Appariement d'ensembles avec édition
Application à la distance d'édition bipartie entre graphes

Sébastien Bougleux[1] Luc Brun[2] Benoit Gaüzère[3]

[1] Normandie Univ, UNICAEN, CNRS, GREYC UMR 6072
[2] Normandie Univ, ENSICAEN, CNRS, GREYC UMR 6072
[3] Normandie Univ, INSA Rouen, LITIS

Résumé

La distance d'édition bipartie entre graphes (bGED) est une mesure de dissimilarité, développée pour approximer la distance d'édition entre graphes (GED) dont le calcul est un problème NP-complet. Calculer la bGED entre deux graphes nécessite de résoudre un problème d'appariement des nœuds des graphes avec édition. Cela revient à transformer un ensemble en un autre en substituant, supprimant et insérant des éléments. Ce problème est formulé comme un problème linéaire d'affectation (LSAP), qui peut être résolu en temps polynomial avec l'algorithme Hongrois. Néanmoins, cela nécessite d'augmenter artificiellement les ensembles pour qu'ils aient le même cardinal, impliquant de la mémoire et du temps de calcul inutiles. Pour éviter cet inconvénient, nous définissons formellement le problème d'appariement optimal avec édition de deux ensembles, et nous proposons une adaptation de l'algorithme Hongrois pour le résoudre. Notons que le problème traité n'est pas restreint au calcul de la bGED.

Mots Clef

problème d'affectation, appariement d'ensembles, distance d'édition

Abstract

The bipartite graph edit distance (bGED) is a dissimilarity measure designed to approximate the graph edit distance between two graphs, whose calculus is NP-complete. Computing the bGED between two graphs requires to solve an assignment problem with edition between the set of nodes of both graphs. Such an assignment is equivalent to a transformation of the first set of nodes into the second one using node substitution, insertion and removal operations. This problem may be formulated as a linear sum assignment problem (LSAP) which may be solved in polynomial time using the Hungarian algorithm. Nevertheless, this formulation requires to increase artificially the size of both sets to be assigned, hence inducing useless memory requirements and execution times. In order to avoid such drawbacks we define formally the assignment with edition between two sets and we propose a new algorithm adapted from the Hungarian method to solve it. Let us finally note that the proposed method may be applied to other problems than the graph edit distance.

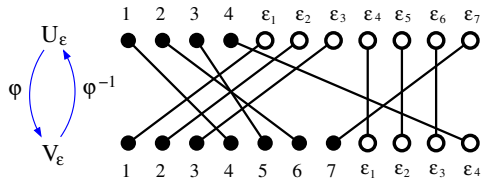
Keywords

linear sum assignment problem, set matching, edit distance

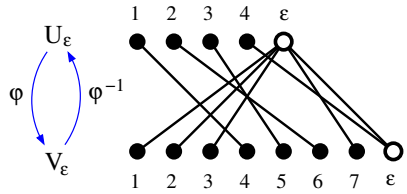
1 Introduction

Calculer efficacement une mesure robuste de similarité ou de dissimilarité entre graphes est un problème majeur en reconnaissance structurelle de formes. La distance d'édition entre graphes (GED), développée dans le contexte de l'appariement de graphes, fournit une telle mesure. Elle est définie par le coût d'une séquence optimale d'opérations d'édition (chemin d'édition) transformant un graphe en un autre, voir par exemple [2]. Bien qu'il soit possible de calculer exactement la GED, par exemple avec l'algorithme A*, c'est un problème NP-complet qui rend difficile le traitement de graphes constitués de plus d'une dizaine de nœuds, même lorsqu'ils sont faiblement connectés. Plusieurs approches ont donc été développées pour approximer efficacement la GED. Parmi elles, la distance bipartie d'édition entre graphes simples (bGED) [9, 7] a reçu plus d'attention [13, 10, 5, 8, 4, 11], en raison du bon rapport entre la qualité de l'approximation qu'elle fournit et son temps de calcul. La bGED entre deux graphes se calcule en deux étapes principales. La première étape consiste à résoudre un problème d'appariement des nœuds des graphes. La seconde étape consiste à construire, à partir de l'appariement, un chemin d'édition. Nous ne détaillerons pas cette dernière étape.

L'appariement calculé par la bGED est tel que chaque élément du premier ensemble de nœuds $\mathcal{U}$ peut être substitué par un unique élément du second ensemble de nœuds $\mathcal{V}$, ou bien supprimé, et tel que chaque nœud non-affecté de $\mathcal{V}$ soit inséré dans $\mathcal{U}$. La suppression d'un élément est équivalente à son insertion dans l'autre ensemble. Pour représenter cet appariement guidé par des opérations d'édition (substitution, suppression/insertion), l'ensemble de nœuds $\mathcal{U}$ (resp. $\mathcal{V}$) est traditionnellement augmenté par un ensemble $\mathcal{E}^{\mathcal{U}}$ (resp. $\mathcal{E}^{\mathcal{V}}$) d'éléments nuls de telle sorte que chaque



(a) proposé dans [9, 7]



(b) proposé dans cet article

FIGURE 1 – Deux représentations d'une même transformation de $\mathcal{U}$ en $\mathcal{V}$ avec opérations d'édition.

nœud puisse être supprimé, c.-à-d. attribué à un unique élément de $\mathcal{E}^{\mathcal{U}}$ (resp. $\mathcal{E}^{\mathcal{V}}$) [9, 7]. Comme les deux ensembles $\mathcal{U}\cup\mathcal{E}^{\mathcal{U}}$ et $\mathcal{V}\cup\mathcal{E}^{\mathcal{V}}$ ont par construction la même cardinalité $n+m$, avec $n=|\mathcal{U}|=|\mathcal{E}^{\mathcal{V}}|$ et $m=|\mathcal{V}|=|\mathcal{E}^{\mathcal{U}}|$, l'appariement prend la forme d'une bijection $\varphi:\mathcal{U}\cup\mathcal{E}^{\mathcal{U}}\to\mathcal{V}\cup\mathcal{E}^{\mathcal{V}}$ (Fig. 1(a)) telle que :

- $\varphi(u)=v$ si u est substitué par v,
- $\varphi(u)=\epsilon_u$ si u est supprimé,
- $\varphi(\epsilon_v)=v$ si v est inséré dans $\mathcal{U}$,
- $\varphi(\epsilon_v)=\epsilon_u$ sinon (permet de garantir la bijectivité).

Étant donné un coût c associé à toute opération d'édition, le problème d'attribution (désigné par sLSAPE) consiste à calculer une telle bijection pour laquelle la somme des coûts est minimale :

$$\underset{\varphi\in\mathcal{B}}{\operatorname{argmin}}\left\{A(\varphi,c)\overset{\text{def.}}{=}\sum_{u\in\mathcal{U}\cup\mathcal{E}_{\mathcal{U}}}c(u,\varphi(u))\right\} \qquad (1)$$

où $\mathcal{B}$ désigne l'ensemble des bijections de $\mathcal{U}\cup\mathcal{E}^{\mathcal{U}}$ sur $\mathcal{V}\cup\mathcal{E}^{\mathcal{V}}$. Eq. 1 est un problème classique d'appariement linéaire (LSAP), qui peut être résolu en temps polynomial, par exemple avec l'algorithme Hongrois en $O((n+m)^3)$ en temps et en $O((n+m)^2)$ en mémoire, voir [3, 7] pour plus de détails.

