# Dominance au voisinage pour le problème d'isomorphisme de sous-graphe *

**G. Audemard[1] C. Lecoutre[1] M. Samy-Modeliar[1] G. Goncalves[2] D. Porumbel[2]**

[1] CRIL-CNRS, UMR 8188, [2] LGI2A,
Université d'Artois, France

**Résumé**

Cet article propose une approche de filtrage générale, SND en abrégé pour *Scoring-based Neighborhood Dominance*, pour le problème d'isomorphisme de sous-graphe. En raisonnant sur des propriétés de dominance entre sommets basées sur diverses fonctions de score et de voisinage, SND apparaît comme un puissant mécanisme de filtrage. Nous étudions une spécialisation de SND basée sur le nombre de chemins de longueur $k$ comme fonction de score ainsi que sur trois manières de considérer le voisinage. Avec cette spécialisation, nous montrons que SND est plus fort que LAD et incomparable à SAC. Notre étude expérimentale montre que SND atteint dans la plupart des cas les mêmes performances en terme de filtrage que SAC tout en étant plus rapide de plusieurs ordres de grandeurs. Cela permet de résoudre le problème d'isomorphisme de sous-graphe en étant beaucoup plus efficace que MAC et légèrement meilleur que LAD.

## 1 Introduction

La littérature scientifique comporte de nombreuses références à des objets représentés sous forme de graphes (arbres, réseaux, cartes combinatoires, etc.) Dans les applications où ce type de concepts est utilisé, on a souvent besoin de trouver des associations entre graphes. On parle alors du problème d'isomorphisme de (sous-)graphe, ou bien de l'une de ses généralisations, comme par exemple, l'épimorphisme, le couplage de graphes, l'isomorphisme correcteur d'erreurs, etc. [2, 3]

Mis à part le problème classique d'isomorphisme de graphe qui est de complexité indéterminée, la plupart des autres problèmes sont NP-difficiles. D'un point de vue théorique, des efforts importants ont été réalisés afin de déterminer les classes de complexité exactes de ces différents problèmes (citons par exemple le prix Fulkerson concernant les travaux portant sur les graphes de degré borné [14]). De manière générale, les résultats de complexité et les algorithmes (polynomiaux) proposés pour de nouveaux problèmes d'isomorphisme constituent un domaine de recherche très actif [7, 8]. Enfin, de nombreuses applications autour de ces problèmes ont été proposées au fil des années.

Historiquement, les premiers algorithmes de recherche d'isomorphismes de (sous-)graphe remontent aux années 70 [6, 22]. De manière schématique, ces méthodes construisent un couplage entre sommets de manière récursive et exploitent des techniques de filtrage de type "*look-ahead*" pour éliminer certaines valeurs incompatibles. Toutefois, des progrès importants ont été réalisés au cours des années. Par exemple, VF2 [5] montre une accélération exponentielle comparée à l'approche initiale proposée par Ullmann. On peut également mentionner les méthodes basées sur le calcul des groupes d'automorphisme de graphe ou encore les techniques de raffinement par coloriage (voir par exemple l'algorithme de Weisfeiler-Lehman [23]) qui sont utilisées pour partitionner les sommets par rapport à certains invariants (par exemple leurs degrés).

Les méthodes incomplètes basées sur des (méta-)heuristiques sont également très populaires pour résoudre des problèmes difficiles ou de grande taille. Elles sont par exemple très largement utilisées dans la littérature (très vaste) liée aux problèmes de *matching* de graphes (des centaines de références sont données dans [2, 3]). Par ailleurs, certains de ces algorithmes

*Ce travail a été financé par le BQR (Bonus Qualité Recherche) de l'Université d'Artois

peuvent être adaptés au cas du pur problème d'isomorphisme de (sous-)graphe. Par exemple, l'heuristique définie dans [17] essaie de minimiser la mesure de similarité entre deux graphes, et atteindre la valeur 0 correspond en conséquence à trouver un isomorphisme.

En ce qui concerne les méthodes complètes, une direction de recherche consiste à utiliser des techniques de filtrage issus de la programmation par contraintes (voir [21, 24, 16] pour des développements récents). En effet, le problème d'isomorphisme de (sous-)graphe peut facilement être transformé en un problème de satisfaction de contraintes. On peut alors exploiter des propriétés liées aux contraintes et réseaux de contraintes comme par exemple la cohérence d'arc (AC) ou la singleton cohérence d'arc (SAC) [13]. Afin de renforcer le processus d'inférence, des techniques de filtrage spécifique au problème d'isomorphisme de (sous-)graphes ont été proposées, comme par exemple ILF [24] et LAD [21] qui utilisent des informations liées aux degrés, des multi-ensembles ou encore une relation de voisinage pour partitionner les sommets et ainsi filtrer les domaines.

Dans cet article, nous proposons une approche générale pour raisonner sur la dominance entre sommets selon deux axes principaux : tout d'abord, en associant un ensemble de scores à chaque couple de sommets et en considérant ensuite diverses façons de définir le voisinage entre sommets. Nous nommons cette approche SND pour *Scoring-based Neighborhood Dominance*. Nous montrons que LAD est un cas particulier de SND et nous étudions des spécialisations de SND en considérant le nombre de chemins de longueur $k$ dans les graphes et trois façons de relier les ensembles de sommets. Avec ces spécialisations, nous montrons que SND est plus fort que LAD et incomparable à SAC. Enfin, l'étude expérimentale que nous avons menée démontre l'efficacité de notre approche.

## 2 PPC et isomorphisme de sous-graphe

Nous commençons cette section par quelques rappels concernant la programmation par contraintes (PPC), et poursuivons avec la description du problème d'isomorphisme de sous-graphe ainsi que sa modélisation en PPC.

### 2.1 Définitions préliminaires et terminologie PPC

Un *réseau de contraintes* $P$ est composé d'un ensemble fini de variables, noté $\text{vars}(P)$, et d'un ensemble fini de contraintes, noté $\text{cons}(P)$. Chaque *variable* $x$ possède un *domaine* qui est l'ensemble fini des valeurs pouvant être assignées à $x$. Le domaine initial d'une variable $x$ est noté $dom^{init}(x)$ alors que le domaine courant de $x$ est noté $dom(x)$ ; on a toujours $dom(x) \subseteq dom^{init}(x)$. Chaque *contrainte* $c$ implique un ensemble ordonné de variables, appelé la *portée* (*scope*) de $c$ et noté $scp(c)$. Une interprétation $I$ d'un ensemble ordonné de variables $X = \{x_1, \ldots, x_r\}$ est un ensemble $\{(x_1, a_1), \ldots, (x_r, a_r)\}$ tel que $\forall i \in 1..r, a_i \in dom^{init}(x_i)$ ; chaque $a_i$ est noté $I[x_i]$. Il est important de noter que chaque contrainte $c$ est sémantiquement définie par une *relation* qui contient l'ensemble des interprétations autorisées pour la portée de $c$ ; on dit que ces interprétations satisfont la contrainte $c$. Une *solution* d'un réseau de contraintes $P$ est une interprétation $I$ de $\text{vars}(P)$ telle que chaque contrainte $c \in \text{cons}(P)$ est satisfaite par $I$ (restreinte aux variables de $scp(c)$). L'ensemble des solutions d'un réseau de contraintes $P$ est noté $sols(P)$.

