]. Dans cette méthode, un espace vectoriel, appelé l'espace

propre du graphe, est définie en utilisant les vecteurs propres des matrices d'adjacence et les nœuds qui sont projetés comme des points dans cet espace. Ensuite, un algorithme de regroupement a été utilisé pour trouver les nœuds des deux graphes qui doivent être appariés. Les auteurs ont montré que cette méthode est robuste à la distorsion des graphes. **[17]**

II. 2. 4. Autres techniques

La méthode de décomposition introduite dans [Messmer et al. 1997] pour l'appariement exact a été étendue pour l'appariement inexact dans [Messmer et al. 1998].

Les auteurs ont montré que leur algorithme peut estimer les coûts optimaux d'appariement d'un graphe avec une base en temps polynomial (en fonction du nombre de graphes de la base). Les auteurs ont proposé une amélioration de cet algorithme en utilisant une méthode stochastique afin de limiter l'espace de recherche. **[17]**

II-2-4-1- Les réseaux de neurones

Des méthodes d'appariement, d'une deuxième famille, sont basées sur l'utilisation des réseaux de neurones. La plupart de ces méthodes emploient un réseau de neurones de type Hopfield qui cherche à minimiser l'énergie. Une autre approche a proposé l'utilisation d'un réseau de neurones récurrents pour calculer la distance entre les graphes orientés acycliques. Cette technique se base sur la projection des graphes dans un espace vectoriel et sur l'utilisation de la distance euclidienne. **[17]**

II-2-4-2--Les algorithmes génétiques sont également utilisés dans l'appariement entre graphes.

Parmi les travaux qui se sont penché sur ces algorithmes, où un algorithme micro-génétique est appliqué pour résoudre des AGP problème, un algorithme génétique est utilisé pour trouver l'« erreurs-correction » un algorithme génétique est employé pour un homomorphe flou. Un algorithme génétique intéressant est présenté pour trouver le SCM entre deux graphes. **[17]**

Conclusion

Les problèmes d'isomorphisme de graphes peuvent être différenciés en deux catégories les problèmes exacts et les problèmes inexacts. Les isomorphismes exacts veulent que les graphes étudiés soient exactement isomorphes alors que les isomorphismes inexacts tolèrent des variations d'attribut et/ou de structure. Ces derniers sont souvent formulés comme des problèmes de minimisation de distance alors que les isomorphismes exacts s'apparentent à des problèmes booléens..

Chapitre III

Etude détaillée des algorithmes

Sommaire

Introduction

La plupart des algorithmes de comparaison exacte sont basées sur une représentation par arbre de recherche. L'idée est de représenter chaque mise en correspondance sous la forme d'une chaîne dans un arbre de recherche. Pour ce faire, on représente chaque mise en correspondance entre deux sommets sous la forme d'un nœud dans l'arbre de recherche, et chaque mise en correspondance entre deux arêtes sous la forme d'un lien entre deux nœuds de l'arbre de recherche. Cela permet de diviser les calculs de mise en correspondance dans le but d'éviter de refaire plusieurs fois le même calcul. Cela permet aussi d'éviter certains calculs en élaguant certaines branches de l'arbre. Notons qu'une des conditions pour obtenir une solution exacte est que toutes les solutions soient atteignables en explorant l'arbre de recherche. Les algorithmes sont généralement basés sur des recherches en profondeur d'abord ou sur le branch-and-bound.

L'une des premières méthodes qui a rencontré un franc succès dans la littérature est celle d'Ullmann. Cette méthode peut résoudre les problèmes d'isomorphisme et d'isomorphisme de sous-graphe.

VF2 se base sur une stratégie de recherche en profondeur avec un ensemble de règles pour élaguer l'abre de recherche et utilise des structures de données plus sophistiquées.

III-1-Définition L'algorithme Ullmann

Est le plus populaire pour résoudre le problème de l'isomorphisme de sous-graphe. Malgré qu'il date de **1976**, il est encore largement utilisé. Les problèmes abordes par l'algorithme sont l'isomorphisme de graphes, l'isomorphisme de sous-graphes et le monomorphisme. Pour tailler les appariements infructueux, Ullmann propose une procédure dite de raffinement, qui travaille sur une matrice de nœuds possible a apparies dans le futur pour enlever, en se basant sur une condition nécessaire et bien définie, ceux qui ne sont pas compatibles avec le courant appariement partiel. **[01]**

C'est le plus connu pour résoudre le problème d'isomorphisme de sous graphes dans le cas général des graphes orientés. Il s'applique à un graphe de motif $\boldsymbol{G_M}$ à détecter et à un graphe $\boldsymbol{G_D}$ dans lequel faire la reconnaissance. Comme détaillé par la suite, sa complexité est exponentielle en le nombre de sommets maximal du graphe de motif $\boldsymbol{G_M}$

III-2-Le fonctionnement de cet algorithme se fait par énumération de tous les sous-graphes et vérification si le sous-graphe construit est isomorphe au second graphe. Pour éviter de considérer des sous-graphes impossibles, l'algorithme se dote d'une procédure, exécutée après chaque ajout de nœud dans le sous-graphe en construction, visant à réduire le nombre de nœuds possibles à considérer après le nœud inséré**. [08]**

Cet algorithme permet une exploration de la combinatoire tout en permettant une réduction significative de la complexité par un principe de retour arrière. Un certain nombre d'articles ont proposé des algorithmes avec des complexités moindres exploitant certaines contraintes **[10]**

L'algorithme d'Ullmann est basé sur la procédure récursive de backtracking en combinaison avec une procédure de raffinement de l'espace de recherche (la routine *forward checking*). L'entrée de l'algorithme est composée d'un graphe modèle $\boldsymbol{G_M}$ $(\boldsymbol{V_M}, \boldsymbol{E_M})$ et d'un graphe de données $\boldsymbol{G_D}$ $(\boldsymbol{V_D}, \boldsymbol{E_D})$. **[11]**

L'algorithme d'Ullmann cherche à trouver tous les isomorphismes de sous-graphes possibles du graphe $\boldsymbol{G_D}$ à partir du graphe $\boldsymbol{G_M}$. Pour réduire le nombre d'appariements à tester, une approche d'appariement incrémentale des sommets a été proposée. A chaque étape, un couple de sommets est ajouté dans l'appariement courant et la condition d'isomorphisme de sous-graphes est testée. Si, pour un certain couple de sommets l'isomorphisme n'était pas

trouvé, le système fait un retour en arrière « *backtrack* » et le sommet du graphe modèle utilisé précédemment va être mis en correspondance avec un autre sommet afin de tester une nouvelle condition d'isomorphisme. L'espace de recherche se réduit très efficacement grâce à la technique de « *forward checking* ». Elle a pour but de détecter par avance des instanciations inconsistantes en appliquant un critère de test de consistance sur les arcs pendant la recherche. **[11]**

La complexité de l'algorithme d'Ullmann dépend de la taille des graphes. Supposons que la taille du graphe modèle $\boldsymbol{G_M}$ est $\boldsymbol{n}$ et celle du graphe $\boldsymbol{G_D}$ est $\boldsymbol{m}$, le coût de l'algorithme dans le meilleur des cas est ***O(mn).*** Dans le pire des cas, lorsque la connexité du graphe est très élevée, le coût augmente jusqu'à $\boldsymbol{O(m^n n^2)}$. Par conséquent, le temps de traitement explose exponentiellement. La figure. **III-1** fournit l'algorithme d'Ullmann. **[11]**

Un graphe attribué *G* défini par un 4-tuple $\boldsymbol{G = (V, E, L_V, L_E)}$ où : $\boldsymbol{V}$ est l'ensemble fini des sommets, $\boldsymbol{E \sqsubseteq V * V}$ est l'ensemble des arcs, $\boldsymbol{L_V : V \rightarrow \sum_V * A_V^K}$ est une fonction qui assigne les étiquettes aux sommets, $\boldsymbol{L_V : V \rightarrow \sum_V * A_V^K}$ est une fonction qui assigne les étiquettes aux arcs. **[11]**

III-3-algorithme Ullmann [11]

Algorithme d'Ullmann ($\boldsymbol{(G, G')}$

Entrée : deux graphes attribués $\boldsymbol{G} = (V, E, L_V, L_E)$, $\boldsymbol{G'} = (V', E', L_V', L_E')$.

Sortie : $\boldsymbol{f}$ **[1 … |*V*|]** vecteur représentant un isomorphisme de sous-graphes du $\boldsymbol{G}$ à $\boldsymbol{G'}$.

/* *f*[*i*] = *j* montre que le sommet $\boldsymbol{v_i \in V}$ a été associé avec le sommet $\boldsymbol{w_j \in V'}$ */

Soit **P = [1… |*V*| ; 1…|*V'* |]** une matrice de compatibilité entre le sommet du ***G*** et ***G'***.

0. **début**
1. **pour** *i* = 1 à |*V* |
2. **pour** *j* = 1 à |*V* ' |
3. **si** $L_V(\boldsymbol{v_i}) = L_V'(w_j)$ **alors** *P* [*i*, *j*] : = 1 ;
4. **sinon** *P* [*i*, *j*] : = 0 ;
5. **pour** *i* = 1 à |*V* |
6. *f*[*i*] : = 0 ;
7. **retourner** Backtracking (*P*, 1, *f)*
8. **fin.**

III-3-1-Algorithm Backtracking (*P, i, f*) **[11]**

Entrée : Une matrice de compatibilité ***P*** entre les sommets de deux graphes **G** et $\boldsymbol{G}'$, le niveau courant ***i*** dans l'arbre de recherche, un vecteur ***f*** représentant l'isomorphisme partiel jusqu'au niveau ***i*-1**.

Sortie : l'isomorphisme partial *f* jusqu'au niveau ***i***.

0. **début**
1. **si** $i > |V|$ **alors retourner** *f* comme un isomorphisme de sous graphe du ***G*** à ***G*'**.
2. **sinon**
3. **pour** $j = 1$ à $|V'|$
4. **si P** $[i, j] = 1$ **alors**
5. **début**
6. $f[i] := j$;
7. $P' := P$;
8. **pour** $k = i + 1$ à $|V|$ $p'[k, j] := 0$;
9. **si** Forward_checking (p', i, f) **alors**
retourner Backtracking ($p', i + 1, f$) ;
10. **sinon** $f[i] := 0$;
11. **fin.**
12. **fin.**

III-3-2-Algorithm Forward checking (*P, i, f*) **[11]**

Entrée : Une matrice de compatibilité ***P*** entre les sommets de deux graphes ***G*** et$\boldsymbol{G}'$, le niveau courant ***i*** dans l'arbre de recherche, un vecteur ***f*** représentant l'isomorphisme partiel jusqu'au niveau ***i*-1**.

Sortie : **VRAI** si pour chaque sommet $\boldsymbol{V_k} \in \boldsymbol{V}(\boldsymbol{k} > \boldsymbol{i})$ il existe au moins un appariement compatible avec appariements des sommets existants présents dans ***f***, **FAUX**, sinon.