Néanmoins, notons que les bijections définies ci-dessus sont contraintes. En effet, chaque élément de $\mathcal{E}^{\mathcal{U}}$ (resp. $\mathcal{E}^{\mathcal{V}}$) ne peut être attribué qu'à un seul élément de $\mathcal{V}$ (resp. $\mathcal{U}$), ce qui est pris en compte à travers la fonction de coût en interdisant l'attribution aux autres éléments par un coût infini. De plus, comme les attributions entre éléments nuls servent à garantir la bijectivité, elles ne doivent pas influencer la valeur de A, leur coût est donc nul. Ces deux dernières contraintes sont dues à l'augmentation des données pour ramener le problème d'appariement d'ensembles avec édition à un problème classique d'appariement d'ensembles (LSAP). Cela a pour effet de traiter explicitement ces contraintes lors du calcul de la solution, consommant du temps et de la mémoire inutilement. Pour éviter cet inconvénient, une variante du sLSAPE a été proposée dans [11], mais elle ne permet l'insertion/suppression que dans un seul des deux ensembles.

Dans cet article nous proposons de formaliser le problème d'appariement d'ensembles avec édition en ne considérant qu'un seul élément nul par ensemble (Section 2), n'augmentant pas les données du problème (Fig 1(b)) et autorisant ainsi l'insertion/suppression dans les deux ensembles. L'analyse du problème avec édition, équivalent au sLSAPE, montre qu'il est possible d'adapter des algorithmes résolvant initialement le LSAP. Nous proposons à la Section 3 une adaptation de l'algorithme Hongrois présenté dans [6, 3]. Cette adaptation calcule une solution en $O(\min\{n,m\}^2\max\{n,m\})$ en temps et en $O(nm)$ en mémoire, ce qui améliore les complexités obtenues pour le sLSAPE. Ceci est confirmé à travers différentes expérimentations présentées à la Section 4. Les démonstrations des propriétés sont présentées en détail dans [1].

2 LSAPE

2.1 Affectations avec édition

Soient $\mathcal{U}$ et $\mathcal{V}$ deux ensembles, avec $n=|\mathcal{U}|$ et $m=|\mathcal{V}|$. Une affectation avec édition (ϵ-affectation), des éléments de $\mathcal{U}$ aux éléments de $\mathcal{V}$, associe (Fig.1(b)) :

- à chaque élément de $\mathcal{U}$ un unique élément de $\mathcal{V}$ (substitution) ou l'élément nul noté ϵ (suppression), et
- à chaque élément non-attribué de $\mathcal{V}$ l'élément nul ϵ (insertion dans $\mathcal{U}$).

Soient $\mathcal{U}_\epsilon=\mathcal{U}\cup\{\epsilon\}$, $\mathcal{V}_\epsilon=\mathcal{V}\cup\{\epsilon\}$, et $\mathcal{P}(\cdot)$ l'ensemble des parties. Une ϵ-affectation peut être représentée par une fonction $\psi:\mathcal{U}_\epsilon\to\mathcal{P}(\mathcal{V}_\epsilon)$ satisfaisant les contraintes suivantes :

$$\begin{cases}\forall u\in\mathcal{U}, & |\psi(u)|=1\\ \forall v\in\mathcal{V}, & |\psi^{-1}[v]|=1\\ \epsilon\in\psi(\epsilon)\end{cases} \qquad (2)$$

où $\psi^{-1}[v]=\psi^{-1}[\{v\}]$ désigne la pré-image d'un singleton par ψ. L'ensemble des ϵ-affectations de $\mathcal{U}$ vers $\mathcal{V}$ est noté $\Psi_\epsilon(\mathcal{U},\mathcal{V})$. Une ϵ-affectation peut être vue comme une bijection où la contrainte de bijectivité a été relâchée pour l'élément ϵ des deux ensembles (Eq. 2). Soient $\mathcal{U}^\sharp=\{u\in\mathcal{U}\mid\psi(u)\in\mathcal{V}\}$ et $\mathcal{V}^\sharp=\{v\in\mathcal{V}\mid\psi^{-1}[v]\subset\mathcal{U}\}$ les ensembles associés aux substitutions. Notons que la restriction de ψ à $\psi^\sharp:\mathcal{U}^\sharp\to\mathcal{V}$ est injective. Soit $\mathcal{I}(\mathcal{U},\mathcal{V})$ l'ensemble des injections d'un sous-ensemble de $\mathcal{U}$ vers $\mathcal{V}$. Nous avons alors le résultat suivant.

Proposition 1 *L'application $f:\Psi_\epsilon(\mathcal{U},\mathcal{V})\to\mathcal{I}(\mathcal{U},\mathcal{V})$ telle que $f(\psi)=\psi^\sharp$ est bijective. Il existe alors*

$$|\Psi_\epsilon(\mathcal{U},\mathcal{V})|=|\mathcal{I}(\mathcal{U},\mathcal{V})|=\sum_{p=0}^{\min\{n,m\}}C_n^pC_m^pp! \qquad (3)$$

ϵ-affectations différentes de $\mathcal{U}$ vers $\mathcal{V}$, avec $n=|\mathcal{U}|$, $m=|\mathcal{V}|$, et C_n^p désigne le nombre de combinaisons.

2.2 Affectations de coût minimal

Étant donné une fonction de coût, le problème d'affectation consiste alors à trouver une ϵ-affectation de coût total minimal (LSAPE), c.-à-d. satisfaisant :

$$\underset{\psi\in\Psi_\epsilon(\mathcal{U},\mathcal{V})}{\operatorname{argmin}} \left\{ A_\epsilon(\psi,c) \overset{\text{def.}}{=} \sum_{u\in\mathcal{U}_\epsilon} \sum_{v\in\psi(u)} c(u,v) \right\} \quad (4)$$

Le coût total A_ϵ peut se décomposer en trois termes, chacun faisant intervenir un type d'opération d'édition :

$$A_\epsilon(\psi,c) = \underbrace{\sum_{\substack{u\in\mathcal{U}\\ \psi(u)=\{v\}}} c(u,v)}_{\text{substitutions}} + \underbrace{\sum_{\substack{u\in\mathcal{U}\\ \psi(u)=\{\epsilon\}}} c(u,\epsilon)}_{\text{suppressions}} + \underbrace{\sum_{\substack{v\in\mathcal{V}\\ \psi^{-1}[v]=\{\epsilon\}}} c(\epsilon,v)}_{\text{insertions}}$$

Il est alors possible de montrer que le LSAPE (Eq. 4) est équivalent au sLSAPE (Eq. 1) [1]. Afin de résoudre le LSAPE, nous l'exprimons matriciellement.