L'espace de recherche initial d'un réseau de contraintes $P$ est exactement le produit cartésien des domaines initiaux des variables de $P$. Trouver une solution dans cet espace est une tâche NP-difficile. Heureusement, il est possible de réduire l'espace de recherche en supprimant des valeurs connues pour être incohérentes (c'est-à-dire ne pouvant mener à aucune solution), tout en préservant l'ensemble des solutions. Cela est réalisé au moyen d'algorithmes de filtrage qui peuvent être appelés en séquence jusqu'à ce qu'un point fixe soit atteint, représentant un processus appelé *propagation de contraintes*.

Un tel processus cherche de manière itérative des valeurs qui ne respectent pas une propriété cible appelée *cohérence*. L'une des plus connues est l'arc-cohérence (AC) pour les contraintes binaires, appelée pour des raisons historiques *Generalized AC* (GAC) pour les contraintes non-binaires. Elle est définie ainsi. Une interprétation $I$ est *valide* ssi $\forall (x, a) \in I, a \in dom(x)$. Une interprétation $I$ de $scp(c)$ est un *support* de $c$ ssi $I$ est une interprétation valide satisfaisant $c$. Si $I$ est un support de $c$ tel que $x \in scp(c)$ et $I[x] = a$, on dit que $I$ est un support de $c$ pour $(x, a)$. Une contrainte $c$ est *arc-cohérente* (GAC) ssi $\forall x \in scp(c), \forall a \in dom(x)$, il existe au moins un support de $c$ pour $(x, a)$. Un réseau de contraintes est GAC si toutes ses contraintes sont GAC.

GAC correspond au niveau de filtrage maximal que l'on peut atteindre lorsqu'on considère les contraintes de manière indépendante. Si $P$ est un réseau de contraintes alors $GAC(P)$ représente le réseau de contraintes obtenu à partir de $P$ en supprimant itérativement toutes les valeurs détectées sans support d'au moins une contrainte de $P$. Bien entendu, il est possible de filtrer davantage en étendant la portée du raisonnement local. L'une des cohérences plus fortes que GAC est la singleton arc-cohérence (SAC) [9]. Si $P$ est un réseau de contraintes et $(x, a)$ un couple tel

que $x \in scp(c)$ et $a \in dom(x)$ alors $P|_{x=a}$ est le réseau de contraintes obtenu à partir de $P$ en restreignant le domaine de $x$ à $a$. Une valeur $(x, a)$ est *singleton arc-cohérente* (SAC) ssi $GAC(P|_{x=a}) \neq \bot$, signifiant qu'aucun domaine n'est devenu vide après l'affectation de $x$ à $a$ et l'application de GAC. Un réseau de contraintes est SAC si toutes ses valeurs sont SAC.

En utilisant la terminologie donnée dans [10], un processus de filtrage $\phi$ est considéré plus fort qu'un autre processus $\psi$ ssi $\phi$ supprime toujours au moins autant de valeurs que $\psi$ (pour tout réseau de contraintes), et $\phi$ est strictement plus fort que $\psi$ si en plus $\phi$ supprime plus de valeurs que $\psi$ pour au moins un réseau de contraintes. On dit aussi que $\phi$ et $\psi$ sont incomparables si aucun des deux n'est plus fort que l'autre.

### 2.2 Modélisation PPC du problème d'isomorphisme de sous-graphe

Nous rappelons d'abord quelques définitions classiques concernant les graphes. Un graphe non-orienté $G = (V, E)$ est défini par un ensemble $V$ de *sommets* et un ensemble $E \subseteq V^2$ d'*arêtes*. Nous considérons uniquement des graphes simples, i.e., il ne peut y avoir au maximum qu'une arête $\{v, v'\} \in E$ entre tout couple de sommets $v, v' \in V$ et pour tout sommet $v$, on a $\{v, v\} \notin E$. Pour chaque sommet $v \in V$, $\Gamma(v)$ désigne l'ensemble des sommets *adjacents* à $v$, i.e., $\Gamma(v) = \{v' \in V : \{v, v'\} \in E\}$.

Une instance du problème d'isomorphisme de sous-graphe est définie par un graphe motif (pattern graph) $G_p = (V_p, E_p)$ et un graphe cible (target graph) $G_t = (V_t, E_t)$, l'objectif consistant à déterminer si $G_p$ est isomorphe à un sous-graphe de $G_t$. Trouver une solution à une instance de ce problème consiste à trouver une fonction sous-isomorphe qui est un couplage injectif $f : V_p \to V_t$ tel que chaque arête de $G_p$ soit préservée : $\forall (v, v') \in E_p, (f(v_p), f(v'_p)) \in E_t$. Notez que dans cet article nous traitons du problème d'isomorphisme de sous-graphe partiel et pas induit.

Nous introduisons maintenant une façon classique et naturelle de représenter une instance de ce problème sous forme d'un réseau de contraintes $P$. Tout d'abord, une variable $x_{v_p}$ est introduite dans vars($P$) pour chaque sommet $v_p$ du graphe motif tel que le domaine de $x_{v_p}$ est exactement l'ensemble $V_t$ : si $x_{v_p}$ est affectée à la valeur $v_t$, cela veut alors dire que le sommet $v_p$ du graphe motif est couplé au sommet $v_t$ du graphe cible. Ensuite, une contrainte globale allDiff(vars($P$)) est introduite et garantit l'injection, chaque sommet motif devant être lié à un seul et unique sommet du graphe cible. Enfin, pour chaque arête $(v_p, v'_p)$ du graphe motif, une contrainte binaire en extension $\{x_{v_p}, x_{v'_p}\} \in E_t$ garantit que cette arête est associée à une arête du graphe cible. Clairement, l'ensemble des solutions du réseau de contraintes $P$ est exactement l'ensemble des fonctions sous-isomorphes.