0. **début**
1. **pour** $k = i + 1$ à $|V|$
2. **pour** $j = 1$ à $|V'|$
3. **si P** $[k, j] = 1$ **alors**
4. **pour** $l = 1$ à i -1

5. **début**

6. vérifier les contraintes suivantes de consistance des arcs :

(1) pour chaque arc e = **(*vk, vl*) ∈*E*** existe est-il un arc ***e'*= (*wj, wf*[l]) ∈ *E'*** tel que $\boldsymbol{LE(e) = L'_E(e')}$

(2) pour chaque arc ***e'*= (*wj, wf*[l]) ∈ *E'*** existe est-il un arc ***e* = (*vk, vl*) ∈*E*** tel que $\boldsymbol{LE(e) = L'_E(e')}$

7. **si** (1) ou (2) sont FAUX **alors** *P* [*k, j*] : = 0 ;

8. **fin**

9. **si** il existe une ligne **k** *dans* **P** *telle* que **P [*k, j*] = 0** pour *j* = 1…. $|V'|$ **alors retourner** FAUX

10. **sinon retourner** VRAI ;

11. **fin**.

L'algorithme d'énumération [08]

Soient deux graphes, $\boldsymbol{G_a\,(V_a, E_a)}$ et $\boldsymbol{G_b\,(V_b, E_b)}$ et leur matrice d'adjacence, respectivement, $\boldsymbol{A = [a_{ij}]}$ et $\boldsymbol{B = [a_{ij}]}$ avec $\boldsymbol{p_a = |V_a|, q_a = |E_a|}$, $\boldsymbol{p_b = |V_b|}$ et $\boldsymbol{q_b = |E_b|}$

Cet algorithme va énumérer tous les isomorphismes entre $\boldsymbol{G_B}$ et des sous-graphes de $\boldsymbol{G_A}$. Pour ce faire, une matrice $\boldsymbol{M'}$ de taille $\boldsymbol{p_a * p_b}$ est définie comme une matrice de **0** ou de **1** telle que chaque ligne contient un et un seul **1**, tandis que chaque colonne contient au plus un **1**. Celle-ci sera utilisée pour permuter des lignes et colonnes de **B**, dans le but de construire une matrice

$\boldsymbol{C = M'(M'B)^T}$ Telle que si

$$\boldsymbol{\forall_i(1 \leqslant i \leqslant p_a), \forall_j(1 \leqslant j \leqslant p_b) : a_{ij} = 1 \mapsto c_{ij} = 1 \ldots\ldots\ldots\ldots} \quad \textbf{(4.5)}$$

Alors $\boldsymbol{M'}$ détermine un isomorphisme entre $\boldsymbol{G_B}$ et un sous-graphe de $\boldsymbol{G_A}$.

Dès lors, si $\boldsymbol{m'_{ij} = 1}$, alors le $\boldsymbol{i^{me}}$ point de $\boldsymbol{G_a}$ correspond au $\boldsymbol{j^{me}}$ point de $\boldsymbol{G_b}$.

Au départ de l'algorithme d'énumération, une matrice $\boldsymbol{p_a * p_b}$ $\boldsymbol{M^0 = m^0{}_{ij}}$ est construite de la manière suivante :

$\boldsymbol{m^0{}_{ij}}$ ⌈ 1 si le degré du $\boldsymbol{j^{me}}$ nœud de $\boldsymbol{G_B} \geqslant$ le degré du $\boldsymbol{i^{me}}$ noeud de $\boldsymbol{G_A}$

⌊ 0 sinon.

En effet, si le degré du nœud $j \epsilon V_b$ est inférieur au degré du nœud $i \epsilon V_a$, il n'est pas possible de trouver une correspondance entre le nœud i du graphe G_a et le nœud j d'un sous-graphe de G_b puisque l'adjacence sera déjà incorrecte. Par contre, si le nœud j a un degré supérieur au nœud i, cette correspondance peut mener à un isomorphisme puisqu'on pourra toujours supprimer des nœuds adjacents à j pour obtenir le sous-graphe et, ainsi, avoir une adjacence similaire

L'algorithme d'énumération fonctionne donc de la manière suivante : partant de M^0, une matrice M' est générée en ne laissant qu'un seul 1 sur chaque ligne de M' (et en faisant en sorte, bien sûr, qu'il n'y a pas plus d'un 1 dans chaque colonne). Une fois cette matriceM' générée, la condition 4.5 est testée et la matrice M' est conservée si la condition se révèle vraie. Dans l'arbre de recherche, les feuilles sont les M' différents et la profondeur de ces feuilles est p_a. Chaque noeud de profondeur inférieure à p_a contient une matrice M_d , d étant la profondeur, telle que d lignes de cette matrice ont déjà été traitées (ne contiennent donc plus qu'un et un seul **1**).

La procédure d'épuration [08]

Il est facile de se rendre compte que l'algorithme décrit ci-dessus demande un grand nombre de calculs à faire. Pour réduire ce nombre, une procédure d'épuration de l'arbre de recherche a été établie, qui élimine certains **1** de **M** et, donc, élimine certains nœuds successeurs dans l'arbre de recherche. M est donc un nœud non-terminal de l'arbre de recherche et nous appelons la matrice M' un isomorphisme sous **M** si M' est une des feuilles du sous-arbre de racine **M**. Il est évident que les **0** dans M déterminent une non-correspondance entre des nœuds de G_a et des nœuds de G_b. Dès lors, pour chaque isomorphisme M' sous M, si $m'_{ij} = 0$, et $m_{ij} = 1$, (c'est-à-dire que l'isomorphisme M' affirme que les nœuds **i** et **j** ne correspondent pas, tandis que, dans **M**, $m'_{ij} = 1$,), alors nous pouvons mettre à **0** le m_{ij} sans perdre des isomorphismes sous **M**. Dès lors, nous pouvons imaginer une condition nécessaire qui travaille sur les adjacences et telle que, si elle n'est pas satisfaite pour un certain $m_{ij} = 1$, alors ce $m_{ij} = 1$ sera changé en $m_{ij} = 0$.

Soit v_{ai} le i^{me} nœud de V_a et v_{bj} le j^{me} nœud de V_b. Soient $\{v_{a1}, v_{a2} \ldots\ldots , v_{ak}\}$ l'ensemble des nœuds de V_a adjacents à v_{ai}. Considérons l'isomorphisme M' sous **M**. D'après la définition de l'isomorphisme de sous-graphe, il est nécessaire que l'adjacence soit

respectée, c'est-à-dire que si $\boldsymbol{v_{ai}}$ correspond à $\boldsymbol{v_{bj}}$ dans $\boldsymbol{M'}$, alors, pour chaque $\boldsymbol{x = 1, 2, \ldots, k}$ il doit exister un nœud $\boldsymbol{v_{by} \epsilon V_b}$. Qui est adjacent à $\boldsymbol{v_{bj}}$ tel que $\boldsymbol{v_{by}}$ correspond à vax dans ce même isomorphisme $\boldsymbol{M'}$. Dès lors, si $\boldsymbol{v_{ai}}$ correspond à $\boldsymbol{v_{bj}}$ dans chaque isomorphisme sous **M**, alors pour chaque $\boldsymbol{x = 1, 2, \ldots, k}$, il doit y avoir un **1** dans **M** correspondant à $\{\boldsymbol{v_{ax,}\ v_{by}}\}$ tel que $\boldsymbol{v_{by}}$ est adjacent à $\boldsymbol{v_{bj}}$. En termes de formule, cela donne :

$$\forall_x(\mathbf{1} \leqslant \boldsymbol{x} \leqslant \boldsymbol{p_a})\boldsymbol{a_{ix}} = \mathbf{1} \Rightarrow \exists_y(\mathbf{1} \leqslant \boldsymbol{y} \leqslant \boldsymbol{p_b}) : \boldsymbol{m_{xy}} * \boldsymbol{b_{yj}} = \mathbf{1} \ldots\ldots \text{(4.6)}$$

Si cette condition n'est pas satisfaite pour $\boldsymbol{m_{ij} = 1}$, alors nous pouvons changer $\boldsymbol{m_{ij} = 1}$ en $\boldsymbol{m_{ij} = 0}$. Détaillons quelque peu cette condition. Le terme $\boldsymbol{m_{xy}}$ est à **1** si le degré du nœud $\boldsymbol{v_{by}}$ est supérieur ou égal au degré du nœud $\boldsymbol{v_{ax}}$. Dès lors, pour chaque nœud $\boldsymbol{v_{ai}}$ adjacent au nœud $\boldsymbol{v_{ax}}$, il existe un nœud $\boldsymbol{v_{bj}}$ adjacent au nœud $\boldsymbol{v_{by}}$.

La procédure d'épuration teste, pour chaque **1** dans **M**, la condition 4.6 et les changements sont effectués dans le cas où elle n'est pas satisfaite. Mais, le changement d'un $\boldsymbol{m_{ij}}$ peut entraîner la non-satisfaction de la condition 4.6 sur un autre $\boldsymbol{m_{kl}}$ et, dès lors, un second changement de $\boldsymbol{m_{kl} = 1}$ vers $\boldsymbol{m_{kl} = 0}$ doit être effectué. La condition est donc évaluée constamment sur tous les **1** de **M** jusqu'à ce qu'il n'y ait plus de **1** dans **M** qui ne satisfassent pas la condition 4.6.

Une observation importante à faire est que si la condition 4.6 est évaluée sur une matrice $\boldsymbol{M'}$ et que aucun **1** dans $\boldsymbol{M'}$ n'est changé en **0**, une correspondance **1** : 1 a été trouvée entre les nœuds de $\boldsymbol{G_a}$ et les nœuds d'un sous-graphe de $\boldsymbol{G_b}$. En effet, si aucun **1** n'est changé dans $\boldsymbol{M'}$, cela signifie que, comme il y a un et un seul **1** par ligne, tous les nœuds de $\boldsymbol{G_a}$ ont été mis en correspondance avec des nœuds de $\boldsymbol{G_b}$ et, comme il y a au plus un **1** dans chaque colonne de $\boldsymbol{M'}$, cela signifie que certains (peut-être pas tous) nœuds de $\boldsymbol{G_b}$ ont été mis en correspondance avec des nœuds de $\boldsymbol{G_a}$. On a donc bien une correspondance 1 : **1** entre tous les nœuds de $\boldsymbol{G_a}$ et une partie des nœuds de $\boldsymbol{G_b}$ (donc un sous-graphe de $\boldsymbol{G_b}$). Dès lors, la condition 4.6 devient nécessaire et suffisante pour l'isomorphisme de sous-graphe et peut donc remplacer la condition 4.5 vue juste avant. De plus, pendant la procédure d'épuration, nous pouvons constamment vérifier que toutes les lignes de **M** contiennent un et un seul **1**. Si une des lignes de **M** ne contient aucun **1**, la recherche des successeurs de **M** est stoppée puisqu'on ne pourra pas trouver d'isomorphisme entre $\boldsymbol{G_a}$ et un sous-graphe de $\boldsymbol{G_b}$.

Exemple ;[24]

Soient deux graphes attribués $G_B = (V, E,)$, $G_A = (V', E',)$.