Toute ϵ-affectation ψ peut être représentée par une matrice binaire $\mathbf{X}\in\{0,1\}^{(n+1)\times(m+1)}$ telle que ses n premières lignes correspondent aux éléments de $\mathcal{U}$, ses m premières colonnes aux éléments de $\mathcal{V}$, sa ligne $n+1$ et sa colonne $m+1$ à l'élément ϵ, et

$$\begin{cases} x_{i,j} = \delta_{\psi(u_i)=\{v_j\}}, & \forall (i,j)\in\{1,\dots,n\}\times\{1,\dots,m\} \\ x_{n+1,j} = \delta_{v_j\in\psi(\epsilon)}, & \forall j = 1,\dots,m \\ x_{i,m+1} = \delta_{\psi(u_i)=\epsilon}, & \forall i = 1,\dots,n \\ x_{n+1,m+1} = 1 \end{cases}$$

avec $\delta_r = 1$ si la relation r est vraie ou 0 sinon. Du fait des contraintes sur ψ (Eq. 2), une telle matrice satisfait les contraintes suivantes :

$$\begin{cases} \sum_{j=1}^{m+1} x_{i,j} = 1, & \forall i = 1,\dots,n \\ \sum_{i=1}^{n+1} x_{i,j} = 1, & \forall j = 1,\dots,m \\ x_{n+1,m+1} = 1 \end{cases} \quad (5)$$

Notons $\mathcal{S}_{n,m,\epsilon}$ l'ensemble des matrices binaires satisfaisant ces contraintes. Contrairement à une matrice de permutation, où la somme de chaque ligne et de chaque colonne est égale à 1, la dernière ligne et la dernière colonne d'une matrice satisfaisant Eq. 5 satisfont :

$$1 \le \sum_{j=1}^{m+1} x_{n+1,j} \le m+1, \quad \text{et} \quad 1 \le \sum_{i=1}^{n+1} x_{i,m+1} \le n+1$$

Soit $\mathbf{C}\in[0,+\infty[^{(n+1)\times(m+1)}$ la matrice exprimant les coûts de toutes les opérations possibles d'édition :

$$c_{i,j} = \begin{cases} c(u_i,v_j) & \text{si } (i,j)\in\{1,\dots,n\}\times\{1,\dots,m\} \\ c(u_i,\epsilon) & \text{si } i\in\{1,\dots,n\},\ j=m+1 \\ c(\epsilon,v_j) & \text{si } j\in\{1,\dots,m\},\ i=n+1 \\ c(\epsilon,\epsilon)=0 & \text{si } i=n+1,\ j=m+1 \end{cases}$$

c.-à-d. ayant la même structure que les matrices de $\mathcal{S}_{n,m,\epsilon}$. Le LSAPE (Eq. 4) revient alors à chercher une matrice

$$\widetilde{\mathbf{X}} \in \underset{\mathbf{X}\in\mathcal{S}_{n,m,\epsilon}}{\operatorname{argmin}} \left\{ A_\epsilon(\mathbf{X},\mathbf{C}) \overset{\text{def.}}{=} \sum_{i=1}^{n+1}\sum_{j=1}^{m+1} c_{i,j}x_{i,j} \right\} \quad (6)$$

Cette expression peut se réécrire sous la forme d'un problème de programmation binaire linéaire :

$$\min_{\mathbf{x}} \left\{ \mathbf{c}^T\mathbf{x} \mid \mathbf{L}\mathbf{x} = \mathbf{1},\ \mathbf{x}\in\{0,1\}^{(n+1)(m+1)} \right\} \quad (7)$$

avec $\mathbf{x} = \text{vec}(\mathbf{X}) \in \{0,1\}^{(n+1)(m+1)}$ le vecteur obtenu en concaténant les lignes de $\mathbf{X}$, et de même $\mathbf{c} = \text{vec}(\mathbf{C}) \in [0,+\infty[^{(n+1)(m+1)}$. Le système linéaire $\mathbf{L}\mathbf{x}=\mathbf{1}$ exprime les contraintes données par Eq. 5, avec la matrice $\mathbf{L}\in\{0,1\}^{(n+m+2)\times(n+1)(m+1)}$ définie $\forall (i,j)\in\mathcal{U}_\epsilon\times\mathcal{V}_\epsilon$ par :

$$\begin{cases} l_{k,(i,j)} &= \delta_{k=i}, & \forall k = 1,\dots,n \\ l_{n+1,(i,j)} &= \delta_{(i=n+1)\wedge(j=m+1)} & \\ l_{n+1+k,(i,j)} &= \delta_{k=j}, & \forall k = 1,\dots,m \\ l_{n+m+2,(i,j)} &= \delta_{(i=n+1)\wedge(j=m+1)} & \end{cases} \quad (8)$$

La matrice $\mathbf{L}$ définit le domaine de recherche des solutions. Elle peut être vue comme la matrice d'adjacence nœuds–arcs d'un graphe mixte $G_{\mathbf{L}}$ composé : du graphe biparti complet généré à partir de $\mathcal{U}$ et de $\mathcal{V}$ comme nœuds, des deux graphes orientés bipartis $(\mathcal{U}\cup\{\epsilon\},\mathcal{U}\times\{\epsilon\})$ et $(\mathcal{V}\cup\{\epsilon\},\mathcal{V}\times\{\epsilon\})$ représentant les insertions et les suppressions, et de l'arête (ϵ,ϵ). Notons qu'il n'existe pas d'arc allant de ϵ à l'un des éléments de $\mathcal{U}$ ou de $\mathcal{V}$.

Le LSAPE et le LSAP minimisent la même fonctionnelle. Bien que ces problèmes diffèrent sur l'expression des contraintes, la méthodologie utilisée pour dériver une solution est la même.

Par définition, la somme de chaque colonne de $\mathbf{L}$ n'est pas plus grande que 2, et l'ensemble de ses lignes peut être partitionné en deux sous-ensembles ($\{1,\dots,n+1\}$ et $\{n+2,\dots,n+m+2\}$) de telle sorte qu'un 1 n'apparaisse au plus qu'une seule fois sur chaque colonne de chacun des sous-ensembles. Ceci implique que la matrice est totalement unimodulaire. Par conséquent le LSAPE a toujours une ou plusieurs solutions binaires optimales [12]. Une solution peut être calculée avec des méthodes génériques de programmation linéaire. Pour un calcul plus efficace en temps et en mémoire, nous proposons à la Sec. 3 un algorithme Hongrois adapté à la résolution du LSAPE et de son dual.

2.3 Problème primal-dual et solutions

Par des outils standards en théorie de la dualité, le problème dual du LSAPE s'écrit comme le programme linéaire

$$\max_{(\mathbf{u},\mathbf{v})} \left\{ \mathbf{1}_{n+1}^T\mathbf{u} + \mathbf{1}_{m+1}^T\mathbf{v} \mid \mathbf{L}^T \begin{pmatrix}\mathbf{u}\\ \mathbf{v}\end{pmatrix} \le \mathbf{c}, \begin{pmatrix}\mathbf{u}\\ \mathbf{v}\end{pmatrix} \in \mathbb{R}^{n+m+2} \right\}$$

où $\mathbf{u}$ et $\mathbf{v}$ associent une capacité (ou étiquette) à chaque élément de $\mathcal{U}_\epsilon$ et de $\mathcal{V}_\epsilon$, respectivement. Remarquons que le système de contraintes ne fait intervenir u_{n+1} et v_{m+1} que dans la contrainte $u_{n+1}+v_{m+1}\le c_{n+1,m+1}=0$. En raison de la maximisation, cette dernière somme ne peut pas être négative, et donc $u_{n+1}+v_{m+1}=0$. Comme u_{n+1} ou v_{n+1} n'interviennent pas dans d'autres contraintes, le

système peut se réécrire

$$\mathbf{u}\mathbf{1}_{m+1}^T + \mathbf{1}_{n+1}\mathbf{v}^T \leq \mathbf{C}, \text{ avec } u_{n+1} = v_{m+1} = 0,$$

sans changer le problème. Le problème dual du LSAPE est similaire au problème dual du LSAP avec la capacité d'un élément (ϵ) dans chaque ensemble contrainte à être nulle.
D'après le théorème de dualité forte, si $\mathbf{X}$ (ou $\mathbf{x}$) est une solution du problème primal, alors son problème dual a une solution optimale $(\mathbf{u}, \mathbf{v})$, et la paire de solutions $(\mathbf{X}, (\mathbf{u}, \mathbf{v}))$ satisfait

$$A_\epsilon(\mathbf{X}, \mathbf{C}) = E(\mathbf{u}, \mathbf{v}) \tag{9}$$

avec $E(\mathbf{u}, \mathbf{v}) = \mathbf{1}_{n+1}^T\mathbf{u} + \mathbf{1}_{m+1}^T\mathbf{v}$. Nous avons alors la propriété suivante.