Il existe d'autres possibilités pour modéliser en PPC le problème d'isomorphisme de sous-graphe, mais celui introduit ci-dessus est celui qui est habituellement utilisé (voir [22, 15, 19, 12, 24, 21]). Des travaux ont été réalisés au cours des années pour résoudre toujours plus efficacement les instances de ce problème, en proposant notamment différents niveaux de filtrage. Par exemple, une forme partielle d'arc-cohérence, appelée *Foward Checking* a été utilisée dans [22, 15]. Plus tard, J.-C. Régin [19] a utilisé GAC sur la contrainte allDiff et AC sur les contraintes binaires en extension. Plus récemment, il y a eu plusieurs tentatives pour combiner les contraintes et filtrer ainsi plus de valeurs. Dans [12], en plus de GAC, les auteurs proposent de raisonner, en considérant le domaine courant des variables, sur la cardinalité des ensembles de voisins de chaque sommet. Nous notons LV2002 ce niveau de filtrage (comme cela est fait dans [21]). Un filtrage, nommé ILF (*Iterated Labeling Filtering*) a été introduit dans [24]. L'idée est d'exploiter la structure des graphes motif et cible afin d'associer une étiquette globale à chaque sommet. Une relation de compatibilité est définie sur ces étiquettes et peut être utilisée pour supprimer d'un domaine d'une variable $x_{v_p}$ chaque sommet $v_t$ tel que l'étiquette associée à $v_t$ n'est pas compatible avec celle associée à $v_p$. ILF est un processus itératif initialisé avec une étiquette de départ. On peut, par exemple, choisir le degré des sommets comme étiquette : on note ILF$^{deg}$ la méthode de filtrage qui en découle. A chaque itération, la nouvelle étiquette d'un sommet est le multi-ensemble composé des étiquettes des voisins du sommet. En raisonnant sur les multi-ensembles ordonnés, on peut alors détecter et supprimer des valeurs incohérentes. ILF$^{dom}$ utilise un autre étiquetage itératif qui intègre les domaines courants dans la relation de compatibilité [24]. Enfin, LAD est une méthode de filtrage [21] qui équivaut à ajouter $n_p \times n_t$ contraintes au réseau de contraintes et à établir GAC. Pour chaque sommet motif $v_p$ et chaque sommet cible $v_t$, il y a une contrainte implicite, notée $\texttt{LAD}(v_p, v_t)$, ajoutée à cons($P$) tel que la portée est $\{x_{v_p}\} \cup \{x_{v'_p} \mid v'_p \in \Gamma(v_p)\}$ et la sémantique est :

$$x_{v_p} = v_t \Rightarrow \\ allDiff(\{x_{v'_p} \mid v'_p \in \Gamma(v_p)\}) \wedge \ x_{v'_p} \in \Gamma(v_t), \forall v'_p \in \Gamma(v_p) \quad (1)$$

Cette contrainte établit que si $v_p$ est couplée à $v_t$ alors chaque sommet motif adjacent à $v_p$ doit être couplé à un sommet distinct adjacent à $v_t$. En pratique, ces contraintes de "voisinage" ne sont pas réellement ajoutées au réseau de contraintes, et un algorithme de

filtrage spécifique est utilisé pour atteindre un niveau de filtrage équivalent à GAC. Notons que l'algorithme de filtrage introduit dans [12] correspond à une forme partielle de GAC sur les contraintes LAD : il assure que le nombre de sommets adjacents à $v_p$ est plus petit ou égal au nombre de sommets du graphe cible qui sont adjacents à $v_t$ et appartiennent à au moins un domaine d'une variable dans $\{x_{v'_p} \mid v'_p \in \Gamma(v_p)\}$.

## 3 Dominance (renforcée) au voisinage

### 3.1 Principe et exactitude

Nous proposons de généraliser le raisonnement sous-jacent à LAD selon deux axes : tout d'abord, en associant un ensemble de scores à chaque couple de sommets d'un graphe et ensuite en considérant différentes formes de voisinage (contrairement à LAD, ces ensembles ne correspondent pas nécessairement aux sommets adjacents d'un sommet donné). Nous nommons cette approche SND (pour Scoring-based Neighborhood Dominance).

SND peut donc être vu comme une approche générique pouvant être paramétrée par deux éléments notés $\mathcal{S}$ pour le score et $\mathcal{N}$ pour le voisinage. Une spécialisation $\text{SND}_{\mathcal{S},\mathcal{N}}$ de SND est donc définie par un ensemble de fonctions de score $\mathcal{S}$ tel que $S \in \mathcal{S}$ donne pour chaque couple de sommets $(v, v')$ d'un graphe un score $S(v, v')$ (défini par un nombre entier ou réel calculé de manière adéquate) et un ensemble de fonctions de voisinage $\mathcal{N}$ tel que chaque $N \in \mathcal{N}$ donne pour chaque sommet $v$ d'un graphe un ensemble $N(v)$ de sommets du même graphe.
Comme c'est le cas pour LAD, pour chaque couple $(v_p, v_t)$ de sommets motif/cible, on considère une contrainte implicite. Elle est notée $\text{SND}_{\mathcal{S},\mathcal{N}}(v_p, v_t)$, sa portée est $\{x_{v_p}\} \bigcup_{N \in \mathcal{N}} \{x_{v'_p} \mid v'_p \in N(v_p)\}$, et sa sémantique est définie comme suit :

$$\begin{aligned} x_{v_p} = v_t \Rightarrow \\ \bigwedge_{N \in \mathcal{N}} \Big( allDifferent(\{x_{v'_p} \mid v'_p \in N(v_p)\}) \\ \wedge\ x_{v'_p} \in N(v_t), \forall v'_p \in N(v_p) \\ \wedge\ S(v_p, v'_p) \leq S(v_t, x_{v'_p}), \forall v'_p \in N(v_p), \forall S \in \mathcal{S} \Big) \end{aligned} \tag{2}$$

Lorsque la dernière condition de l'équation 2 n'est pas vérifiée (dans le contexte d'une fonction de voisinage $N$), i.e. $S(v_p, v'_p) > S(v_t, v'_t)$ où $v'_t = x_{v'_p}$, on dit alors que $(v_p, v'_p)$ n'est pas dominé par $(v_t, v'_t)$. Si il n'y a pas d'interprétation possible de $\{x_{v'_p} \mid v'_p \in N(v_p)\}$ telle que chaque couple $(v_p, v'_p)$ soit dominé, on peut alors conclure que $x_{v_p} \neq v_t$. Bien entendu, la façon dont on calcule les scores et la façon dont on définit les voisinages doivent être cohérentes par rapport au réseau de contraintes. En d'autres termes, chaque contrainte $\text{SND}_{\mathcal{S},\mathcal{N}}(v_p, v_t)$ ajoutée à $\text{cons}(P)$ doit être une conséquence de $P$. Nous dirons qu'une contrainte de la forme $\text{SND}_{\mathcal{S},\mathcal{N}}(v_p, v_t)$ est *impliquée* (par rapport à $P$) ssi $sols(P) = sols(P \oplus \text{SND}_{\mathcal{S},\mathcal{N}}(v_p, v_t))$ où $P \oplus c$ est le réseau de contraintes $P$ avec la contrainte additionnelle $c$.

Dans cet article, nous proposons des combinaisons originales de fonctions de score et de voisinage. D'un coté, nous proposons d'utiliser le nombre de chemins entre deux sommets comme fonction de score. En effet, il est bien connu que si $M_G$ est la matrice d'adjacence associée à un graphe $G$, alors $M^k_G[v][v']$ indique le nombre de chemins de longueur $k$ dans $G$ qui relient $v$ à $v'$. L'idée sous-jacente est qu'il n'est pas possible d'associer un couple de sommets du graphe motif $(v_p, v'_p)$ à un couple $(v_t, v'_t)$ du graphe cible si pour un certain $k$ on a $M^k_{G_p}[v_p][v'_p] > M^k_{G_t}[v_t][v'_t]$. En conséquence, cette observation peut être utilisée lors du filtrage. Par simplicité et abus de notation $M^k$ désignera la fonction de score $S$ tel que pour chaque couple de sommets $(v, v')$ d'un graphe $G$, on a $S(v, v') = M^k_G[v][v']$. Désormais, nous ne considérons que ces fonctions de score (appelées fonctions de *longueur*). Dans ce contexte, $\mathcal{M}$ désignera l'ensemble de toutes les fonctions de longueur, i.e. $\mathcal{M} = \{M^1, M^2, \ldots\}$.