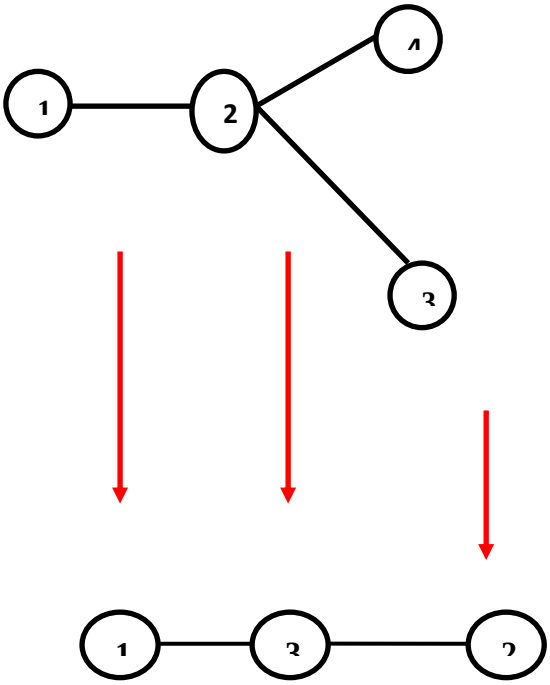


FIG. III.2 - deux graphes

Matrice d'adjacence $B = [a_{ij}]$

G_B

0	1	0	0
1	0	1	1
0	1	0	0
0	1	0	0

$B = B^T$

Matrice d'adjacence $A = [a_{ij}]$

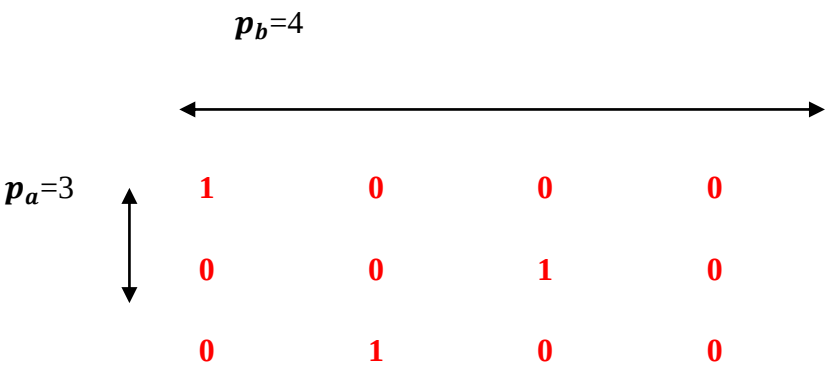
G_A

0	0	1
0	0	1
1	1	0

Matrice M' de taille $p_a * p_b$

n = | V_a |, m = | V_b |, n × m matrice de permutation M' avec forme suivante:

- M ne contient que des '**0**' et '**1**'
- exacte d'un «**1**» dans chaque rangée
- Pas plus d'un '**1**' dans chaque colonne



$$M' = p_a * p_b$$

FIG. III.3 - Matrice M' de taille $p_a * p_b$

$C = M'(M'B)^T$ Telle que si

$\forall_i(1 \leqslant i \leqslant p_a), \forall_j(1 \leqslant j \leqslant p_b) : a_{ij} = 1 \mapsto c_{ij} = 1$

Calculer $M'B$: Move row j to row i $M'_{ij} = 1$

Permuter matrice d'adjacence B en multipliant avec M'

$$\begin{matrix}1&0&0&0\\0&0&1&0\\0&1&0&0\end{matrix} * \begin{matrix}0&1&0&0\\1&0&1&1\\0&1&0&0\\0&1&0&0\end{matrix} = \begin{matrix}0&1&0&0\\0&1&0&0\\1&0&1&1\end{matrix}$$

FIG. III.4- Calculer la matrice $M'B$

-$(\boldsymbol{M'B})^{\boldsymbol{T}}$ **;** Move column j to column i

1	**0**	**0**	**0**		**0**	**1**	**0**	**0**		**0**	**0**	**1**
0	**0**	**1**	**0**	*	**1**	**0**	**1**	**1**	=	**1**	**1**	**0**
0	**1**	**0**	**0**		**0**	**1**	**0**	**0**		**0**	**0**	**1**
	$\boldsymbol{M' = M'^{T}}$				**0**	**1**	**0**	**0**		**0**	**0**	**1**
					$\boldsymbol{B = B^{T}}$							

FIG. III.5 - Calculer la matrice $(\boldsymbol{M'B})^{\boldsymbol{T}}$

Calculer $\boldsymbol{C = M'(M'B)^{T}}$ **;** Move column j to column i and row j to row i

1	**0**	**0**	**0**		**0**	**0**	**1**		**0**	**0**	**1**
0	**0**	**1**	**0**	*	**1**	**1**	**0**	=	**0**	**0**	**1**
0	**1**	**0**	**0**		**0**	**0**	**1**		**1**	**1**	**0**
					0	**0**	**1**				

FIG. III.6- Calculer $\boldsymbol{C = M'(M'B)^{T}}$

$(\boldsymbol{M'B})^{\boldsymbol{T}}$: Move column j to column **i**

$\boldsymbol{M'(M'B)^{T}}$: Move column j to column i and row j to row i

0	**0**	**1**		
0	**0**	**1**	=	**C**
1	**1**	**0**		

FIG. III.7 - résultat C

-Une matrice $\boldsymbol{p_a * p_b}$ $\boldsymbol{M^0 = m^0{}_{ij}}$ est construite de la manière suivante :

$m^0{}_{ij}$ ⌈ 1 si le degré du $G_B \geqslant$ le degré de G_A

⌊ 0 sinon.

	G_B			
	0	**1**	**0**	**0**
	1	**0**	**1**	**1**
	0	**1**	**0**	**0**
	0	**1**	**0**	**0**
Deg	**1**	**3**	**1**	**1**

G_A		
0	**0**	**1**
0	**0**	**1**
1	**1**	**0**
Deg	**1**	**1**	**2**

FIG. III.8 - Une matrice $p_a * p_b$ $M^0 = m^0{}_{ij}$

Donc : $M^0 = m^0{}_{ij}$

1	**1**	**1**	**1**
1	**1**	**1**	**1**
0	**1**	**0**	**0**

FIG. III.9 - : $M^0 = m^0{}_{ij}$

1 1 1 1

1 1 1 1

0 1 0 0

1 0 0 0

1 1 1 1

0 1 0 0

0 1 0 0

1 1 1 1

0 1 0 0

1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1

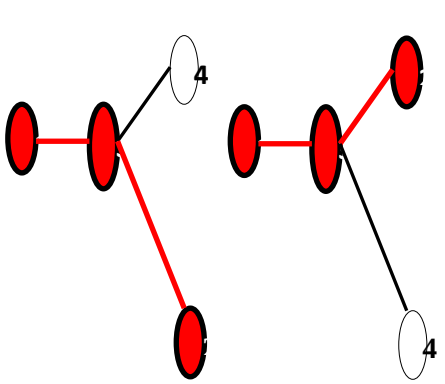
1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1

0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0

1 0 0 0 1 0 0 0 1 0 0 0

0 1 0 0 0 0 1 0 0 0 0 1

0 1 0 0 0 1 0 0 0 1 0 0



1 1 1 1

1 1 1 1

0 1 0 0

0 0 1 0 1 1 1 1 0 1 0 0

0 0 0 1 1 1 1 1 0 1 0 0

0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 1

1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1

0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0

0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 1

1 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0

0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0

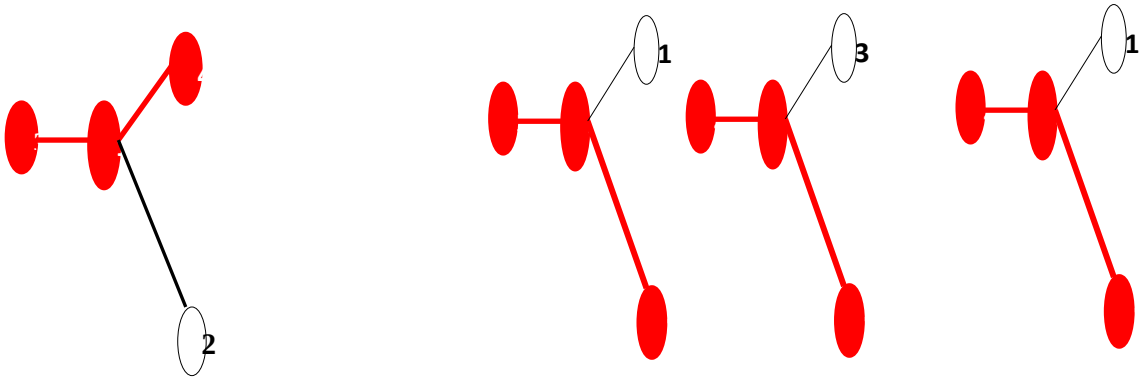


FIG. III.10 –trouver tout les isomorphismes

III-5- L'algorithme VF2 : SSR : Algorithme de Cordella 2001 (VF2)

Un algorithme plus récent a la fois pour l'isomorphisme de graphe et de sous graphe est l'algorithme VF. Cet algorithme résout avec une approche directe ces problèmes, en utilisant une heuristique qui est basée sur l'analyse des ensembles des nœuds adjacents a ceux déjà pris en compte dans l'appariement partiel. Cette heuristique est rapide à calculer amenant dans de nombreux cas à une amélioration significative a l'algorithme d'Ullmann et bien d'autres L'algorithme VF reprend a la base les grandes lignes de celui d'Ullmann.

L'algorithme VF2 [08]

C'est un algorithme qui permet de trouver le plus grand mapping **M** entre les nœuds de $\boldsymbol{G_1}$ et les nœuds de $\boldsymbol{G_2}$. Un tel mapping $\boldsymbol{M \sqsubset V_1 \times V_2}$ est un isomorphisme si **M** est une fonction bijective qui préserve la structure des arcs des deux graphes.
M est donc un ensemble des couples de nœuds **(n,m)** tels que $\boldsymbol{n \epsilon G_1}$ et $\boldsymbol{m \epsilon G_2}$ (**n** correspond à **m**). Les différentes étapes de la recherche d'un tel mapping peuvent être représentées sous la forme d'états : chaque état s du matching de graphe peut être associé à une solution partielle de mapping, **M(s)**, sous-ensemble de **M. M(s)** détermine donc univoquement deux sous-graphes, $\boldsymbol{G_1(s) = \big(M_1(s); E_1(s)\big) et\ G_2(s) = (M_2(s), M_2(s))}$ obtenus en ne prenant que les nœuds de $\boldsymbol{G_1}$ et $\boldsymbol{G_2}$ qui sont dans **M** et les arcs reliant ces nœuds dans $\boldsymbol{G_1}$ et $\boldsymbol{G_2}$, respectivement.

Dès lors, le passage d'un état **S** à un état $\boldsymbol{S'}$ représente donc l'ajout d'un couple **(n,m)** de nœuds mappés. Pour réduire le nombre d'états possibles, l'algorithme possède un ensemble de règles de faisabilité qui élimineront les états qui ne respectent pas la règle générale :

$$\boldsymbol{F(s,n,m) = F_{syn}(s,n,m) \wedge F_{sem}(s,n,m)}$$

Où $\boldsymbol{F_{syn}(s,n,m)}$ sont les règles syntaxiques (ne dépendant donc que de la structure) et

$F_{sem}(s, n, m)$ Sont les règles sémantiques (ne dépendant donc que des attributs). Ces, règles seront expliquées plus en détails, plus loin.