Proposition 2 *Soit $\mathbf{C} \in [0, +\infty[^{(n+1)\times(m+1)}$ une matrice de coûts d'édition. Soient $\mathbf{u} \in \mathbb{R}^{n+1}$ et $\mathbf{v} \in \mathbb{R}^{m+1}$ deux vecteurs tels que $u_{n+1} = v_{m+1} = 0$ et $\mathbf{u}\mathbf{1}_{m+1}^T + \mathbf{1}_{n+1}\mathbf{v}^T \leq \mathbf{C}$. Soit $\mathbf{X} \in \{0,1\}^{(n+1)\times(m+1)}$ une ϵ-affectation, elle satisfait $A_\epsilon(\mathbf{X}, \mathbf{C}) = A_\epsilon(\mathbf{X}, \overline{\mathbf{C}}) + E(\mathbf{u}, \mathbf{v})$, où la matrice transformée $\overline{\mathbf{C}} = \mathbf{C} - \mathbf{u}\mathbf{1}_{m+1}^T - \mathbf{1}_{n+1}\mathbf{v}^T$ est une matrice de coûts d'édition.*

Par conséquent, toute matrice $\mathbf{X}$ minimisant $A_\epsilon(\mathbf{X}, \mathbf{C})$ minimise aussi $A_\epsilon(\mathbf{X}, \overline{\mathbf{C}})$, puisque $E(\mathbf{u}, \mathbf{v})$ est constant, et les matrices $\mathbf{C}$ et $\overline{\mathbf{C}}$ sont dites équivalentes. À partir de Eq. 9 nous avons alors la propriété suivante.

Proposition 3 *$(\mathbf{X}, (\mathbf{u}, \mathbf{v}))$ résout le LSAPE et son dual ssi $A_\epsilon(\mathbf{X}, \overline{\mathbf{C}}) = 0 \overset{def.}{\Leftrightarrow} A_\epsilon(\mathbf{X}, \mathbf{C}) = E(\mathbf{u}, \mathbf{v})$.*

Résoudre le LSAPE et son dual revient donc à calculer une matrice $\mathbf{X}$ et une matrice $\overline{\mathbf{C}}$ satisfaisant :

$$\begin{aligned} & x_{i,j}\overline{c}_{i,j} = 0, \ \forall (i,j) \in \{1,\ldots,n+1\} \times \{1,\ldots,m+1\} \\ \Leftrightarrow & ((x_{i,j} = 1) \wedge (\overline{c}_{i,j} = 0)) \vee ((x_{i,j} = 0) \wedge (\overline{c}_{i,j} \geq 0)) \\ \Leftrightarrow & ((x_{i,j} = 1) \wedge (c_{i,j} = u_i + v_j)) \\ & \vee ((x_{i,j} = 0) \wedge (c_{i,j} \geq u_i + v_j)) \end{aligned} \tag{10}$$

où $\mathbf{u}$ et $\mathbf{v}$ sont les variables duales avec $u_{n+1} = v_{m+1} = 0$. Cette relation est exploitée à la section suivante pour résoudre le LSAPE.

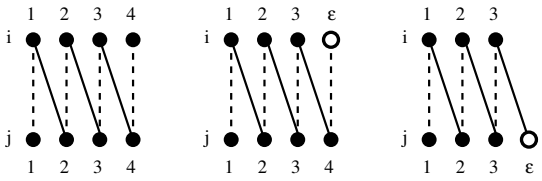
3 Algorithme Hongrois

Afin de résoudre le LSAPE (Eq. 4), et son problème dual, nous adaptons l'algorithme Hongrois présenté dans [6, 3], initialement conçue pour résoudre le problème d'affectation classique défini par Eq. 1 et son dual. Pour simplifier les notations, nous supposons que $\mathcal{U} = \{1,\ldots,n\}$, $\mathcal{V} = \{1,\ldots,m\}, \mathcal{U}_\epsilon = \mathcal{U} \cup \{n+1\}$ et $\mathcal{V}_\epsilon = \mathcal{V} \cup \{m+1\}$.

3.1 ϵ-affectations partielles et codage

Une ϵ-affectation partielle peut être représentée par une matrice $\mathbf{X} \in \{0,1\}^{(n+1)\times(m+1)}$ satisfaisant les contraintes suivantes :

$$\begin{cases} \sum_{j=1}^{m+1} x_{i,j} \leq 1, & \forall i = 1,\ldots,n \\ \sum_{i=1}^{n+1} x_{i,j} \leq 1, & \forall j = 1,\ldots,m \\ x_{n+1,m+1} = 1 \end{cases} \tag{11}$$



(a) que des substitutions (b) se termine par une suppression (c) se termine par une insertion

FIGURE 2 – Trois cas de chemins augmentés commençant avec une arête non-attribuée (i_1, j_1). Les arêtes/arcs non-attribués sont en traits pointillés.

relâchant celles sur les ϵ-affectations. Les algorithmes présentés ci-dessous ne construisent pas explicitement de matrice $\mathbf{X}$ mais une paire de vecteurs (ρ, ϱ) définis par $\rho \in \{0,1,\ldots,m+1\}^n$ et $\varrho \in \{0,1,\ldots,n+1\}^m$ tels que :

$$\begin{aligned} \rho_i &= \begin{cases} j & \text{si } \exists j \in \{1,\ldots,m+1\} \mid x_{i,j} = 1 \\ 0 & \text{sinon (non-affecté)} \end{cases} \\ \varrho_j &= \begin{cases} i & \text{si } \exists i \in \{1,\ldots,n+1\} \mid x_{i,j} = 1 \\ 0 & \text{sinon (non-affecté)} \end{cases} \end{aligned} \tag{12}$$

Étant donné une matrice $\mathbf{C}$ de coûts d'édition, une ϵ-affectation partielle initiale, ainsi qu'une paire de variables duales associées $(\mathbf{u}, \mathbf{v})$ avec $u_{n+1} = v_{m+1} = 0$, peuvent être simplement calculées avec :

$$\begin{cases} \forall i = 1,\ldots,n, & u_i = \min\{c_{i,j},\ j = 1,\ldots,m+1\} \\ \forall j = 1,\ldots,m, & v_j = \min\{c_{i,j} - u_i,\ i = 1,\ldots,n+1\} \end{cases}$$

puis en construisant la paire (ρ, ϱ) pour qu'elle satisfasse la condition définie par Eq. 10.