D'un autre coté, nous proposons trois fonctions de voisinage définies de la manière suivante sur un graphe $G = (V, E)$ :

- `Id` est la fonction identité définie par $\texttt{Id}(v) = \{v\}, \forall v \in V$. Combinée à $\mathcal{M}$ elle peut être utilisée pour filtrer les domaines en considérant le nombre de chemins menant d'un sommet à lui-même.
- $\Gamma$ est la fonction (définie précédemment) qui retourne l'ensemble des sommets adjacents à un sommet donné. C'est le voisinage utilisé par LAD.
- `all` est la fonction qui retourne l'ensemble des sommets du graphe excepté le sommet d'entrée : $\texttt{all}(v) = V \setminus \{v\}, \forall v \in V$. Elle est complémentaire à `Id`.

Après avoir observé que LAD est un cas particulier de SND, nous prouvons la correction de SND avec les fonctions de longueur et les fonctions de voisinage `Id`, $\Gamma$, et `all`.

**Remarque 1** *LAD est équivalent à $SND_{\mathcal{S},\mathcal{N}}$, avec $\mathcal{S} = \emptyset$ et $\mathcal{N} = \{\Gamma\}$.*

**Proposition 1** *Chaque contrainte $SND_{\mathcal{S},\mathcal{N}}(v_p, v_t)$, avec $\mathcal{S} \subseteq \mathcal{M}$ et $\mathcal{N} \subseteq \{\texttt{Id}, \Gamma, \texttt{all}\}$ est impliquée.*

*Preuve.* Supposons tout d'abord que $\mathcal{N} = \{\Gamma\}$. D'après [21], nous savons que chaque contrainte LAD

est impliquée. La seule différence entre $\texttt{SND}_{\mathcal{S},\{\Gamma\}}$ et LAD, est que, si $M^k \in \mathcal{S}$ alors cette fonction empêche de coupler un noeud $v'_p$ adjacent à $v_p$ à un noeud $v'_t$ adjacent à $v_t$ quand le nombre de chemins de longueur $k$ de $v_p$ à $v'_p$ est strictement plus grand que le nombre de chemins de longueur $k$ allant de $v_t$ à $v'_t$. Clairement, si $M^k_{G_p}[v_p][v'_p] > M^k_{G_t}[v_t][v'_t]$, il n'y a pas de fonction sous-isomorphe qui relie $v_p$ à $v_t$ sachant que $v'_p$ est relié à $v'_t$. Aussi, la restriction imposée par $M^k$ garantit que la contrainte est impliquée. Supposons maintenant que $\mathcal{N} = \{\texttt{Id}\}$, et que $M^k \in \mathcal{S}$. Le sommet $v_p$ ne peut être couplé à $v_t$ lorsque le nombre de chemins de longueur $k$ allant de $v_p$ à lui-même est strictement supérieur au nombre de chemins de longueur $k$ allant de $v_t$ à lui-même. Si $M^k_{G_p}[v_p][v_p] > M^k_{G_t}[v_t][v_t]$, il n'y a pas de fonction sous-isomorphe qui permette de relier $v_p$ à $v_t$. Donc, la contrainte est impliquée. La preuve est similaire pour $\mathcal{N} = \{\texttt{all}\}$, et peut être étendue à tout sous-ensemble de $\{\texttt{Id}, \Gamma, \texttt{all}\}$. □

Pour nos expérimentations décrites plus loin, nous utilisons les fonctions de voisinage définies ci-dessus. Notons toutefois qu'il existe d'autres possibilités. On peut, par exemple, raisonner sur des voisinages définis comme étant les ensembles de sommets à distance 1 ou 2, à distance 1 et 2, et ainsi de suite. De plus, à la place des fonctions de longueur nous pouvons exploiter d'autres fonctions de score, basées par exemple sur le nombre de cliques ou de cycles. L'étude de ces alternatives dépasse le cadre de cet article, et est une perspective de notre travail à moyen terme.

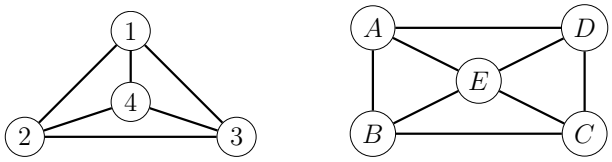
### 3.2 Filtrage des contraintes SND

Le filtrage des contraintes SND dans le but d'atteindre GAC est similaire à ce qui a été proposé pour les contraintes allDiff [18] et LAD. Ceci est basé sur le concept de couplage couvrant dans un graphe biparti. Un *couplage* dans un graphe est un sous-ensemble d'arêtes qui ne partagent aucun sommet. Un couplage *couvre* un ensemble de sommets ssi chacun d'eux apparaît dans une arête du couplage. Un graphe biparti est un graphe $G = (V, E)$ où $V$ est partitionné en deux sous-ensembles $V_1$ et $V_2$ tel que chaque arête de $E$ est associée à un sommet de $V_1$ et à un sommet de $V_2$, pour mettre l'accent sur la partition, nous notons $V = V_1|V_2$.

On peut associer un graphe biparti $G_N$ à chaque combinaison d'une contrainte $\texttt{SND}_{\mathcal{S},\mathcal{N}}(v_p, v_t)$ et d'une fonction de voisinage $N \in \mathcal{N}$. Le graphe biparti $G_N = (V, E)$ est défini ainsi :

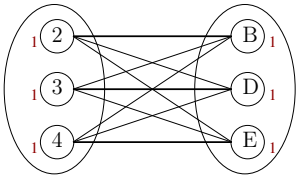
- $V = \mathrm{N}(v_p)|\mathrm{N}(v_t)$ ;
- $E = \{(v'_p, v'_t) \in \mathrm{N}(v_p) \times \mathrm{N}(v_t) \mid v'_t \in dom(x_{v'_p}) \wedge S(v_p, v'_p) \leq S(v_t, v'_t), \forall S \in \mathcal{S}\}$

Si il n'existe pas de couplage du graphe biparti qui couvre $\mathrm{N}(v_p)$, alors la partie droite de l'équation 2 (après le symbole $\Rightarrow$) est fausse et ainsi $(x_{v_p}, v_t)$ n'a pas de support dans la contrainte. Cette valeur peut donc être supprimée de la variable $x_{v_p}$.

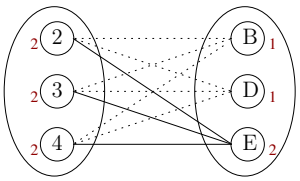



(a) Graphe motif $G_p$ (b) Graphe cible $G_t$

FIGURE 1 – Une instance du problème d'isomorphisme de sous-graphe.




(a) $\texttt{SND}_{\{M^1\},\{\Gamma\}}(1, A)$




(b) $\texttt{SND}_{\{M^2\},\{\Gamma\}}(1, A)$

FIGURE 2 – L'exemple (a) montre l'existence d'un couplage dans le graphe bi-parti $\texttt{SND}_{\{M^1\},\{\Gamma\}}(1, A)$ et l'exemple (b) montre qu'il n'y a pas de couplage faisable pour $\texttt{SND}_{\{M^2\},\{\Gamma\}}(1, A)$.