III-5-1-Le pseudo-code de l'algorithme [08]

L'algorithme 1 contient le pseudo-code de l'algorithme **VF2** : détaillons-le. Au départ, le matching est vide, c'est-à-dire : $M(S_0) = \emptyset$ avec S_0 comme état initial. Pour chaque état intermédiaire, l'algorithme calcule un ensemble, **P(s)**, qui va contenir toutes les paires de nœuds candidates à être matchées. Après avoir calculé cet ensemble, l'algorithme parcours chaque paire de **P(s)** et regarde si cette paire respecte les règles de faisabilité (cf. l'équation ci-dessus). Si la paire respecte les règles, l'algorithme calcule le nouvel état et appelle la fonction **Match** sur ce nouvel état.

Il est à noter que l'algorithme pourrait créer deux fois le même état en passant par des étapes différentes. C'est pourquoi un ordre total (arbitraire), noté $\prec$, sur les nœuds de qui sont dans **P(s)** est défini. Dès lors, étant donné que l'ordre d'insertion des paires dans le matching n'influe pas sur le résultat final, l'algorithme ne considère pas les paires $p = (n_i, m_j) \epsilon$ P(s) si le matching contient déjà un nœud m_k , tel que $m_k \prec m_k$.
Ceci permettra à l'algorithme de ne générer chaque état qu'une et une seule fois.

III-5-2-Algorithm 1 Match(s)

Input: un état intermédiaire s ; l'état initial s0 possède M(s0) = ;

Output: le mapping entre les deux graphes

if M(s) couvre tous les noeuds de G2 **then**

return M(s)

else

Calculer l'ensemble P(s) des paires candidates à insérer dans M(s)

for all p dans P(s) do

if p satisfait les règles de faisabilité pour l'inclusion dans M(s) **then**

Calculer l'état suivant, s'

Appeler Match(s')

end if

end for

end if

FIG. III.11 : L'algorithme VF2

III-5-3-Le calcul de P(s) [08]

Pour le calcul de P(s), quatre ensembles sont construits :

– $T_1^{out}(s)$ est l'ensemble des nœuds qui n'appartiennent pas à **M(s)** et qui sont les destinations des arcs partant de $G_1(s)$

– $T_2^{out}(s)$ est construit de la même manière, avec $G_2(s)$

– $T_1^{in}(s)$ est l'ensemble des nœuds qui n'appartiennent pas à **M(s)** et qui sont les origines des arcs arrivant dans $G_1(s)$

– $T_2^{in}(s)$ est construit de la même manière, avec $G_2(s)$

Donc, l'ensemble **P(s)** contiendra les paires **(n,m)** telle que $n \in T_1^{out}(s)$ et $m \in T_2^{out}(s)$. Si l'un des deux ensembles est vide, les noeuds n et m seront pris, respectivement, dans $T_1^{in}(s)$ et $T_1^{in}(s)$. Lorsque le graphe est non-connecté, ces deux derniers ensembles peuvent être également vides. Dans ce cas, P(s) sera en fait toutes les paires de nœuds **(n, m)** telles que $n \notin G_1(s)$ et $m \notin G_2(s)$.

III-5-4-Les règles de faisabilité [08]

Une paire candidate n'est acceptée, c'est-à-dire n'est insérée dans le matching partiel **M(s)**, que si elle respecte les règles de faisabilité. Rappelons la formule générale de ces règles :

$$F(s,n,m) = F_{syn}(s,n,m) \wedge F_{sem}(s,n,m)$$

Nous allons donc, dans un premier temps, détailler $F_{syn}(s,n,m)$ et, ensuite, $F_{sem}(s,n,m)$.

La formule qui détermine la règle syntaxique, $F_{syn}(s,n,m)$ est définie par cinq sous-règles : R_{pred} , R_{succ} , R_{in} , R_{out} et R_{new} Les deux premières vérifient la consistance de la solution partielle à laquelle on ajoute une paire **(n,m).** Les trois suivantes servent à élaguer l'arbre de recherche, c'est-à-dire à la réduction de la taille de l'arbre en enlevant les nœuds qui ne peuvent pas faire partie de la solution. Pour le détail de ces cinq formules, une série de notions et notations doivent être introduites. Étant donné un graphe **G = (V ; E)** et un nœud **n ϵ V**, **Pred(G; n)** et **Succ (G; n)** dénotent, respectivement, l'ensemble des prédécesseurs et l'ensemble des successeurs de n. Également, les ensembles

$T_1(s) = T_1^{in}(s) \sqsubset T_1^{out}(s)$ **Et** $\tilde{\upsilon}_1(s) = V_1 - M_1(s) - T_1(s)$ seront utilisés dans les définitions des cinq règles (T_2 et e $\tilde{\upsilon}_2$ sont définis de la même manière).

$R_{pred}(s,n,m) \Leftrightarrow$

$$(\forall n' \in M_1(s) \cap Pred(G_1, n): \exists m' \in pred\ (G_2, m) | (n', m') \epsilon M(s)) \wedge \forall m' \in M_2(s) \cap Pred(G_2, m): \exists n' \in pred\ (G_1, n) | (n', m') \epsilon M(s))$$

$$R_{succ}(s, n, m) \Leftrightarrow (\forall n' \in M_1(s) \cap succ(G_1, n): \exists m' \in succ\ (G_2, m) | (n', m') \epsilon M(s)) \wedge \forall m' \in M_2(s) \cap succ(G_2, m): \exists n' \in succ\ (G_1, n) | (n', m') \epsilon M(s))$$

$$R_{in}(s, n, m) \Leftrightarrow (card\ (succ\ G_1, n) \sqcap {T_1}^{in}(s) \geqslant card\ (succ\ G_2, m) \sqcap {T_2}^{in}(s))) \wedge card\ (pred\ G_1, n) \sqcap {T_1}^{in}(s) \geqslant card\ (pred\ G_2, m) \sqcap {T_2}^{in}(s)))$$

$$R_{out}(s, n, m) \Leftrightarrow (card\ (succ\ G_1, n) \sqcap {T_1}^{out}(s) \geqslant card\ (succ\ G_2, m) \sqcap {T_2}^{out}(s))) \wedge card\ (pred\ G_1, n) \sqcap {T_1}^{out}(s) \geqslant card\ (pred\ G_2, m) \sqcap {T_2}^{out}(s)))$$

$$R_{new}(s, n, m) \Leftrightarrow (card\ \left(\tilde{\upsilon}_1(s) \cap Pred(G_1, n)\right) \geqslant card\ \left(\tilde{\upsilon}_2(s) \cap Pred(G_2, m)\right) \wedge card\ (\tilde{\upsilon}_1(s) \cap succ(G_1, n)) \geqslant card(\tilde{\upsilon}_2(s) \cap succ(G_2, m)))$$

Intuitivement, les deux premières règles vérifient la consistance de la solution partielle **M (s')** (obtenue en ajoutant la paire **(n;m)** à la solution partielle **M(s)).** Les deux suivantes sont nécessaires pour élaguer l'arbre de recherche en regardant un coup en avance. Enfin, la dernière est également utile pour élaguer l'arbre de recherche, mais cette fois-ci en regardant 2 coup en avance.

Pour la formule de la règle sémantique, il faut définir un opérateur d'égalité, ≈, sur les nœuds et les arcs des graphes. Cet opérateur, pour pouvoir comparer deux noeuds, pourrait utiliser les attributs de ceux-ci (comme, par exemple, le label du noeud). Une fois un tel opérateur défini, $F_{sem}(s, n, m)$ peut être défini aisément :

$$F_{sem}(s, n, m) \Leftrightarrow n \approx m$$
$$(\forall (n', m') \in M(s), ((n, n',) \epsilon E_1 \Longrightarrow (n, n') \approx (m, m') \wedge (\forall (n', m') \in M(s), ((n', n) \epsilon E_1 \Longrightarrow (n', n) \approx (m', m)$$

Intuitivement, cette règle de sémantique, va regarder plusieurs choses :

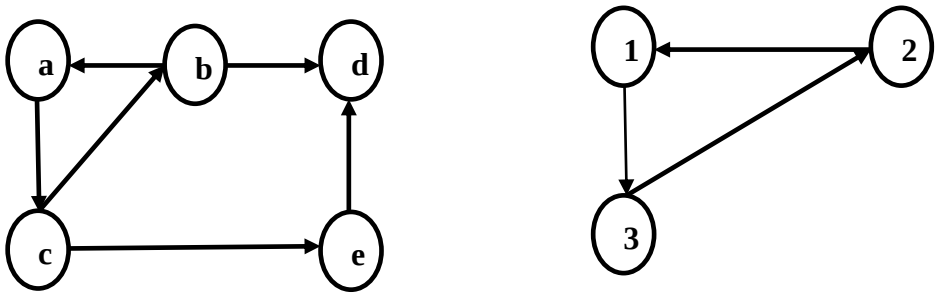
1. Les deux nœuds **n** et **m** doivent être similaires ;

2. Pour chaque paire $(\boldsymbol{n}', \boldsymbol{m}')$ déjà dans le matching, si l'arc $((\boldsymbol{n}, \boldsymbol{n}',)$ (ou $((\boldsymbol{n}', \boldsymbol{n})$ existe, alors il y a une correspondance entre l'arc $((\boldsymbol{n}, \boldsymbol{n}',)$ $((\boldsymbol{n}', \boldsymbol{n})$ et l'arc $(\boldsymbol{m}, \boldsymbol{m}')$ $(\boldsymbol{m}', \boldsymbol{m})$.

III-6-VF2 : Exemple de notations [15]

Les 2 graphiques dessus sont$\boldsymbol{G_1}$ et $\boldsymbol{G_2}$

Dans le algoritm **VF2 Je** vais essayer de faire correspondre chaque nœud $\boldsymbol{G_1}$ avec un nœud dans $\boldsymbol{G_2}$**.**



$\boldsymbol{G_1}$ $\boldsymbol{G_2}$

FIG. III.12 : deux graphes non-orientés

étape 1 :

Je correspondre $\boldsymbol{G_1}$ vide avec vide $\boldsymbol{G_1}$**:** il fonctionne toujours

M(s)=(a,1)

étape 2:

Je peux correspondre $\boldsymbol{V_1} \in \boldsymbol{G_1}$ avec $\boldsymbol{V_2} \in \boldsymbol{G_2}$

Je correspondre **a,1** sorcière »: il fonctionne toujours

$$\boldsymbol{M_1(s) = \{a\}, M_2(s) = \{1\}}$$

$$T_1{}^{in}(s) = \{b\}, T_1{}^{out}(s) = \{c\}$$

$$T_2{}^{in}(s) = \{2\}, T_2{}^{out}(s) = \{3\}$$

P(s)=(c,3) mapping :c,3

Pred(G_1,c)={a},Succ(G_1,c)={b,e}

Pred(G_2,3)={1},Succ(G_2,3)={2}

faisabilité : true

F(s, c, 3)=Vrai

$M(S') = \{(a, 1), (c, 3)\}$

$$M_1(s') = \{a, c\}, M_2(s) = \{1, 3\}$$

$$T_1{}^{in}(s) = \{b\}, T_1{}^{out}(s) = \{b, e\}$$

$$T_2{}^{in}(s) = \{2\}, T_2{}^{out}(s) = \{2\}$$

P(s)=(b,2),(e,2)

Pred(G_1,e)={c},Succ(G_1,e)={d}

Pred(G_2,e)={3},Succ(G_2,e)={1}

***(e,2)** viole le prédicat $R_{succ}(S', e, 2)$. **De fait:**

$1 \in M_2(s) \cap succ(G_2, 2)$ OR $succ(G_2, e) \cap M_1(s) = \emptyset$

*On arrive donc à l'appriement : **M(s'') = (a,1),(c,3),(b,2)**

Et l'on a terminé puisque l'on couvre tous les sommets de G_2

Conclusion

Le problème principal de la mise en correspondance de graphes par détection de cliques maximales est sa complexité en temps. Pour deux graphes $\boldsymbol{G_M}$ et $\boldsymbol{G_D}$ avec $\boldsymbol{n}$ et $\boldsymbol{m}$ sommets respectivement, la taille du graphe d'association dépend fortement au nombre d'étiquettes dans $\boldsymbol{G_M}$ et $\boldsymbol{G_D}$. Si $\boldsymbol{n \leq m}$ et les étiquettes dans les deux graphes sont différentes, ce qui correspond aux meilleurs cas, la complexité sera seulement en $\boldsymbol{O(nm+n^2)}$. Cependant, dans le pire des cas, quand les étiquettes dans les deux graphes sont identiques, la complexité augmente alors jusqu'à $\boldsymbol{O(nm^2))}$.