3.2 Chemins augmentés

Supposons qu'au moins un élément de $k \in \mathcal{V}$ ne soit pas attribué par l'ϵ-affectation partielle $\mathbf{X}$. Le sous-problème du LSAPE (Eq. 4) consiste à calculer une nouvelle ϵ-affectation telle que k soit attribué et que tous les éléments de $\mathcal{U}$ et de $\mathcal{V}$ préalablement attribués le soient toujours. Nous notons $\mathcal{S}_{n,m,\epsilon,k}$ l'ensemble de toutes les ϵ-attributions augmentées à partir de k.
Pour cela considérons le graphe biparti $G_\mathbf{L}$ représentant le domaine de recherche (Sec. 2). Soit G^0 le sous-graphe de $G_\mathbf{L}$ satisfaisant $c_{i,j} = u_i + v_j$ pour chacune de ses arêtes/arcs. Par définition, l'ϵ-affectation représentée par $\mathbf{X}$ correspond à un sous-graphe de G^0 et définit un ensemble de nœuds et d'arêtes attribués (dont les deux nœuds sont attribués).
Nous proposons de construire une nouvelle ϵ-affectation par un algorithme de calcul de plus courts chemins, qui est une adaptation de celui présenté dans [6, 3]. Par définition, un chemin P dans G alterne un nœud d'un ensemble et un nœud de l'autre. Sa longueur, pénalisée par les coûts d'édition, se mesure par

$$\gamma_\mathbf{C}(P) = \sum_{(i,j) \subset P} c_{i,j} \tag{13}$$

Un chemin est alterné si ses arêtes sont alternativement non-attribuées et attribuées. Un chemin alterné qui commence avec un élément non-attribué $k \in \mathcal{V}$ (resp. $\mathcal{U}$) et se termine par (Fig. 2) :

- un élément non-attribué de $\mathcal{U}$ (reps. $\mathcal{V}$), ou
- l'élément nul ϵ de $\mathcal{U}_\epsilon$ ou de $\mathcal{V}_\epsilon$,

est un chemin augmenté si sa longueur définie par Eq. 13 est minimale parmi l'ensemble des chemins alternés connectant k à un élément satisfaisant l'un des deux cas ci-dessus. Notons que comme il n'existe pas d'arc connectant ϵ à un élément de $\mathcal{U}$ ou de $\mathcal{V}$ dans $G_{\mathbf{L}}$, si ϵ appartient au chemin alors c'est obligatoirement un nœud terminal.
Contrairement à la définition classique [3], un chemin augmenté peut ici se terminer par une arête attribuée. Supposons que le premier nœud du chemin P soit $k \in \mathcal{V}$, alors les deux configurations suivantes peuvent être rencontrées :

- Si $i_l \in \mathcal{U}_\epsilon$ est le dernier nœud de P, P correspond à une séquence $(k = j_1, i_1, j_2, i_2, \ldots, j_l, i_l)$ avec $j_s \in \mathcal{V}$ et $i_r \in \mathcal{U}$. Comme la première arête est non-attribuée, et comme le chemin est alterné, la dernière arête est aussi non-attribuée, comme dans le cas classique.
- Si $\epsilon \in \mathcal{V}_\epsilon$ est le dernier nœud de P, P correspond à une séquence $(k = j_1, i_1, j_2, i_2, \ldots, j_l, i_l, \epsilon)$ avec $j_s \in \mathcal{V}$ et $i_r \in \mathcal{U}$. Dans ce cas, la dernière arête est attribuée, c.-à-d. i_l est associé à ϵ.

Proposition 4 *Soient* $\mathbf{X}$ *une matrice représentant une* ϵ*-affectation partielle et* $(\mathbf{u}, \mathbf{v})$ *la paire de variables duales associées telles que Eq. 10 soit satisfaite. Soit* P *un chemin augmenté commençant par un élément non-attribué* $k \in \mathcal{V}$ *(resp.* $\mathcal{U}$*). La matrice* $\mathbf{X}'$*, de terme général* $x'_{i,j} = 1 - x_{i,j}$ *si* $(i,j) \subset P$ *ou* $x'_{i,j} = x_{i,j}$ *sinon, représente une* ϵ*-affectation partielle optimale*

$$\mathbf{X}' \in \operatorname*{argmin}_{\mathbf{Y}} \{A_\epsilon(\mathbf{Y}, \mathbf{C}) \mid \mathbf{Y} \in S_{n,m,\epsilon,k}\}, \qquad (14)$$

avec un coût total $A_\epsilon(\mathbf{X}', \mathbf{C}) = \gamma_{\overline{\mathbf{C}}}(P) + E(\mathbf{u}, \mathbf{v})$ *où* $\overline{\mathbf{C}} = \mathbf{C} - \mathbf{u}\mathbf{1}_{m+1}^T - \mathbf{1}_{n+1}\mathbf{v}^T$.

Remarquons que si toutes les arêtes de P satisfont $\overline{c}_{i,j} = 0$ (P est donc contenu dans G^0), alors $\gamma_{\overline{\mathbf{C}}}(P) = 0$ et par conséquent $A_\epsilon(\mathbf{X}', \mathbf{C}) = E(\mathbf{u}, \mathbf{v})$. Résoudre le sous-problème revient donc à calculer un chemin augmenté P pour lequel il existe une matrice de coûts $\overline{\mathbf{C}}$ telle que $\gamma_{\overline{\mathbf{C}}}(P) = 0$.
L'Algorithme 1 détaille le calcul d'un tel chemin augmenté à partir d'un élément non-attribué $k \in \mathcal{V}$. Il consiste à construire itérativement un arbre de chemins alternés enraciné en k, tout en transformant la matrice de coûts, jusqu'à ce qu'un chemin augmenté soit trouvé. Cet arbre est représenté à la fois par le vecteur $\mathbf{pred} \in \mathcal{V}^n$ (pour les arêtes non-attribuées) et par la paire de vecteurs (ρ, ϱ) codant l'ϵ-affectation (pour les arêtes attribuées). Les arcs de $\mathcal{U}$ ou de $\mathcal{V}$ vers ϵ ne sont pas explicitement codés.
L'arbre est construit en faisant croître un ensemble $SU \subset \mathcal{U}$ et un ensemble $SV \subset \mathcal{V}$, représentant les nœuds de l'arbre,

Algorithme 1 : Chemin augmenté à partir de $k \in \mathcal{V}$.

```
1  début Augment(k, C, ρ, ϱ, u, v)
2      (n, m) ← nombre de lignes−1 et de colonnes−1 de C
3      π ← +∞_n, pred ← 0_n, j ← k
4      SV, SU, LU ← ∅, U ← {1, ..., n}
5      tant que vrai faire
6          SV ← SV ∪ {j}
7          si (ϱ_j ≤ n) ∧ (c_{n+1,j} − v_j = 0) alors
8              retourner (n+1, j, u, v, pred)
9          // recherche de candidats pour l'augmentation
10         pour chaque i ∈ U \ LU faire
11             si c_{i,j} − u_i − v_j < π_i alors
12                 pred_i ← j
13                 π_i ← c_{i,j} − u_i − v_j
14                 si π_i = 0 alors
15                     si ρ_i ∈ {0, m+1} alors
16                         retourner (i, 0, u, v, pred)
17                     LU ← LU ∪ {i}
18         // pas de candidat, m.-à-j. des variables duales
19         // et ajout des candidats
20         si LU \ SU = ∅ alors
21             δ_s ← min{π_i, i ∈ U \ LU}
22             (l, δ_ϵ) ← (arg, min){c_{n+1,j} − v_j, j ∈ SV}
23             δ ← min{δ_s, δ_ϵ}
24             pour chaque j ∈ SV faire v_j ← v_j + δ
25             pour chaque i ∈ LU faire u_i ← u_i − δ
26             si δ_ϵ ≤ δ_s retourner (n+1, l, u, v, pred)
27             pour chaque i ∈ U \ LU faire
28                 π_i ← π_i − δ
29                 si π_i = 0 alors
30                     si ρ_i ∈ {0, m+1} alors
31                         retourner (i, 0, u, v, pred)
32                     LU ← LU ∪ {i}
33         // augmentation
34         i ← un élément de LU \ SU
35         SU ← SU ∪ {i}, j ← ρ_i
```