**Exemple 1** *La figure 1 montre une instance du problème d'isomorphisme de sous-graphe. Les matrices d'adjacence $M_{G_p}$ et $M_{G_t}$ ainsi que $M^2_{G_p}$ et $M^2_{G_t}$, sont :*

$$M_{G_p} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \quad M_{G_t} = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

$$M^2_{G_p} = \begin{pmatrix} 3 & 2 & 2 & 2 \\ 2 & 3 & 2 & 2 \\ 2 & 2 & 3 & 2 \\ 2 & 2 & 2 & 3 \end{pmatrix} \quad M^2_{G_t} = \begin{pmatrix} 3 & 1 & 3 & 1 & 2 \\ 1 & 3 & 1 & 3 & 2 \\ 3 & 1 & 3 & 1 & 2 \\ 1 & 3 & 1 & 3 & 2 \\ 2 & 2 & 2 & 2 & 4 \end{pmatrix}$$

*Considérons les contraintes $SND_{\{M^1\},\{\Gamma\}}(1, A)$ et $SND_{\{M^2\},\{\Gamma\}}(1, A)$ (par simplicité, nous considérons une unique fonction de longueur et une unique fonction de voisinage). Le graphe biparti associé à ces*

*contraintes (et à $\Gamma$) est schématisé dans la figure 2, où $\Gamma(1) = \{2, 3, 4\}$ et $\Gamma(A) = \{B, D, E\}$ (en supposant que les domaines courant de $x_2$, $x_3$ et $x_4$ contiennent les valeurs initiales). Notons que les fonctions de score sont affichées (en rouge) à coté des sommets. Il est clair qu'avec $M^1$, nous ne pouvons rien déduire sur $(1, A)$ puisqu'il existe un couplage couvrant $\Gamma(1)$ (montré en gras dans la figure 2(a)). Néanmoins, avec $M^2$, on peut déduire que $x_1 \neq A$. En effet, lorsqu'on considère le score basé sur $M^2$, toutes les arêtes en pointillés peuvent être supprimées. Par exemple, le score de $x_2$ est $M_p^2[x_1][x_2] = 2$ alors que le score de $B$ est seulement $M_t^2[A][B] = 1$. Cela veut dire que 2 ne peut pas être relié à $B$ et que cette arête peut être supprimée. Finalement, en considérant les arêtes restantes (celles qui ne sont pas en pointillé), il est clair qu'il n'y a pas de couplage couvrant de $\Gamma(1)$.*

### 3.3 Simplification du graphe cible

Lorsque les domaines sont réduits durant le processus de propagation, il est quelquefois possible de raffiner les fonctions de score que nous utilisons. En effet, une arête peut être supprimée du graphe cible quand on a la garantie qu'aucun couple de sommets du graphe motif ne peut être couplé à cette arête. Plus précisément, on dit qu'une arête $(v_t, v'_t) \in E_t$ est *inaccessible* ssi pour chaque arête $(v_p, v'_p) \in E_p$, $(v_t, v'_t) \notin dom(x_{v_p}) \times dom(x_{v'_p})$ et $(v_t, v'_t) \notin dom(x_{v'_p}) \times dom(x_{v_p})$. Ainsi, après le processus de filtrage, on peut simplifier le graphe cible en supprimant les arêtes inaccessibles. Les fonctions de score peuvent alors être mises à jour, comme par exemple les fonctions de longueur dans $\mathcal{M}$ puisque les matrices d'adjacence sont modifiées dès lors qu'une arête est supprimée. En conséquence, on peut recommencer le raisonnement de dominance pour les contraintes SND.

## 4 Étude qualitative

A partir de maintenant, $\text{SND}_{\mathcal{S}}$ sera utilisé comme raccourci de $\text{SND}_{\mathcal{S},\{\texttt{Id},\Gamma,\texttt{all}\}}$, et correspondra au filtrage établi par GAC sur le réseau initial augmenté de toutes les contraintes $\text{SND}_{\mathcal{S},\{\texttt{Id},\Gamma,\texttt{all}\}}(v_p, v_t)$. Dans notre étude, $\mathcal{S}$ pourra être $\mathcal{M}$ (l'ensemble des fonctions de longueur) ou une seule fonction $M^k$ de $\mathcal{M}$.

Nos premiers résultats montrent qu'il peut être utile de raisonner à la fois avec $M^k$ et $M^{k+1}$.

**Proposition 2** *Pour certain $k \geq 1$, $SND_{\{M^k\}}$ n'est pas plus fort que $SND_{\{M^{k+1}\}}$.*

*Proof.* L'exemple 1 prouve que $\text{SND}_{\{M^1\}}$ ne peut pas être plus fort que $\text{SND}_{\{M^2\}}$, puisque $\text{SND}_{\{M^2\}}$ est capable d'inférer $x_1 \neq A$, contrairement à $\text{SND}_{\{M^1\}}$. □

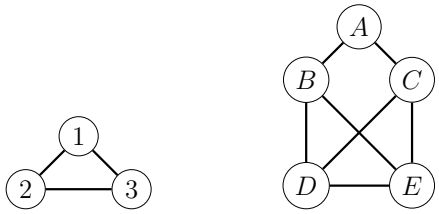
**Proposition 3** *Pour certain $k \geq 1$, $SND_{\{M^{k+1}\}}$ n'est pas plus fort que $SND_{\{M^k\}}$.*

*Proof.* Considérons le problème défini dans la figure 3. Les matrices $M^5_{G_p}$, $M^5_{G_t}$, $M^6_{G_p}$ and $M^6_{G_t}$ sont les suivantes :

$$M^5_{G_p} = \begin{pmatrix} 10 & 11 & 11 \\ 11 & 10 & 11 \\ 11 & 11 & 10 \end{pmatrix} \quad M^5_{G_t} = \begin{pmatrix} 8 & 40 & 40 & 22 & 22 \\ 40 & 26 & 26 & 51 & 51 \\ 40 & 26 & 26 & 51 & 51 \\ 22 & 51 & 51 & 40 & 41 \\ 22 & 51 & 51 & 41 & 40 \end{pmatrix}$$

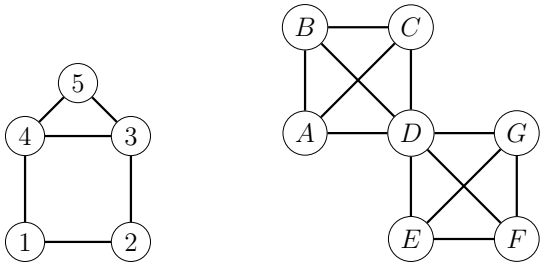
$$M^6_{G_p} = \begin{pmatrix} 22 & 21 & 21 \\ 21 & 22 & 21 \\ 21 & 21 & 22 \end{pmatrix} \quad M^6_{G_t} = \begin{pmatrix} 80 & 52 & 52 & 102 & 102 \\ 52 & 142 & 142 & 103 & 103 \\ 52 & 142 & 142 & 103 & 103 \\ 102 & 103 & 103 & 143 & 142 \\ 102 & 103 & 103 & 142 & 143 \end{pmatrix}$$

Sur cet exemple, d'un côté, $\text{SND}_{\{M^6\}}$ ne permet d'effectuer aucune inférence (observez que toutes les valeurs dans $M^6_{G_t}$ sont systématiquement plus grandes que celles de $M^6_{G_p}$, ce qui garantit une couverture couvrante pour chaque fonction de voisinage). De l'autre coté, comme $M^5_{G_p}[1][1] = 10 > M^5_{G_t}[A][A] = 8$, avec `Id`, on peut inférer que $x_1 \neq A$. □




(a) $G_p$ (b) $G_t$

FIGURE 3 – Un exemple qui prouve que $\text{SND}_{\{M^6\}}$ n'est pas plus fort que $\text{SND}_{\{M^5\}}$.