La complexité théorique de la recherche d'isomorphismes de graphes n'est, à ce jour, pas complètement établie. Il n'a jamais été montré que le problème appartenait à la classe des problèmes *NP*-complet mais aucun algorithme de résolution de complexité polynomiale n'a été trouvé en 1979 par Garey. Il existe des algorithmes polynomiaux pour certains graphes particuliers. Par contre, le problème de la recherche d'un isomorphisme de sous-graphes est un problème *NP*-complet [Alt, 1984] et exponentiel pour les graphes généraux [Messmer, 1995]. Lorsque les graphes sont planaires, la complexité devient polynomiale (cette situation est assez fréquente en traitement d'images).

Chapitre IV

Implémentation

Sommaire

Introduction

L'algorithme Ullmann est le plus populaire pour résoudre le problème de l'isomorphisme de sous-graphe. Malgré qu'il date de 1976, il est encore largement utilise. Les problèmes abordés par l'algorithme sont l'isomorphisme de graphes, l'isomorphisme de sous-graphes et le monomorphisme.

L'algorithme VF2 est une très bonne amélioration de l'algorithme Ullmann

Les auteurs proposent une modification de l'implémentation de l'algorithme, appelé il réduit les besoins, en mémoire de $O(N^2)$ à O(N), ce qui rend l'algorithme particulièrement intéressant pour travailler avec des graphes de grande taille.

Notre travail consiste à implémente ces deux algorithmes puis effectuer plusieurs tests pour confirmer pratiquement les résultats cites ci-dessus

IV.1-Petit historique du langage java

On peut faire remonter la naissance de Java à 1991. À cette époque, des ingénieurs de chez SUN ont cherché à concevoir un langage applicable à de petits appareils électriques (on parle de *code embarqué*). Pour ce faire, ils se sont fondés sur une syntaxe très proche de celle de C++, en reprenant le concept de machine virtuelle déjà exploité auparavant par le Pascal UCSD. L'idée consistait à traduire d'abord un programme source, non pas directement en langage machine, mais dans un pseudo langage universel, disposant des fonctionnalités commune sà toutes les machines. Ce code intermédiaire, dont on dit qu'il est formé de *byte codes*, se trouve ainsi compact et portable sur n'importe quelle machine ; il suffit simplement que cette dernière dispose d'un programme approprié (on parle alors de *machine virtuelle*) permettant de l'interpréter dans le langage de la machine concernée.**[21]**

IV.2-Environnement de programmation

L'environnement de développement de notre application est le java éclipse.

Eclipse est un environnement de développement (IDE) historiquement destiné au langage Java, même si grâce à un système de plugins il peut également être utilisé avec d'autres langages de programmation, dont le C/C++ et le PHP.

Eclipse nécessite une machine virtuelle Java (JRE) pour fonctionner. Mais pour compiler du code Java, un kit de développement (JDK) est indispensable. JRE et JDK sont disponibles sur le site officiel d'Oracle : Java SE Downloads **[Net, 02]**

Le JDK (*Java Developpement Kit*) de Java est livré, en standard, avec différentes bibliothèques ou "paquetages" ou "API" (*Application Programming Interface*) fournissant de nombreuses classes utilitaires**[Net, 02]**

IV.3-Présentation de java éclipse

Eclipse est un projet, décliné et organisé en un ensemble de sous-projets de développements logiciels, de la Fondation Eclipse visant à développer un environnement de production de logiciels libre qui soit extensible, universel et polyvalent, en s'appuyant principalement sur Java. **[21]**

Son objectif est de produire et fournir des outils pour la réalisation de logiciels, englobant les activités de programmation (notamment environnement de développement intégré et Framework) mais aussi d'AGL recouvrant modélisation, conception, testing, gestion de configuration, vise notamment à supporter tout langage de programmation à l'instar de Microsoft Visual Studio.

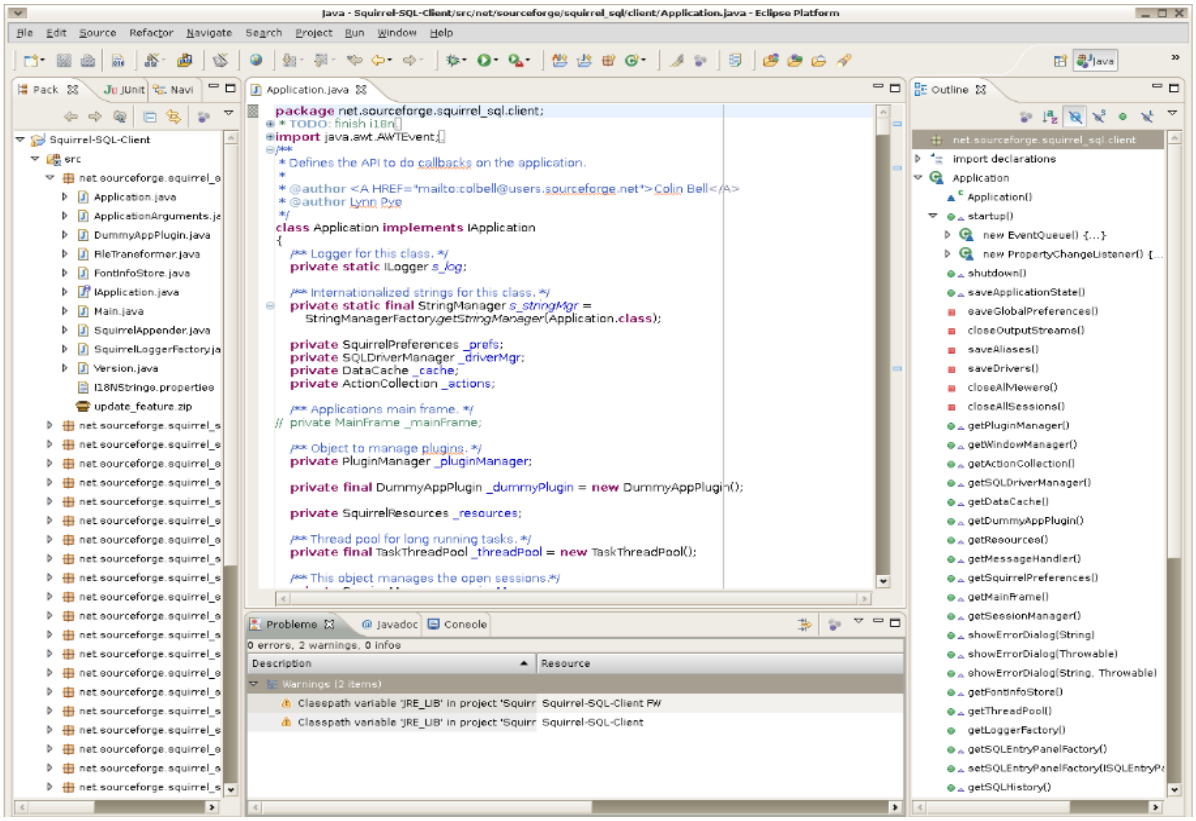


FIG.IV.01- interface de java [Net, 02]

IV.4-Java et la programmation orientée objet

La P.O.O. (programmation orientée objet) possède de nombreuses vertus universellement reconnues désormais. Notamment, elle ne renie pas la programmation structurée (elle se fonde sur elle), elle contribue à la fiabilité des logiciels et elle facilite la réutilisation de code existant. Elle introduit de nouveaux concepts, en particulier ceux d'objets, d'encapsulation, de classe et d'héritage. **[21]**

IV.5-L'interface de java éclipse

IV.5.1- Java et la programmation événementielle

IV.5.1.1- Interface console ou interface graphique

Actuellement, on peut distinguer deux grandes catégories de programmes, en se fondant sur leur *interface* avec l'utilisateur, c'est-à-dire sur la manière dont se font les échanges d'informations entre l'utilisateur et le programme :

• les programmes à interface console,

• les programmes à interface graphique. **[21]**

IV.5.1.2- Les programmes à interface console (ou en ligne de commande)

Historiquement, ce sont les plus anciens. Dans de tels programmes, on fournit des informations à l'écran sous forme de lignes de texte s'affichant séquentiellement, c'est-à-dire les unes à la suite des autres. Pour fournir des informations au programme, l'utilisateur frappe des caractères au clavier (généralement un "écho" apparaît à l'écran).

Entrent dans cette catégorie :

• les programmes fonctionnant sur PC sous DOS ou, plus fréquemment, dans une fenêtre DOS de Windows,

• les programmes fonctionnant sous Unix ou Linux et s'exécutant dans une "fenêtre de commande".

Avec une interface console, c'est le programme qui décide de l'enchaînement des opérations : l'utilisateur est sollicité au moment voulu pour fournir les informations demandées**[21]**

IV.5.1.3- Les programmes à interface graphique (G.U.I.)

Dans ces programmes, la communication avec l'utilisateur se fait par l'intermédiaire de *composants* tels que les menus déroulants, les menus surgissants, les barres d'outils ou les boîtes de dialogue, ces dernières pouvant renfermer des composants aussi variés que les boutons poussoirs, les cases à cocher, les boutons radio, les boîtes de saisie, les listes déroulantes... **[21]**

L'utilisateur a l'impression de piloter le programme, qui semble répondre à n'importe laquelle de ses demandes. D'ailleurs, on parle souvent dans ce cas de *programmation événementielle,* expression qui traduit bien le fait que le programme réagit à des évènements provoqués (pour la plupart) par l'utilisateur.

On notera que le terme G.U.I. (*Graphical User Interface*) tend à se généraliser pour désigner ce genre d'interface. Manifestement, il met en avant le fait que, pour permettre ce dialogue, on ne peut plus se contenter d'échanger du texte et qu'il faut effectivement être

capable dedessiner, donc d'employer une interface graphique. Il n'en reste pas moins que l'aspect le plus caractéristique de ce type de programme est dans l'aspect événementiel1. **[21]**

IV.6- Les fenêtres associées à un programme

IV.6.1- Cas d'une interface console

L'interface console n'utilise qu'une seule fenêtre (dans certains anciens environnement, la fenêtre n'était même pas visible, car elle occupait tout l'écran). Celle-ci ne possède qu'un petit nombre de fonctionnalités : déplacement, fermeture, parfois changement de taille et défilement. **[21]**

IV.6.2- Cas d'une interface graphique

L'interface graphique utilise une fenêtre principale qui s'ouvre au lancement du programme. Il est possible que d'autres fenêtre apparaissent par la suite : l'exemple classique est celui d'un logiciel de traitement de texte qui manipule différents documents associés chacun à une fenêtre. **[21]**

L'affichage des informations dans ces fenêtres ne se fait plus séquentiellement. Il est généralement nécessaire de prendre en compte l'aspect "coordonnées". En contrepartie, on peut afficher du texte en n'importe quel emplacement de la fenêtre, utiliser des polices différentes jouer sur les couleurs, faire des dessins, afficher des images...