c.-à-d. les éléments qui ne seront plus parcourus par l'algorithme. À chaque itération, l'algorithme ajoute un nouveau nœud (l. 6) $j \in \mathcal{V} \setminus SV$ à SV (initialement la racine k) et explore les voisins $i \in \mathcal{U} \setminus LU$ de ce nœud (l. 10) pour permettre à l'arbre de s'étendre, où $LU \subset \mathcal{U}$ désigne les nœuds déjà explorés, *c.-à-d.* ceux de SU et les nœuds candidats pour l'extension. L'extension est possible (l. 14) si $\overline{c}_{i,j} = 0$ ((i,j) est une arête de G^0). Dans ce cas i est ajouté à LU (l. 17), devenant ainsi un nœud candidat. Lorsqu'il n'existe aucun nœud candidat pour l'extension dans $LU \setminus SU$ (l. 20), la matrice de coûts est implicitement mise à jour à travers celle des variables duales (l. 24-25) pour qu'au moins un nœud $i \in \mathcal{U} \setminus LU$ satisfasse $\overline{c}_{i,j} = 0$ (voir ci-dessous), garantissant ainsi que $LU \setminus SU \neq \emptyset$. Ceci permet d'étendre l'arbre (l. 34-35) en sélectionnant un nœud candidat $i \in LU \setminus SU$ et en l'ajoutant à SU, ce qui ajoute implicitement l'arête non-attribuée (i,j) de G^0 à l'arbre,

ainsi que l'arête attribuée (i, ρ_i) de G^0. Le nœud ρ_i est ensuite considéré pour l'itération suivante. Le processus se termine (l. 8, 16, 26 et 31) lorsqu'une arête de G^0 possédant un nœud non-attribué de $\mathcal{U}$ est rencontrée ($\rho_i = 0$), ou lorsque l'élément nul $\epsilon \in \mathcal{V}_\epsilon$ est atteint ($\rho_i = m+1$), ou bien lorsque l'élément nul $\epsilon \in \mathcal{U}_\epsilon$ est atteint (l. 7 et 26).

L'étape de mise à jour des variables duales consiste à augmenter G^0 avec au moins une nouvelle arête non-attribuée qui connecte un nœud $j \in SV$ à un nœud $i \in I_\epsilon$ avec $I = \mathcal{U} \setminus LU$ et $I_\epsilon = I \cup \{n+1\}$.

Proposition 5 *Soient* $(\mathbf{X}, (\mathbf{u}, \mathbf{v}))$ *et* $\mathbf{C}$ *tels que Eq. 10 soit satisfaite. Soit* $\delta = \min\{c_{i,j} | (i,j) \in I_\epsilon \times SV\}$. *Par construction,* $I_\epsilon \times SV$ *ne contient pas d'arête attribuée, ni d'arête de* G^0. *La mise à jour des variables duales avec* $u_i \leftarrow u_i - \delta$ *sur* LU *et* $v_j \leftarrow v_j + \delta$ *sur* SV *est telle que* $(\mathbf{X}, (\mathbf{u}, \mathbf{v}))$ *et* $\mathbf{C}$ *satisfont Eq. 10.*

Cette proposition montre qu'au moins une arête est ajoutée à G^0, garantissant ainsi que $LU \setminus SU \neq \emptyset$ à chaque itération. Remarquons que le calcul de δ s'effectue efficacement grâce à π, qui sauvegarde, pour chaque $i \in \mathcal{U}$, le coût minimum à l'ensemble des $j \in SV$.

La construction de l'arbre dépend de la structure de données utilisée pour coder l'ensemble $\mathcal{U} \setminus SU$. Dans nos expérimentations, nous avons utilisé une stratégie FIFO. Dans ce cas, l'ensemble $\mathcal{U} \setminus SU$ et les ensembles LU, $\mathcal{U} \setminus LU$ et SU peuvent efficacement être représentés par une permutation de $\mathcal{U}$ étant donné que $\mathcal{U} = SU \cup (LU \setminus SU) \cup (\mathcal{U} \setminus LU)$. Les ensembles $\mathcal{U} \setminus LU$ et LU ont au plus n éléments, et SV a au plus $\min\{n,m\}+1$ éléments. Donc chaque itération a une complexité temporelle en $O(2n + \min\{n,m\})$. À chaque itération, un élément de $\mathcal{V}$ est ajouté à SV (initialement vide). Excepté pour la première itération, cet élément est obtenu à partir du dernier élément de $\mathcal{U}$ ajouté à SU. Comme l'algorithme se termine avant d'augmenter SU, nous avons $|SU| = |SV| - 1$. De plus, comme $|SU| \leq n-1$ et $|SV| \leq m$, nous avons $|SV| = |SU| + 1 \leq n$. Par conséquent la boucle principale a $O(\min\{n,m\})$ itérations.

Proposition 6 *L'Algorithme 1 calcule un chemin augmenté en* $O((2n + \min\{n,m\}) \min\{n,m\})$.

3.3 Algorithme principal

Soit $\mathbf{C}$ une matrice de coûts d'édition. Soient $\mathbf{X}$ une ϵ-affectation partielle, représentée par la paire (ρ, ϱ), et $(\mathbf{u}, \mathbf{v})$ une paire de variables duales satisfaisant Eq. 10. L'ϵ-affectation partielle peut être nulle, c.-à-d. $\rho = \mathbf{0}_n$ et $\varrho_m = \mathbf{0}_m$. Dans ce cas nous avons $\mathbf{u} = \mathbf{0}_n$ et $\mathbf{v} = \mathbf{0}_m$.

L'Algorithme 2 complète itérativement l'ϵ-affectation partielle en augmentant l'ensemble $\mathcal{V}$ jusqu'à ce que tous ses éléments soient attribués à un élément de $\mathcal{U}_\epsilon$, puis l'ensemble $\mathcal{U}$ est augmenté similairement pour que tous ses éléments soient attribués à un élément de $\mathcal{V}_\epsilon$. Pour cela, à chaque itération, un élément non-attribué est sélectionné par **assignCols** et un chemin augmenté enraciné en cet élément est calculé avec l'Algorithme 1. Ce chemin fournit une nouvelle paire $((\rho', \varrho'), (\mathbf{u}', \mathbf{v}'))$ où la nouvelle ϵ-affectation (ρ', ϱ') est construite à partir de **pred** et de (ρ, ϱ) (Prop. 4) en $O(\min\{n,m\})$. De plus, $((\rho', \varrho'), (\mathbf{u}', \mathbf{v}'))$ et $\mathbf{C}$ satisfont Eq. 10 (Prop. 5), montrant que lorsque tous les éléments sont attribués, l'ϵ-affectation est optimale (Prop. 3). Comme m éléments de $\mathcal{V}$ peuvent être non-attribués, le premier appel à **assignCols** s'exécute en $O(m \min\{n,m\}(2n + \min\{n,m\}))$, *c.-à-d.* en $O(\min\{n,m\}^2(m + 2\max\{n,m\}))$. Le second appel à **assignCols** s'exécute en $O(\min\{n,m\}^2(n + 2\max\{n,m\}))$, et nous avons le résultat suivant.