(a) $G_p$ (b) $G_t$

FIGURE 4 – Un exemple qui montre que $\text{SND}_{\mathcal{M}}$ n'est pas plus fort que SAC

**Proposition 4** *$SND_{\mathcal{M}}$ est incomparable à SAC.*

*Preuve.* Considérons tout d'abord l'exemple 1. Il montre que $\text{SND}_{\mathcal{M}}$ peut filtrer plus de valeurs que SAC. En effet, $\text{SND}_{\mathcal{M}}$ est capable d'inférer $x_1 \neq A$ alors qu'aucun test singleton réalisé par SAC ne permet de détecter une valeur incohérente sur cette instance. Par exemple, établir GAC après l'affectation $x_1 = A$ réduit seulement les domaines de $x_2$, $x_3$ et $x_4$ à $\{B, E, D\}$ , la valeur $A$ est supprimée du fait de la contrainte allDiff et la valeur $C$ est supprimée par les contraintes binaires. Comme aucun domaine ne devient vide, on ne peut pas en déduire que $x_1 \neq A$.

D'un autre coté, la figure 4 montre une instance où SAC filtre plus de valeurs que $\text{SND}_{\mathcal{M}}$. En effet, considérons le test singleton pour $(x_3, A)$. Avec la contrainte allDiff, $A$ est tout d'abord supprimé du domaine des autres variables. Avec les contraintes binaires impliquant $x_3$, les valeurs $E$, $F$ et $G$ sont supprimées des domaines de $x_2$, $x_4$ et $x_5$, ce qui donne à cet instant $dom(x_2) = dom(x_4) = dom(x_5) = \{B, C, D\}$. Nous avons ici 3 valeurs pour 3 variables, la contrainte allDiff va donc supprimer $B$, $C$ et $D$ du domaine de $x_1$, donnant $dom(x_1) = \{E, F, G\}$. Enfin, la contrainte binaire entre $x_1$ et $x_2$ supprime $B$ et $C$ de $dom(x_2)$. A ce stade, une incohérence est détectée et, donc, SAC infère que $x_3 \neq A$. Considérons maintenant $M_{G_p}$, $M_{G_t}$, $M^2_{G_p}$ et $M^2_{G_t}$ :

$$M_{G_p} = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix} \quad M_{G_t} = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

$$M^2_{G_p} = \begin{pmatrix} 2 & 0 & 2 & 0 & 1 \\ 0 & 2 & 0 & 2 & 1 \\ 2 & 0 & 3 & 1 & 1 \\ 0 & 2 & 1 & 3 & 1 \\ 1 & 1 & 1 & 1 & 2 \end{pmatrix} \quad M^2_{G_t} = \begin{pmatrix} 3 & 2 & 2 & 2 & 1 & 1 & 1 \\ 2 & 3 & 2 & 2 & 1 & 1 & 1 \\ 2 & 2 & 3 & 2 & 1 & 1 & 1 \\ 2 & 2 & 2 & 6 & 2 & 2 & 2 \\ 1 & 1 & 1 & 2 & 3 & 2 & 2 \\ 1 & 1 & 1 & 2 & 2 & 3 & 2 \\ 1 & 1 & 1 & 2 & 2 & 2 & 3 \end{pmatrix}$$

On peut vérifier que le raisonnement utilisant $\text{SND}_{\{M^1, M^2\}}$ ne permet pas d'identifier de valeurs incohérentes. Il y a toujours la possibilité de trouver une couverture couvrante sur chaque graphe biparti. Cela est également vrai pour $M^3$ (non montré ici) et à cause de la structure du graphe, la situation donne des scores non exploitables pour tout $k > 4$ : la plus grande valeur de $M^4_{G_p}$ est plus petite que la plus petite valeur de $M^4_{G_t}$ (et cela sera toujours vrai pour $k \geq 5$). En d'autres termes, SND ne peut identifier aucune valeur incohérente. □

**Proposition 5** *$\text{SND}_{\mathcal{M}}$ est strictement plus fort que LAD*

*Preuve.* LAD utilise seulement $\Gamma$ comme fonction de voisinage et aucune fonction de score. De plus, sur l'exemple 1, LAD se comporte comme $\text{SND}_{\{M^1\}}$, qui est strictement plus faible que $\text{SND}_{\mathcal{M}}$. □

Un résumé des relations entre les différents processus de filtrage existant dans la littérature pour le problème d'isomorphisme de sous-graphe et $\text{SND}_{\mathcal{M}}$ est donné dans la figure 5.

## 5 Un algorithme SND incomplet

Cette section décrit un algorithme nommé $\text{SND}^w$ qui met en application notre filtrage SND. En fait, cet algorithme réalise un niveau de filtrage plus faible que $\text{SND}_{\mathcal{M},\{\texttt{Id},\Gamma,\texttt{all}\}}$ complet. De plus amples détails sur ce point sont donnés à la fin de cette section.

**Algorithm 1:** $\text{SND}^w(G_p = (V_p, E_p)$ : Pattern, $G_t = (V_t, E_t)$ : Target)

```
1  repeat
2      M^0_{G_p} ← I_{n_p} ; M^0_{G_t} ← I_{n_t}
3      k ← 1
4      repeat
5          modified ← filterUsingScoring(k)
6          k ← k + 1
7      until k > MIN_ITERS or ¬modified
8      if ¬removeTargetEdges() then
9          finished ← true
10 until finished
```

**Function** filterUsingScoring($k$ : integer) : Boolean

```
1  M^k_{G_p} ← M^{k-1}_{G_p} × M_{G_p}
2  M^k_{G_t} ← M^{k-1}_t × M_{G_t}
3  modified ← false
4  foreach variable x_{v_p} ∈ vars(P) do
5      foreach value v_t ∈ dom(x_{v_p}) do
6          if ¬isCoherent((v_p, v_t), Id)
7            ∨¬isCoherent((v_p, v_t), Γ)
8            ∨¬isCoherent((v_p, v_t), all) then
9              remove v_t from dom(x_{v_p})
10             if dom(x_{v_p}) = ∅ then
11                 throw exception INCONSISTENT
12             modified ← true
13 return modified
```

Les étapes principales de $\text{SND}^w$ sont décrites dans l'algorithme 1 ; les entrées sont le graphe motif $G_p$ et le graphe cible $G_t$. Afin de réduire la quantité de mémoire nécessaire, cet algorithme évite de stocker de trop nombreuses matrices et exploite dans ce

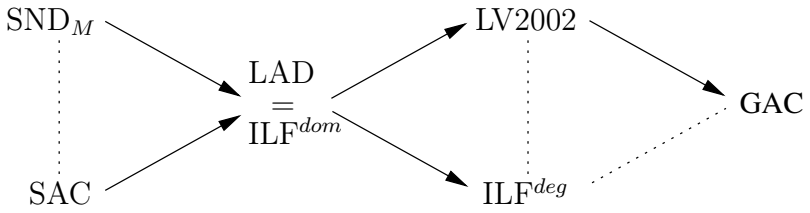



FIGURE 5 – Résumé des relations entre les processus de filtrage. Une flèche de la forme $\phi \to \psi$ signifie que $\phi$ est strictement plus fort que $\psi$. Une ligne en pointillé indique que les deux entités ne sont pas comparables.