3.3 Java et les interfaces

IV.7- La gestion des interfaces graphiques est intégrée dans Java

Dans la plupart des langages, on dispose d'instructions ou de procédures standard permettant de réaliser les entrées-sorties en mode console. **[21]**

IV.8- Applications et applets

À l'origine, Java a été conçu pour réaliser des *applets* s'exécutant dans des pages *Web*. En fait, Java permet d'écrire des programmes indépendants du Web. On parle alors d'*applications* (parfois de "vraies applications").

Les fonctionnalités graphiques à employer sont quasiment les mêmes pour les applets et les applications. D'ailleurs, dans cet ouvrage, nous présenterons l'essentiel de Java en considérant des applications. Un seul chapitre sera nécessaire pour présenter ce qui est spécifique aux applets. **[21]**

Théoriquement, une applet est faite pour que son code (compilé) soit téléchargé dans une page Web. Autrement dit, il peut sembler indispensable de recourir à un navigateur pour l'exécuter (pas pour la compiler). En fait, quel que soit l'environnement, vous disposerez toujours d'un visualisateur d'applets vous permettant d'exécuter une applet en dehors du Web.

IV.9- la bibliothèque JUNG

JUNG (the Java Universal Network/Graph Framework) est un framework de modélisation, d'analyse et de visualisation de graphe. C'est un logiciel libre, sous licence BSD, écrit en Java. **[Net, 03]**

JUNG pour Java Universal Network/Graph Framework est une plateforme logicielle contenant une bibliothèque de programmes Java pour générer des graphes de visualisation de réseaux. Cette plateforme est également disponible sur sourceforge.net. **[25]**

IV.10-Le matériel utilisé

Nous avons travaillé sur un PC de type Intel qui possède comme caractéristique :

- ✓ Un processeur Intel ® core ™ i3-2348MCPu @2.30 Ghz 2.30 Ghz.
- ✓ Type de système : Système d'exploitation 32 bits.
- ✓ Et 2.00 (1.81 Go utilisable) de mémoire vive (RAM).

IV.11-Application

IV.11.1-Objectif

L'objectif de notre travail est une étude la comparative des méthodes appariement de graphe .
On doit implémente des deux puis une comparaison pratique dit être effectueé les méthodes

IV.11.2-Schéma générale de l'application

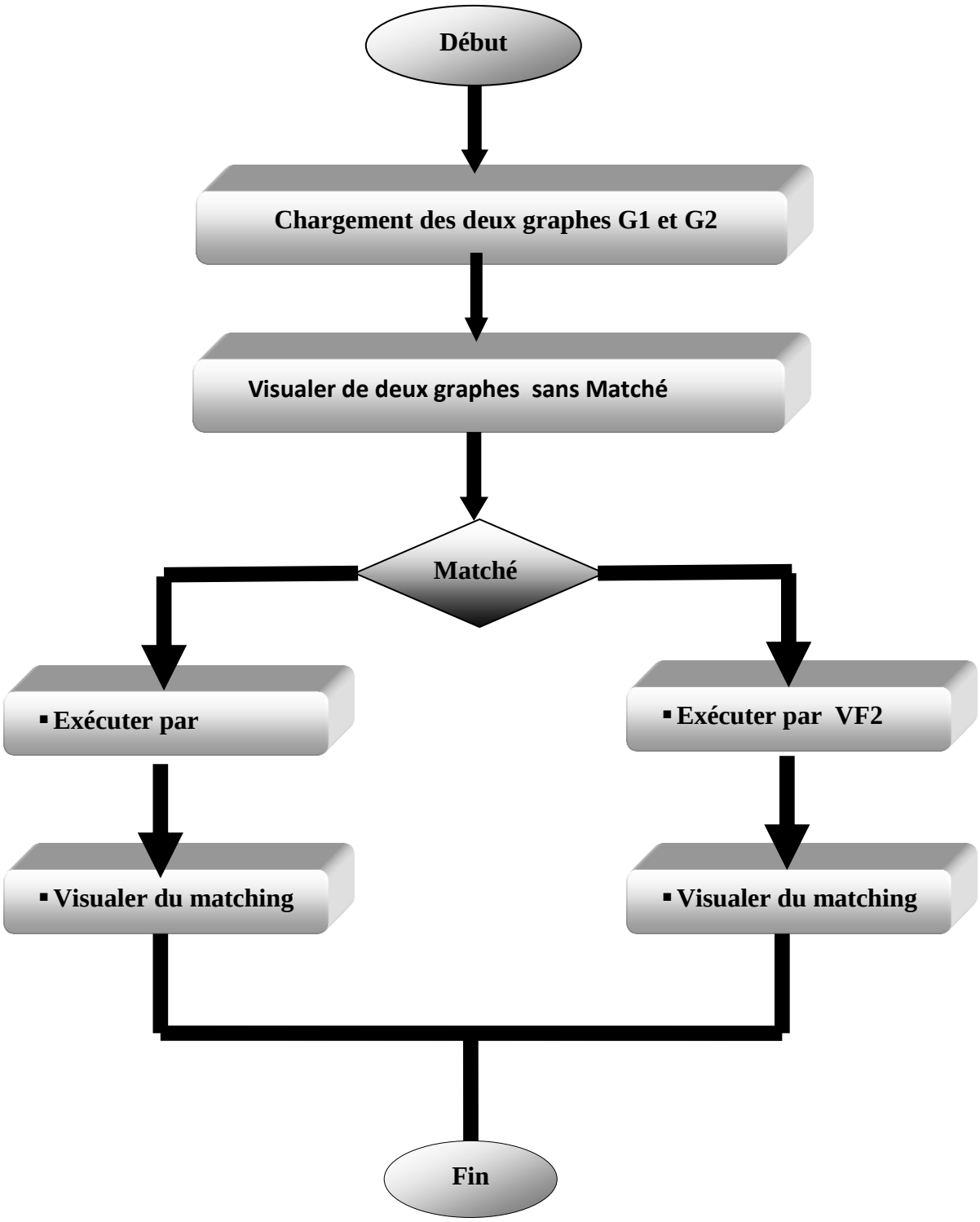


Figure IV.2 *:* Schéma général de l'application.

IV.11.3-Présentation de l'application

IV.11.3.1_Interface principale

L'interface de notre application est représenté sur figure ci-dessus

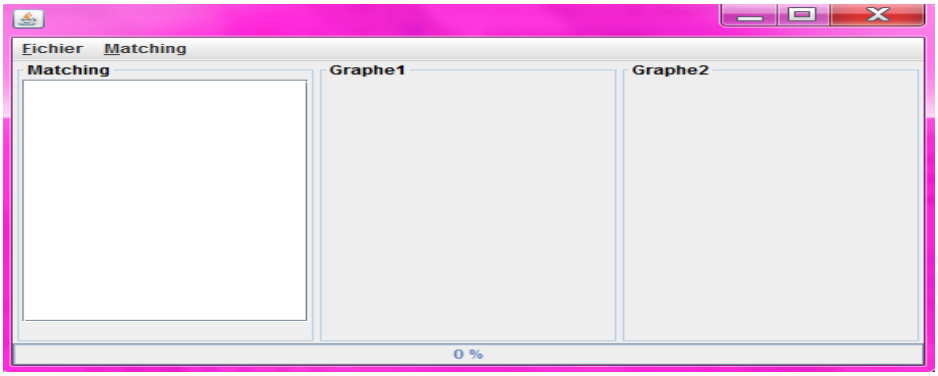


Figure IV.3 : Interface principale de l'application.

IV.11.3.2-Différentes fonctionnalités de l'application

IV.11.3.2.1-Chargement de graphe 1 et 2

Graphe 1				Graphe 2		
0	1	0	0	0	0	1
1	0	1	1	0	0	1
0	1	0	0	1	1	0
0	1	0	0			

Il est possible de charger une graphe grâce au menu fichier → graphe1 puis graphe 2 → à partir d'un fichier.

Exemple 01 : G1=4 sommet ; G2=3 sommet

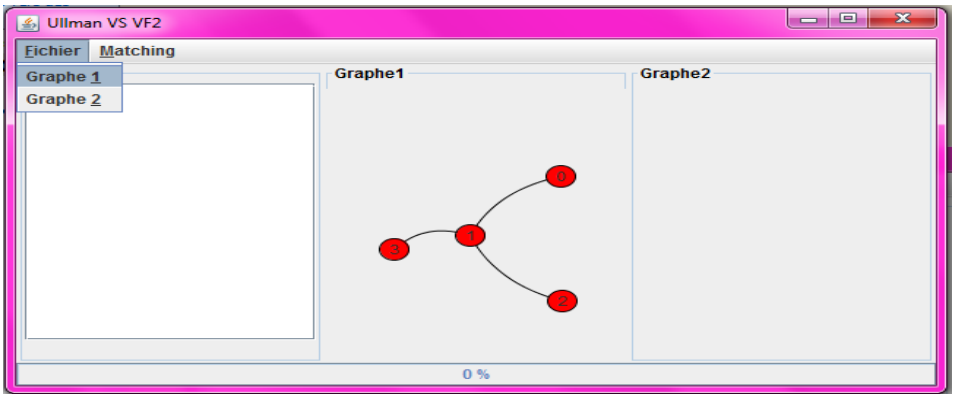


Figure IV.4 : Chargement d'un graphe 1.

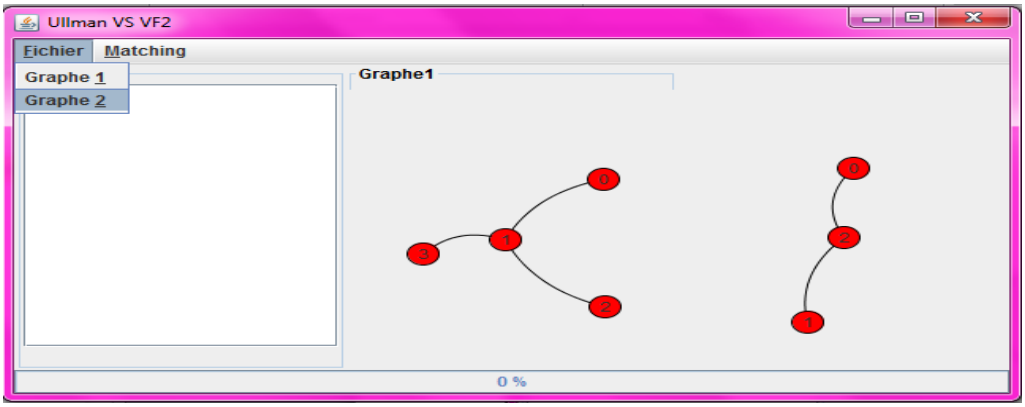


Figure IV.5 : Chargement d'un graphe 2.