Algorithme 2 : Calcul d'une ϵ-affectation.

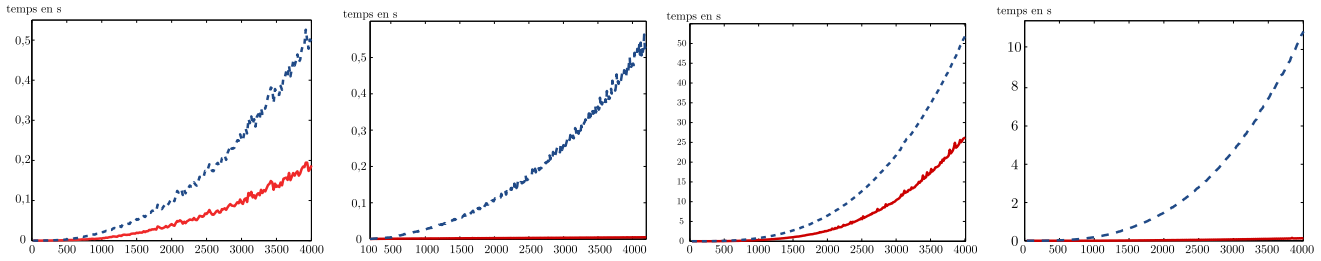
```
1  début HungarianLSAPE(C, ρ, ϱ, u, v)
2      (ρ, ϱ, u, v) ←assignCols(C, ρ, ϱ, u, v)
3      (ϱ, ρ, v, u) ←assignCols(C^T, ϱ, ρ, v, u)
4      retourner ((ρ, ϱ), (u, v))
5  début assignCols(C, ρ, ϱ, u, v)
6      (n, m) ← nombre de lignes−1 et de colonnes−1 de C
7      pour chaque k ∈ {1, ..., m} | ϱ_k = 0 faire
8          (i, j, u, v, pred) ← Augment(k, C, ρ, ϱ, u, v)
9          // mise à jour de l'ϵ-attribution partielle
10         si i = n + 1 alors r ← ϱ_j, ϱ_j ← i, i ← r
11         sinon j ← 0
12         tant que j ≠ k faire
13             j ← pred_i, ρ_i ← j
14             r ← ϱ_j, ϱ_j ← i, i ← r
15     retourner (ρ, ϱ, u, v)
```

Proposition 7 *L'Algorithme 2 résout le LSAPE (Eq. 4), et son problème dual, en* $O(\min\{n,m\}^2 \max\{n,m\})$ *en temps et en* $O(nm)$ *en mémoire.*

Notons que les complexités sont plus faibles que celles de l'algorithme Hongrois équivalent utilisé pour résoudre le sLSAPE (Eq. 1) dans le contexte de la bGED (Sec. 1). Rappelons que le LSAPE et le sLSAPE sont équivalents. L'amélioration de la complexité est d'autant plus importante que n et m sont grands et que la différence $|n-m|$ l'est aussi.

4 Expérimentations

Matrices de coûts synthétiques. Nous avons comparé la complexité temporelle de l'Algorithme 2, qui résout le LSAPE (Eq. 4), à celle de l'algorithme Hongrois proposé dans [6, 3], qui résout le LSAP dont le sLSAPE (Eq. 1) [9, 7]. Nous avons considéré trois types de matrices de coûts d'édition $\mathbf{C}$: (1) des matrices aléatoires (Fig. 3(a),3(b)), (2) des matrices de terme général $c_{i,j} = i \cdot j$ (Fig. 3(c)), et (3) des matrices de terme général $c_{n-i+1,m-j+1} = i \cdot j$ si $i \in \{1, \ldots, n\}$ et $j \in \{1, \ldots, m\}$, et la dernière ligne et la



(a) $n = m$ (b) $n = 100, m \in \{100, ..., 4100\}$ (c) $m = 2n, c_{ij} = i \cdot j$ (d) $m = 2n, c_{n-i+1,m-j+1} = i \cdot j$

FIGURE 3 – Temps d'exécution (en secondes) de l'algorithme Hongrois original (pointillés bleu) et de notre approche (ligne rouge), en fonction du nombre $n + m$ d'éléments.

dernière colonne sont égales à l'avant-dernière ligne et à l'avant-dernière colonne (Fig. 3(d)).

Dans tous les cas l'Algorithme 2 s'est exécuté plus rapidement, avec le comportement annoncé à la section précédente suite au calcul des complexités. En particulier lorsque $n = \min\{n, m\}$ est fixe (Fig. 3(b)) et $m = \max\{n, m\}$ est suffisamment grand, ce dernier prédomine dans la complexité de l'Algorithme 2 (Prop. 7) qui devient linéaire, tandis que celle de la résolution du sLSAPE par l'algorithme Hongrois reste cubique, *c.-à-d.* en $O(m^3)$. Dans le deuxième cas (Fig. 3(c)), connu pour être proche du pire cas pour l'algorithme Hongrois, l'amélioration est non négligeable, comparable à celle obtenue pour les matrices aléatoires (Fig. 3(a)). Pour le troisième cas (Fig. 3(d)), les solutions intègrent beaucoup de suppressions et d'insertions (dernière ligne et dernière colonne), ce qui pénalise considérablement le calcul d'une bijection résolvant le sLSAPE, qui doit systématiquement attribuer les $n + m$ éléments nuls. L'Algorithme 2 évite de calculer de telles attributions, inutiles en terme de coût d'édition.

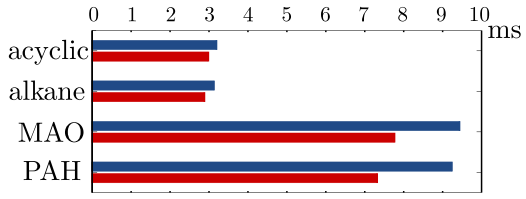
Distance d'édition bipartie entre graphes. L'efficacité de l'Algorithme 2 a ensuite été testée dans le cadre du calcul de la distance d'édition bipartie entre graphes (bGED). Pour cela, nous avons remplacé l'algorithme Hongrois utilisé habituellement dans l'état de l'art [9, 7], par l'Algorithme 2. Nous avons utilisé la bGED décrite dans [5], avec les mêmes valeurs de coûts.

Les expérimentations ont été effectuées sur des données de chémoinformatique [1] représentées par des graphes : étiquetés et acycliques (Acyclic), non étiquetés et acycliques (Alkane), étiquetés et cycliques (MAO), non étiquetés et cycliques (PAH). Acyclic et Alkane sont composés de graphes ayant une taille moyenne de 9 nœuds et environ de 20 pour MAO et PAH. Les temps de calculs moyens de la bGED (Fig. 4(a)) en résolvant le sLSAPE par l'algorithme Hongrois (barres bleues), et en résolvant le LSAPE par l'Algorithme 2 (barres rouges), montrent que le gain en temps est d'autant plus important avec ce dernier que la taille des graphes l'est aussi.