**Function** isCoherent(($v_p$,$v_t$) : vertices, $N$ : Neighborhood) : Boolean

```
1 E ← ∅
2 foreach v'_p ∈ N(v_p) do
3     foreach v'_t ∈ N(v_t) ∩ dom(x_{v'_p}) do
4         if M^k_{G_p}[v_p][v'_p] ≤ M^k_{G_t}[v_t][v'_t] then
5             E ← E ∪ {(v'_p, v'_t)}
6 return findCoveringMatching((N(v_p)|N(v_t), E))
```

but les scores $M^k$ de manière indépendante. Le raisonnement commence donc avec $M^1$, puis avec $M^2$ et ainsi de suite, par le biais d'appels à la fonction filterUsingScoring (voir l'appel à la ligne 5, Alg. 1). La constante `MIN_ITERS` est fixée à une valeur faible (par exemple 4) afin d'exploiter au minimum les fonctions de score $M^i$ avec $1 \leq i \leq$ `MIN_ITERS`. La boucle interne **repeat-until** (lignes 4-7) est stoppée quand le filtrage opéré avec $M^k$ ne permet plus d'inférence (la fonction filterUsingScoring retourne alors `false`) et, comme indiqué ci-dessus, lorsqu'on a itéré suffisamment de fois. Après celà, on essaie de supprimer des arêtes inaccessibles du graphe cible, comme indiquée dans la section 3.3, en utilisant une fonction nommée `removeTargetEdges` (non décrite ici). Si cette opération est effective, on recommence le processus entièrement en modifiant la matrice d'adjacence du graphe cible (retour à la ligne 1).

La fonction `filterUsingScoring` commence par calculer $M^k_{G_p}$ et $M^k_{G_t}$ à partir de $M^{k-1}_{G_p}$ et $M^{k-1}_{G_t}$. Nous devons donc stocker uniquement les matrices de deux niveaux différents ce qui nous permet de contrôler l'espace mémoire nécessaire par notre approche. Ensuite, pour chaque couple $(v_p, v_t)$ tel que $v_t \in dom(x_{v_p})$, les conditions des lignes 6–8 correspondent à l'évaluation de la partie droite de l'équation 2, avec $M^k$ comme unique fonction de longueur et $\{\texttt{Id}, \Gamma, \texttt{all}\}$ comme fonctions de voisinage. Quand cette condition est fausse (évaluée au moyen de la fonction isCoherent), la valeur $v_t$ est supprimée de $dom(x_{v_p})$. Si c'était la dernière valeur possible du domaine, une exception est générée, indiquant une incohérence globale. Notons que la fonction filterUsingScoring retourne `true` quand au moins une valeur de domaine est supprimée. Finalement, la fonction isCoherent construit le graphe biparti associé au couple $(v_p, v_t)$ et à la fonction de score $M^k$, et appelle ensuite un algorithme de couplage couvrant (voir l'appel à la fonction `findCoveringMatching` qui n'est pas décrite ici).

Notre algorithme SND$^w$ est clairement plus faible que SND$_{\mathcal{M},\{\texttt{Id},\Gamma,\texttt{all}\}}$ pour plusieurs raisons. Tout d'abord, comme indiqué précédemment, nous exploitons les différents scores de manière indépendante (par simplicité et pour contrôler l'espace mémoire nécessaire). Ensuite, dans nos expérimentations nous avons programmé une version incomplète du couplage couvrant d'un graphe biparti. Enfin, les domaines pouvant être modifiés de façon permanente, nous pouvions envisager de considérer à nouveau les fonctions de score utilisées précédemment (nous pouvions donc ajouter une boucle à l'algorithme 1). Nous pensons toutefois que le sur-coût aurait été trop important.

## 6 Résultats expérimentaux

Afin de montrer l'intérêt pratique de notre approche, nous avons mené diverses expérimentations en utilisant un cluster de machines Xeon 3.0GHz avec 13Go de RAM. Nous avons utilisé les instances d'isomorphisme de sous-graphe provenant de [21] et classées de la manière suivante :

- La série `lv` est composée de 793 instances, provenant de la base Stanford [11, 12].
- Les séries `si2`, `si4`, et `si6` sont composées de 390 instances chacune et proviennent de la base VFlib [4, 20].
- La série `sf` est composée de 100 instances de réseaux *scale free* générées de manière aléatoire en utilisant la distribution d'une loi de puissance [24].

Nous comparons tout d'abord les temps cpu (exprimés en secondes) et le niveau de filtrage (donné par le nombre de valeurs supprimées (del)) de GAC,

| Séries | | | GAC | | LAD | | SAC | | SND$^w$ | |
|---|---|---|---|---|---|---|---|---|---|---|
| Name | # | D | cpu | del | cpu | del | cpu | del | cpu | del |
| lv | 793 | 5,877 | 0.5 | 188 | 0.01 | 103+754 | 24 | 1,237 | 0.6 | **1,502** |
| si2 | 390 | 78K | 0.6 | 3 | 0.29 | 25K99+24K41 | 168 | **52K58** | 1.1 | 51K48 |
| si4 | 390 | 156K | 0.75 | 8 | 1.05 | 54K34+52K98 | 177 | **109K2** | 1.4 | 107K5 |
| si6 | 390 | 235K | 0.8 | 12 | 2.19 | 76K12+84K09 | 193 | 165K3 | 2.1 | **184K4** |
| sf | 100 | 284K | 1 | 0 | 0.32 | 141K3+135K7 | 567 | 234K3 | 2.5 | **244K2** |

TABLE 1 – *Pré-traitement.* Pour chaque technique de filtrage, `cpu` indique le temps moyen (en secondes) et `del` indique le nombre de valeurs supprimées.

| Séries | | MAC | | LAD | | pSAC | | pSND$^w$ | | MSND$^w$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Name | # | cpu | sol | cpu | sol | cpu | sol | cpu | sol | cpu | sol |
| lv | 793 | 24 | 683 | 8.3 | 726 | 89 | 695 | 7.6 | 724 | 9.6 | **729** |
| si2 | 390 | 38 | 352 | 13 | 345 | 153 | 236 | 25 | 356 | 28 | **357** |
| si4 | 390 | 42 | 357 | 19 | 358 | 142 | 212 | 24 | 359 | 22 | **366** |
| si6 | 390 | 46 | 346 | 20 | 372 | 191 | 204 | 22 | 367 | 24 | **375** |
| sf | 100 | 120 | 58 | 2.6 | **100** | 557 | 62 | 4.3 | 99 | 18 | 99 |

TABLE 2 – *Chercher une solution.* Pour chaque technique de filtrage, `cpu` indique le temps moyen (en secondes) pour résoudre une instance et `sol` représente le nombre d'instances résolues (timeout à 1200 secondes).