IV.11.3.2.2-Exécution par Ullmann

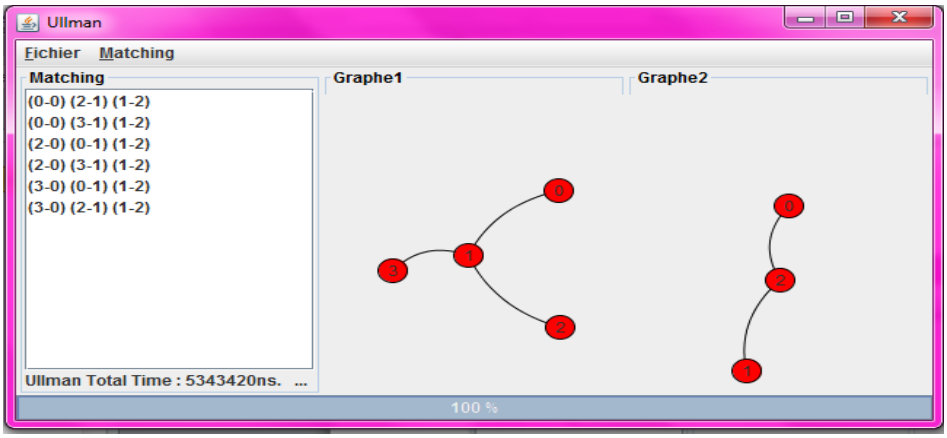


Figure IV.6: résultat d'Ullmann

Défèrent matching

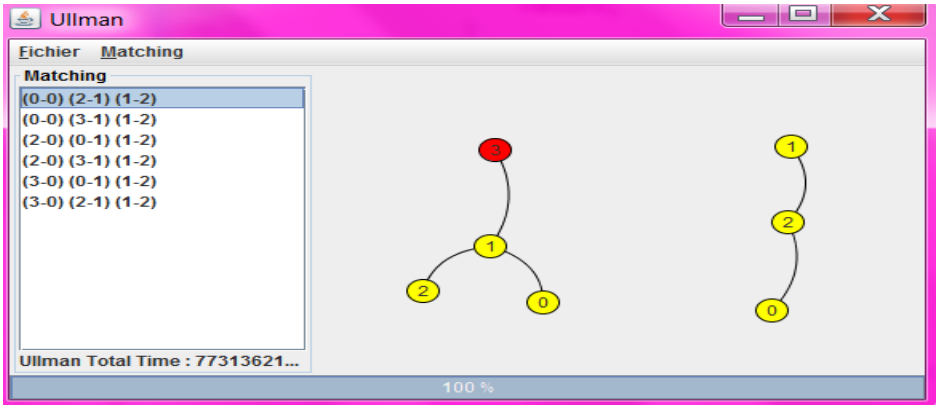


Figure IV.7 : *résultat Ullmann*

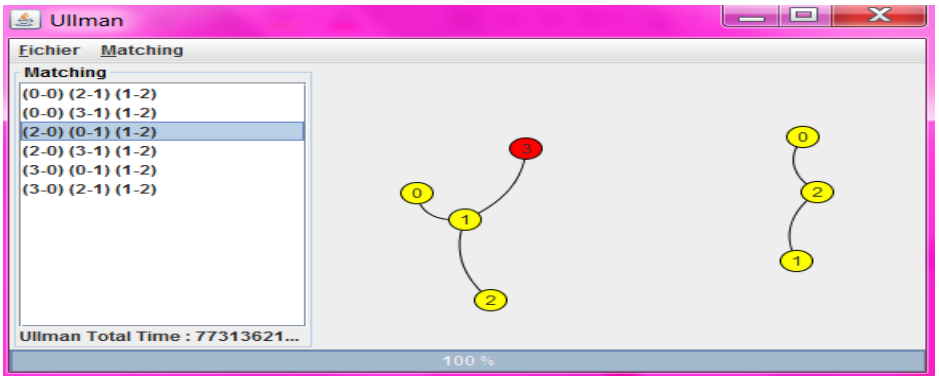


Figure IV.8 *résultat Ullmann en choix matchig*

IV.10.3.2.3-Exécution par VF2

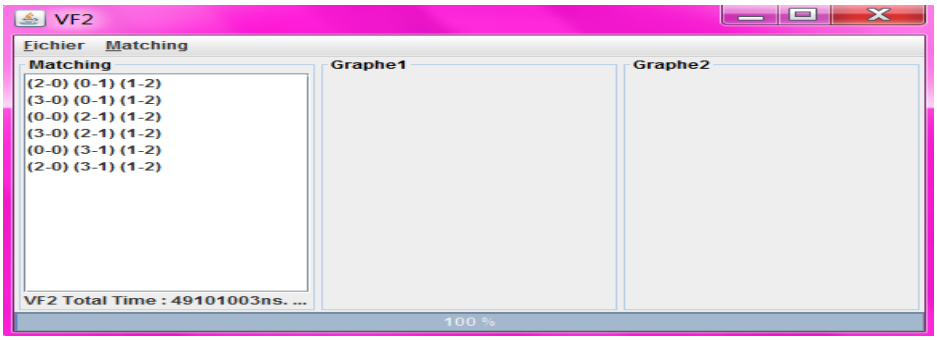


Figure IV.9 : résultat VF2

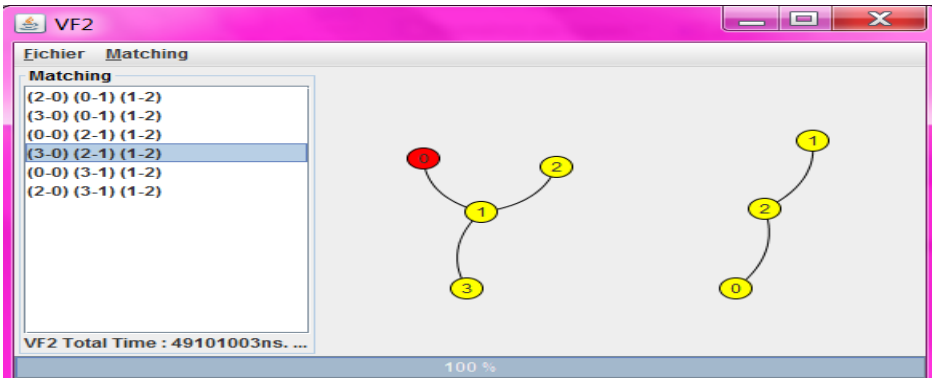


Figure IV.10 ***résultat VF2 en choix matchig***

IV.11.3- Exemple 2 : G1=8 sommet ; G2=4 sommet

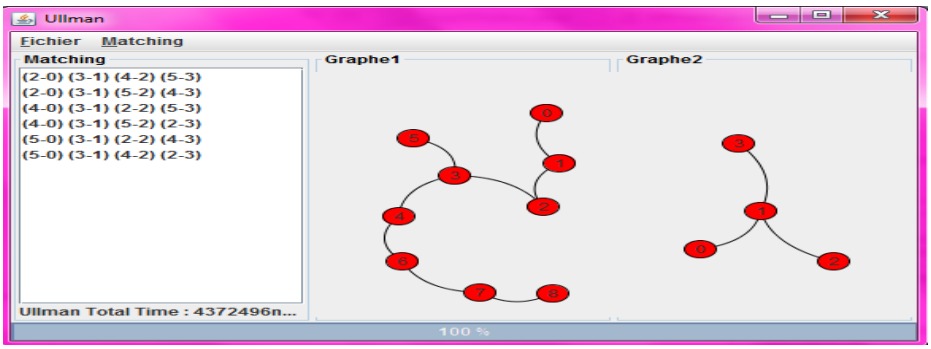


Figure IV.11*:* résultat d'Ullmann

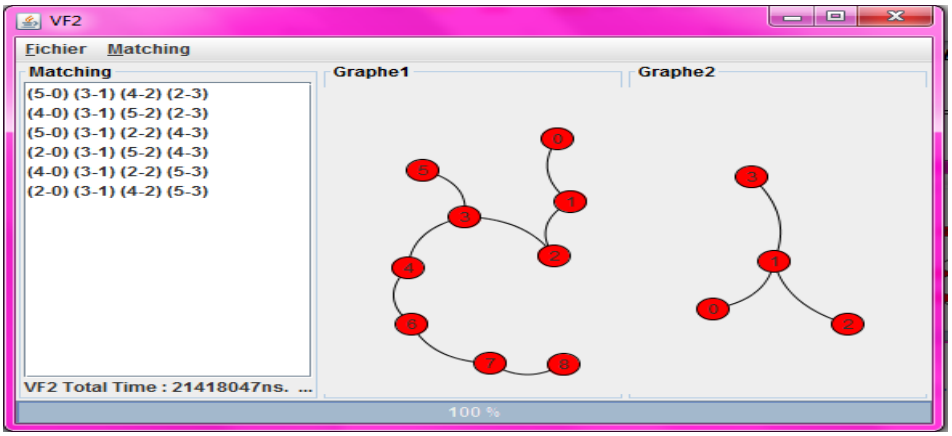


Figure IV.12 ***résultat VF2***

IV.11.4- *Exemple 03 :G1=18 ; G2=5*

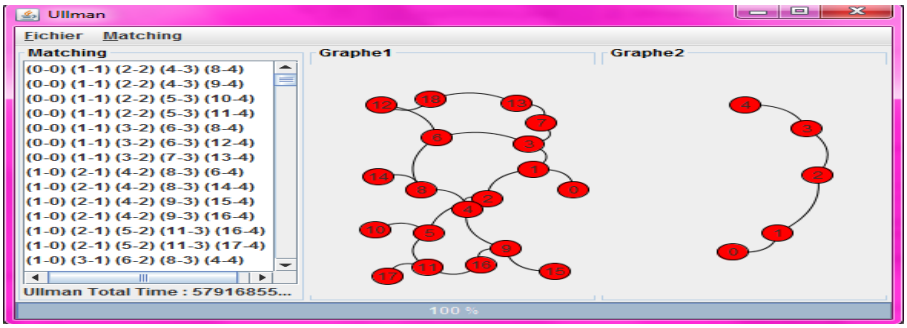


Figure IV.13 : *résultat Ullmann*

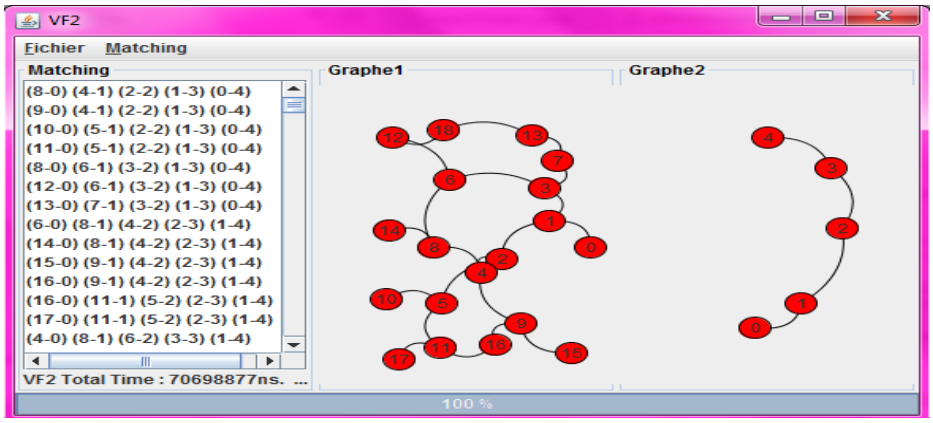


Figure IV.14 : *résultat VF2*

IV.11.5-*Exemple 04 :G1= 24 sommet et G2= 08 sommet*

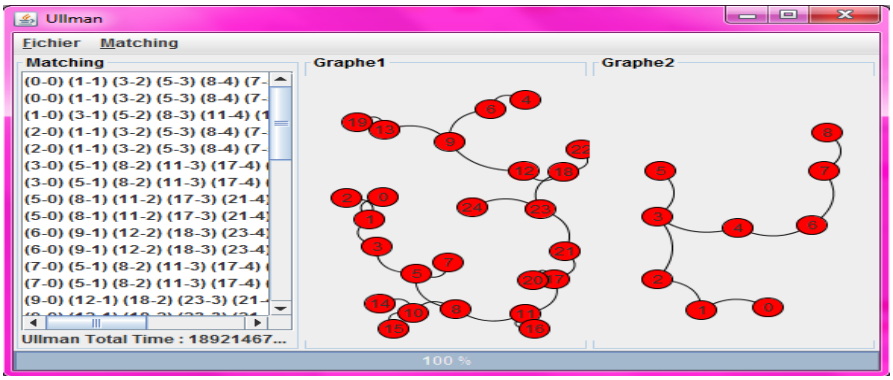


Figure IV.15 : *résultat Ullmann*

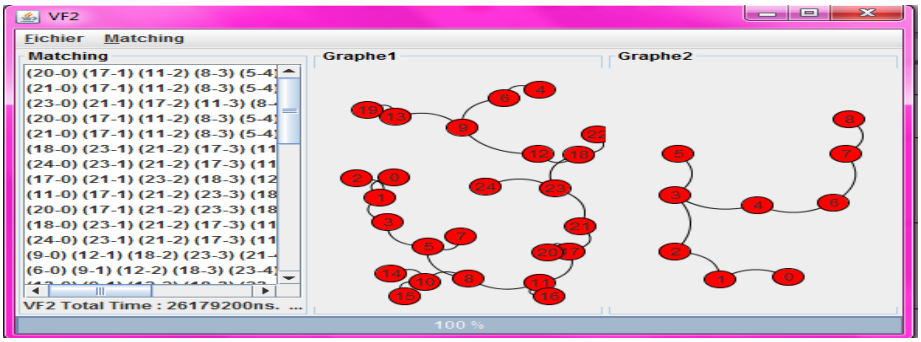


Figure IV.16: ***résultat VF2***

IV.11.6-Etude comparative

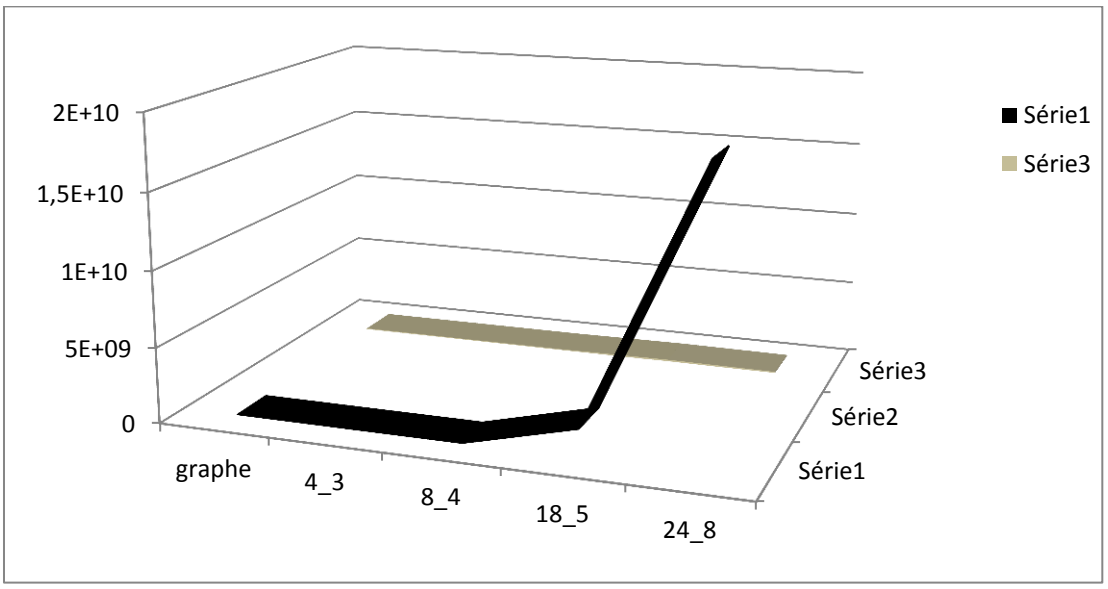
Après l'exécution de notre application sur les jeux d'exemple présente sur cette figure .nous vous obtenus le résultat de la **Figure IV.18**

Exemple	**Graphes**	**ULLMANN**	**VF2**
01	G1 : 4 sommets	Temps1-a : 534320 ns	Temps1-b : 491000
	G2 : 03 sommets		
02	G1 : 08 sommets	Temps2-a : 4372466	Temps2-b : 2141804
	G2 : 04 sommets		
03	G1 : 18	Temps3-a : 1892146766 ns	Temps3-b : 70697788 ns
	G2 : 05		
04	G1 : 24	Temps4-a : 18921467756 ns	Temps4-b : 26179200 ns
	G2 : 08		

Figure IV.17 : Etude comparative entre Ullmann et VF2

IV.11.7- Résultat en graphiquement

graphe	4_3	8_4	18_5	24_8
ULLMANN	534320	4372466	1892146766	18921467756
VF2	491000	2141804	70697788	26179200



- **Ullmann**
- **VF2**

Figure IV.18 : l'algorithme VF2 soit bien meilleur que celle de l'algorithme d'Ullmann

Après l'évaluation des performances de l'algorithme Ullmann et de l'algorithme VF2 en comparant la complexité en temps d'exécution par rapport au nombre de nœuds des graphes, nous concluions que, bien que la performance de l'algorithme VF2 soit bien meilleure que celle de l'algorithme d'Ullmann

Conclusion

- Ce dernier chapitre présente notre implémentation étude comparative des deux méthodes (Ullmann, VF2) à travers un qu'un ensemble de tests sur différents graphe l'origine et sous-graphe, tout en discutant et commentant les résultats obtenus.

Conclusion Générale

Dans notre mémoire nous nous sommes intéressés au problème d'appariement de graphes en général et plus particulièrement à l'appariement exact. Nous avons étudié plusieurs algorithmes de mise en correspondance de graphes dans la théorie des graphes tels que l'algorithme d'isomorphisme de graphes et de sous-graphes. Les deux algorithmes les plus intéressants et les plus souvent utilisés dans différents domaines applicatifs ou de recherche sont Ullmann et VF2.

Après l'évaluation des performances de l'algorithme Ullmann et de l'algorithme VF en comparant la complexité en temps d'exécution par rapport au nombre de nœuds des graphes, nous concluions que, bien que la performance de l'algorithme VF soit bien meilleure que celle de l'algorithme d'Ullmann, nous avons retenu trois limites importantes :

1)- les deux algorithmes ne s'intéressent qu'à la structure du graphe alors que les propriétés de nœud et de liens ajoutent des bonnes caractéristiques.