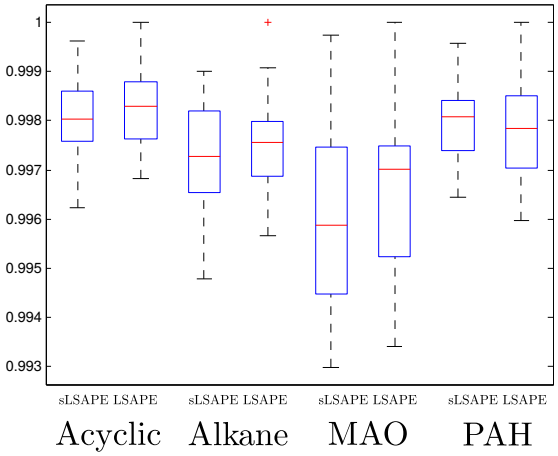
Comme il peut y avoir plusieurs solutions au problème d'appariement, qui correspondent au même minimum global de Eq. 1, les deux approches peuvent fournir des solutions différentes. Elles seront associés à des chemins d'édition différents, donc à des bGED différentes, et par conséquent à des approximations différentes de la GED. De plus, pour chaque algorithme, la solution calculée par les deux algorithmes peut être influencée par l'ordre dans lequel les nœuds des graphes sont indexés dans la matrice de coûts. Nous avons mesuré l'importance de ces phénomènes, en répétant 30 fois le calcul de la bGED pour chaque jeu de données de chemoinformatique, avec une permutation aléatoire des indices des nœuds à chaque répétition. Les résultats des calculs (Fig. 4(b)) montrent que la distance moyenne est faiblement influencée par ces permutations aléatoires : la différence entre la moyenne minimale et la moyenne maximale est inférieure à 0.7%. De plus, on peut observer que les distances moyennes sont similaires avec le sLSAPE et le LSAPE. Ceci s'est précisé statistiquement par un double test de Student (p-value de $0,01$), montrant que la GED est approximée par les deux approches avec une précision équivalente.

Afin de mesurer l'efficacité de l'Algorithme 2 sur de plus grands graphes, un jeu de données synthétiques a été créé. Il contient des graphes ayant la même distribution d'étiquettes de nœuds et d'arêtes, et le même ratio entre le nombre de nœuds et le nombre d'arêtes, que les graphes inclus dans le jeu de données MAO. Pour un nombre donné de nœuds, un jeu est composé de 100 paires de graphes, chaque paire étant composée d'un graphe source et d'un graphe cible. Chaque graphe cible a été créé à partir d'un graphe source en retirant un nœud et en changeant l'étiquette d'un autre nœud. La distance d'édition entre les deux graphes d'une même paire est alors définie par le coût associé à ces deux opérations d'édition sur les nœuds ainsi que l'ensemble des opérations induites sur les arêtes. La distance d'édition entre chaque paire de graphes est d'environ 10. Nous avons créé 13 jeux de données ayant un nombre de nœuds par graphe allant de 10 à 500. Comme auparavant, le gain en temps relativement à la résolution du sLSAPE est d'autant plus important que la taille des graphes l'est également (Fig. 4(c)). Nous pouvons observer que l'utilisation de l'Algorithme 2 permet de diviser les temps de calcul d'un facteur 15 pour les graphes ayant 500

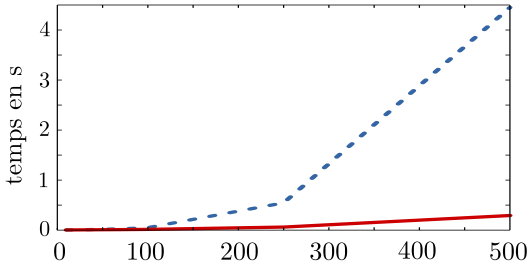
1. https://iapr-tc15.greyc.fr/links.html



(a) Temps de calcul moyen (en ms) pour les données de chémoinformatique : sLSAPE (barres bleues supérieures), LSAPE (barres rouges inférieures)



(b) Boxplot des bGED moyennes obtenues pour 30 répétitions du calcul sur chaque jeu de données, normalisées par $\max\{\max_i\{\bar{d}_i^{\text{sLSAPE}}\}, \max_i\{\bar{d}_i^{\text{LSAPE}}\}\}$, où $\bar{d}_i$ est la distance moyenne obtenue pour la répétition i



(c) Temps (en secondes) de calcul cumulés pour plusieurs tailles de graphes synthétiques : sLSAPE (pointillés bleus), LSAPE (rouge)

FIGURE 4 – Comparaisons de la bGED avec l'algorithme Hongrois (sLSAPE) et Alg. 2 (LSAPE).

nœuds, tout en restant inférieur à la demi-seconde, alors que le sLSAPE est résolu en plusieurs secondes.

Conclusion

Nous avons proposé une nouvelle formulation du problème de transformation optimale d'un ensemble en un autre ensemble, par une séquence d'opérations d'édition, restreintes ici aux substitutions et insertions/suppressions. Ce problème est habituellement exprimé sous la forme d'un LSAP et résolu avec l'algorithme Hongrois en $O((n+m)^3)$ en temps et en $O((n+m)^2)$ en mémoire. Notre analyse du problème montre qu'il peut s'écrire comme un programme binaire linéaire proche du LSAP. Nous avons donc adapté l'algorithme Hongrois pour résoudre le problème en $O(\min\{n,m\}^2 \max\{n,m\})$ en temps et en $O(mn)$ en mémoire. Le gain est d'autant plus important que la taille des ensembles, ainsi que leur différence, sont grandes. Ce comportement est confirmé expérimentalement. En particulier, notre approche permet de calculer la distance d'édition bipartie entre graphes plus rapidement, et traiter ainsi de plus grands graphes, sans perdre en qualité d'approximation de la distance d'édition.

Références

[1] S. Bougleux and L. Brun. Linear sum assignment with edition. Technical report, Normandie Univ, GREYC, 2016.

[2] H. Bunke. Error correcting graph matching : on the influence of the underlying cost function. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 21(9) :917–922, 1999.

[3] R. Burkard, M. Dell'Amico, and S. Martello. *Assignment Problems*. SIAM, 2009.

[4] M. Ferrer, F. Serratosa, and K. Riesen. A first step towards exact graph edit distance using bipartite graph matching. In *GbRPR*, volume 9069 of *LNCS*, pages 77–86. 2015.

[5] B. Gaüzère, S. Bougleux, K. Riesen, and L. Brun. Approximate Graph Edit Distance Guided by Bipartite Matching of Bags of Walks. In *S+SSPR*, volume 8621 of *LNCS*, pages 73–82. 2014.

[6] E. L. Lawler. *Combinatorial Optimization : Networks and Matroids*. Holt, Rinehart and Winston, 1976.

[7] K. Riesen and H. Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision Comp.*, 27 :950–959, 2009.

[8] K. Riesen and H. Bunke. Improving bipartite graph edit distance approximation using various search strategies. *Pattern Recognition*, 28(4) :1349–1363, 2015.

[9] K. Riesen, M. Neuhaus, and H. Bunke. Bipartite graph matching for computing the edit distance of graphs. In *GbRPR*, volume 4538 of *LNCS*, pages 1–12. 2007.

[10] F. Serratosa. Fast computation of bipartite graph matching. *Pattern Recognition Letters*, 45 :244–250, 2014.

[11] S. Serratosa. Speeding up fast bipartite graph matching through a new cost matrix. *Int. Journal of Pattern Recognition*, 29(2), 2015.

[12] G. Sierksma and Y. Zwols. *Linear and Integer Optimization : Theory and Practice*. Chapman and Hall/CRC, 3rd edition, 2015.

[13] Z. Zeng, A. K. H. Tung, J. Wang, J. Feng, and L. Zhou. Comparing stars : On approximating graph edit distance. *Proc. of the VLDB Endowment*, 2(1) :25–36, 2009.