SAC [1], LAD et SND$^w$. Nous avons commencé à appliquer ces algorithmes isolément, en les considérant donc comme faisant partie d'une étape de pré-traitement. Pour GAC, SAC et SND$^w$, nous avons utilisé notre plateforme AbsCon (écrit en java), alors que pour LAD nous avons utilisé le programme C++ disponible sur la page de son auteur. Le tableau 1 montre les résultats en moyenne obtenus sur les cinq séries d'instances. Pour chaque série, # représente le nombre d'instance et $D$ le nombre moyen de valeurs ($D$ est la moyenne de $D_P = \sum_{x \in vars(P)} |dom^{init}(x)|$ pour tous les réseaux de contraintes $P$ de la série considérée). Notons que la valeur `del` est donnée sous la forme $v_1 + v_2$ pour LAD car ce dernier exploite la structure des graphes pour initialiser les domaines ($v_1$ représente le nombre de valeurs supprimées à cette étape). Nous pouvons tout d'abord observer que SAC et SND$^w$ sont très proches en terme de valeurs supprimées : sur certaines séries SAC est meilleur et sur d'autres, c'est SND$^w$ qui l'est. Par contre au niveau du temps cpu, SAC apparaît clairement plus lent que SND$^w$ (de plusieurs ordres de grandeur). Afin d'avoir une comparaison juste entre SAC et SND$^w$, nous n'avons gardé que les instances pour lesquelles SAC finit avant les 1200 secondes de timeout. Cela peut expliquer pourquoi LAD semble filtrer plus sur la série `sf`. En résumé, SND$^w$ est significativement plus puissant que LAD (et proche de SAC), le sur-coût en temps est relativement faible (en prenant en compte aussi que les langages de programmation utilisés sont différents).

Pour notre deuxième expérimentation, nous avons cherché une solution par instance avec un timeout de 1200 secondes. Nous avons utilisé l'algorithme MAC qui maintient GAC durant le processus de recherche, LAD et aussi MSND$^w$, l'algorithme qui maintient SND$^w$ durant la recherche (nous ne faisons qu'une boucle principale de l'algorithme 1, en supprimant l'appel à removeTargetEdges durant la recherche). Nous avons aussi utilisé MAC après pré-traitement de type SAC (pSAC) ou de type SND$^w$ (pSND$^w$). Le tableau 2 montre les résultats obtenus. En terme du nombre d'instances résolues (`sol`), MSND$^w$ est clairement la meilleure approche, sauf pour la série `sf` ou LAD est extrêmement efficace.

## 7 Conclusion

Nous avons proposé une approche générale pour résoudre le problème d'isomorphisme de sous-graphe. Cette approche, dite SND, est paramétrée par deux ensembles de fonctions de score et de voisinage. Nous avons montré l'intérêt théorique et pratique de SND sur une petite sélection de fonctions. Les perspectives

1. Nous avons testé SAC1 et SAC3 [1]. Les résultats sont assez proches ici (en moyenne). Seuls les résultats de SAC1 sont donc montrés.

immédiates de ce travail consistent d'une part à améliorer l'implantation courante et de l'autre, à mettre au point de nouvelles fonctions de score et de voisinage.

## Remerciements

Les deux premiers auteurs bénéficient d'un soutien CNRS et OSEO dans le cadre du projet ISI 'Pajero'.

## Références

[1] C. Bessiere, S. Cardon, R. Debruyne, and C. Lecoutre. Efficient algorithms for singleton arc consistency. *Constraints*, 16(1) :25–53, 2011.

[2] H. Bunke. Recent developments in graph matching. In *Proceeding of ICPR'00*, volume 2, pages 117–124, 2000.

[3] D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(3) :265–298, 2004.

[4] L.P. Cordella, P. Foggia, C. Sansone, and M. Vento. Performance evaluation of the VF graph matching algorithm. In *Proceedings of ICIAP'99*, pages 1172–1177, 1999.

[5] L.P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 26(10) :1367–1372, 2004.

[6] D.G. Corneil and C.C. Gotlieb. An efficient algorithm for graph isomorphism. *Journal of the ACM*, 17(1) :51–64, 1970.

[7] G. Damiand, C. Solnon, C. de la Higuera, J.-C. Janodet, and E. Samuel. Polynomial algorithms for subisomorphism of nD open combinatorial maps. *Computer Vision and Image Understanding*, 115(7) :996 – 1010, 2011.

[8] C. de la Higuera, J.-C. Janodet, E. Samuel, G. Damiand, and C. Solnon. Polynomial algorithms for open plane graph and subgraph isomorphisms. *Theoretical Computer Science*, 498 :76–99, 2013.

[9] R. Debruyne and C. Bessiere. Some practical filtering techniques for the constraint satisfaction problem. In *Proceedings of IJCAI'97*, pages 412–417, 1997.

[10] R. Debruyne and C. Bessiere. Domain filtering consistencies. *Journal of Artificial Intelligence Research*, 14 :205–230, 2001.

[11] D.E. Knuth. *The Stanford GraphBase : A Platform for Combinatorial Computing*. ACM Press, 1993.

[12] J. Larrosa and G. Valiente. Constraint satisfaction algorithms for graph pattern matching. *Mathematical Structures in Computer Science*, 12(4) :403–422, 2002.

[13] C. Lecoutre. *Constraint networks : techniques and algorithms*. ISTE/Wiley, 2009.

[14] E.M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of computer and system sciences*, 25(1), 1982.

[15] J.J. McGregor. Relational consistency algorithms and their application in finding subgraph and graph isomorphisms. *Information Sciences*, 19 :229–250, 1979.

[16] S. Ndojh Ndiaye and C. Solnon. CP models for maximum common subgraph problems. In *Proceedings of CP'11*, pages 637–644, 2011.

[17] D. C. Porumbel. Isomorphism testing via polynomial-time graph extensions. *Journal of Mathematical Modelling and Algorithms*, 10(2) :119–143, 2011.

[18] J.C. Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of AAAI'94*, pages 362–367, 1994.

[19] J.C. Régin. *Développement d'outils algorithmiques pour l'intelligance artificielle. Application à la chimie organique*. PhD thesis, Université Montpellier II, 1995.

[20] M. De Santo, P. Foggia, C. Sansone, and M. Vento. A large database of graphs and its use for benchmarking graph isomorphism algorithms. *Pattern Recognition Letters*, 24(8) :1067–1079, 2003.

[21] C. Solnon. Alldifferent-based filtering for subgraph isomorphism. *Artificial Intelligence*, 174(12-13) :850–864, 2010.

[22] J.R. Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM*, 23(1) :31–42, 1976.

[23] B. Weisfeiler and A. Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, ser. 2(9), 1968.

[24] S. Zampelli, Y. Deville, and C. Solnon. Solving subgraph isomorphism problems with constraint programming. *Constraints*, 15(3) :327–353, 2010.