2)- Ces algorithmes utilisent les appels de fonction récursive donc sont souvent complexe gourmant en mémoire.

3)- La mise en correspondance de graphe est exacte tandis que les problèmes d'appariement sont souvent flous et inexacts, par exemple dans la reconnaissance d'objets.

Dans la théorie de complexité le problème d'isomorphisme de graphes et de sous-graphes est un problème NP-complet, c'est-adire très difficile à résoudre en temps polynomial. Donc comme perspective à court terme on envisage d'appliquer les méta-heuristiques telles que les algorithmes génétiques et le PSO pour résoudre le problème d'appariement inexact de graphes.

[01]	BARATLI KARIM	Conception et implémentation d'un outil de filtrage de netlist pour un système de prototypage rapide Décembre 2012 (17/12/2012)
[02]	LEF`EVRE Jonas	Isomorphisme de graphes.pdf avril 2009
[03]	Thomas BÄRECKE	ISOMORPHISME INEXACT DE GRAPHES PAR OPTIMISATION ÉVOLUTIONNAIRE Il Soutenue publiquement le 22 octobre 2009
[04]	NGUYEN Thi Hong Hiep	Appariement multivoque de graphes par la recherche locale *(15 Novembre 2009)*
[05]	Adel hlaoui	Contribution en appariement de graphe pour la recherche d' image par le contenu (mai 2004)
[06]	Ali idarrou	Entreposage de document multimédia : comparaison de structures (30-03-2013)
[07]	Justine Lebrun	Appariement inexact de graphes applique _a la recherche d'image et d'objet 3D (22 Nov 2011)
[08]	Prof. Thierry Massart	Distribution de code DSL avec contraintes . 2010 - 2011
[09]	DESROSIERS Christian	TECHNIQUES POUR L'EXPLORATION DE DONN´EES STRUCTUR´EES ET POUR LA D´ECOUVERTE DE CONNAISSANCES EN TH´EORIE DES GRAPHES

AVRIL 2008

[10] Graphes et Appariement d'Objets Complexes

25 Janvier 2011

[11] Rashid Jalal QURESHI Reconnaissance de formes et symboles graphiques complexes dans les images de documents

4 mars 2008

[12] Florence Tupin Graphes en traitement d'images et en reconnaissance des formes

[13] LA THÉORIE DES GRAPHES

2005

[14] Salim Jouili, Salvatore Tabbone Applications des graphes en traitement d'images

Tunisie.28 Oct 2008

[15] Luc Brun Interprétation d'images II

Reconnaissance structurelle de formes/objets

[16] Nicolas Sidère Contribution aux méthodes de reconnaissance structurelle de formes : approche à base de projection de graphes

24 Février 2012

[17] Didier Müller Introduction à la théorie des graphes

[18] Mme Sabine DE BLIECK Gastton Lagrafe arrive,, Faites Graphe a vous!!

7 défis pour découvrir la théorie des graphes

UCL-FSA - 2010

[19] Moultazem Ghazal Contribution _a la gestion des données géographiques :

Modélisation et interrogation par croquis

Submitted on 22 Jul 2010

[20] Aline Deruyver La représentation des images par les graphes

-Laboratoire Icube, equipe BFO

[21] Cay S. Horstmann et Gary Cornell — Au cœur de Java™ volume 1 Notions fondamentales. 2008

[22] Gondran M., Minoux M. — Graphes et Algorithmes. Eyrolles, 1995.

[23] Lévy G. — Algorithmique Combinatoire. Dunod, 1994.

[24] Wolf-Dieter — Graph Matching – Algorithms

Pattern Recognition and Image Processing Group (PRIP)

Institute of Computer Graphics and Algorithms

May 11, 2011

[25] Ingénieur d'étude CNRS — Visualisation de l'information Mai 2006

Sites internet :

[Net, 01] https://fr.wikipedia.org/wiki/Th%C3%A9orie_des_graphes consulter 24-12-2015 12:46

[Net, 02] https://fr.wikipedia.org/wiki/Eclipse_(projet) consulter 20-01-2016

[Net, 03] https://fr.wikipedia.org/wiki/JUNG Consulter 29-05-2016