Approximating Graph Edit Distance using GNCCP*

Benoît Gaüzère[1], Sébastien Bougleux[2], and Luc Brun[3]

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

Abstract. The graph edit distance (GED) is a flexible and widely used dissimilarity measure between graphs. Computing the GED between two graphs can be performed by solving a quadratic assignment problem (QAP). However, the problem is NP complete hence forbidding the computation of the optimal GED on large graphs. To tackle this drawback, recent heuristics are based on a linear approximation of the initial QAP formulation. In this paper, we propose a method providing a better local minimum of the QAP formulation than our previous proposition based on IPFP. We adapt a convex concave regularization scheme initially designed for graph matching which allows to reach better local minimum and avoids the need of an initialization step. Several experiments demonstrate that our method outperforms previous methods in terms of accuracy, with a time still much lower than the computation of a GED.

1 Introduction

Graphs provide a flexible framework to represent data including relationships between elements. In addition, graphs come with an underlying powerful theory which allows to infer a lot of information from this representation. However, graph's space is not an euclidean space. This last point avoids the use of classic machine learning methods mainly designed to operate in euclidean spaces. Several approaches aim to bridge the gap between graph space and euclidean spaces in order to combine machine learning and graphs. A first historical approach consists in embedding graphs onto euclidean spaces by computing a set of descriptors describing graphs. Even if this method is straightforward and can be easily controlled by choosing the information to keep, the loss of structural information induced by the euclidean embedding may constitute a major drawback for some applications. An extension of this approach is based on the well known *kernel trick*. Kernel methods implicitly define a scalar product between embedding of graphs in some Hilbert spaces, without requiring an explicit formulation of the Reproducing Kernel Hilbert Space (RKHS) associated to the kernel. The

* This paper appears in Structural, Syntactic, and Statistical Pattern Recognition (S+SSPR), LNCS 10029, pp 496–506, Springer, 2016 (http://dx.doi.org/10.1007/978-3-319-49055-7_44)

use of this trick allows to combine graphs and powerful kernel methods such as Kernel Ridge Regression or Support Vector Machines. However, constraints on the design of graph kernels often complexify the consideration of fine similarities or dissimilarities between graphs.

Another strategy consists in operating directly in the graph space. One of the most used measure is the graph edit distance (GED) [2,14]. The GED of two graphs may be understood as the minimal amount of distortion required to transform a source graph into a target one. This distortion is encoded by an edit path, defined as a sequence of edit operations which includes nodes and edges substitutions, removals and insertions. Depending on the context, each edit operation e included in an edit path γ is associated to a non-negative cost $c(e)$. The sum of all edit operation costs included within γ defines the cost $A(\gamma)$ associated to this edit path. The minimal cost $A(\gamma^\star)$ among all edit paths $\Gamma(G_1, G_2)$ defines the GED between $G1$ and $G2$:

$$\mathrm{GED}(G_1, G2) = A(\gamma^\star) = \min_{\gamma \in \Gamma(G1, G2)} A(\gamma) \tag{1}$$

with $A(\gamma) = \sum_{e \in \gamma} c(e)$. The edit path $\gamma^\star$ corresponds to an optimal edit path. GED has been widely used by the structural pattern recognition community [15,4,11,12] despite the fact that such distance comes along with several drawbacks. First of all, computing the GED of two graphs requires to find a path having a minimal cost among all possible paths, which is a NP-complete problem [8]. Computing an exact GED is generally done using $A^\star$ algorithm. In practice, due to its high complexity, the computation of an exact GED is intractable for graphs having more than 10 nodes [11,4]. Such a limitation restricts applications of GED on real datasets, hence motivating the graph community to focus on heuristics providing suboptimal solutions of Eq. 1.

In this paper, we propose a method to compute an accurate GED approximation. In Section 2, we first show the close relationship between GED and nodes' mappings. This relationship leads to the formulation of the graph edit distance as a quadratic assignment problem (QAP). Then, Section 3 first reviews the method used in [13] to find a local minimum of the QAP associated to GED. Then, we propose a more accurate and reliable optimizer of our QAP formulation. Section 4 shows the effectiveness of our proposal on chemoinformatics problems.

2 GED as a quadratic assignment problem

Exact GED computation is based on a tree search algorithm which finds an optimal edit path among possible ones. The resulting algorithm follows the intuition given by the formal definition of the GED. However, this approach has an exponential complexity with the number of nodes.

Another formulation of the GED is based on its relationship with nodes' mapping. First, let us consider two sets of nodes $V_1 = \{v_1, \ldots, v_n\}$ and $V_2 =$

$\{u_1, \ldots, u_m\}$ of two graphs G_1 and G_2, with $n = |V_1|$ and $m = |V_2|$. The substitution of node $v_i \in V_1$ to node $u_j \in V_2$ can be encoded by mapping v_i to u_j. Insertions/removals of nodes can not be encoded by a node to node mapping since a removed node will no longer appear in the target graph. Therefore, we augment the two sets V_1 and V_2 by adding enough null elements ϵ to encode the removal or insertion of any node: $V_1^\epsilon = V_1 \cup \{\epsilon_1, \ldots, \epsilon_m\}$ and $V_2^\epsilon = V_2 \cup \{\epsilon_1, \ldots, \epsilon_n\}$. Note that $|V_1^\epsilon| = |V_2^\epsilon| = n + m$. Insertion of a node $u_j \in V_2^\epsilon$ can now be represented by mapping an ϵ element to u_j. In the same way, a mapping $v_i \to \epsilon$ encodes the removal of node v_i. It has be shown [1,10] that, under mild assumptions on edit paths, each mapping between the two sets V_1^ϵ and V_2^ϵ of two graphs G_1 and G_2 corresponds to one and only one edit path transforming G_1 into G_2. Considering this bijective relationship, finding the optimal edit path relies on finding an optimal mapping between the two sets V_1^ϵ and V_2^ϵ which minimizes the total mapping cost of nodes and edges.

Let us consider a bijective function $\phi : V_1^\epsilon \to V_2^\epsilon$. The mapping cost induced by this mapping can be defined as a sum of two terms:

$$S(V_1^\epsilon, V_2^\epsilon, \phi) = L_v(V_1^\epsilon, V_2^\epsilon, \phi) + Q_e(V_1^\epsilon, V_2^\epsilon, \phi) \tag{2}$$

The first term $L_v(V_1^\epsilon, V_2^\epsilon, \phi)$ encodes the cost induced by nodes' mappings and the second term $Q_e(V_1^\epsilon, V_2^\epsilon, \phi)$ the cost induced by edges' mappings. Let us consider the matrix $\mathbf{X} \in \mathit{\Pi}$, with $\mathit{\Pi}$ representing the set of binary doubly stochastic matrices: $\mathbf{X}$ encodes a mapping function Φ iff $\mathbf{X}_{i,j} = 1$ with $\phi(i) = j$ and $\mathbf{X}_{i,j} = 0$ otherwise. $\mathbf{X}$ corresponds then to a mapping or permutation matrix. The cost induced by nodes mapping can then be defined as:

$$L_v(V_1^\epsilon, V_2^\epsilon, \phi) = \sum_{i=1}^{n} c(v_i \to \phi(v_i)) + \sum_{i=1}^{m} c(\epsilon_i \to \phi(\epsilon_i)) = \sum_{i=1}^{n+m} \sum_{j=1}^{n+m} \mathbf{C}_{i,j} \mathbf{X}_{i,j} \tag{3}$$

with $\mathbf{C} \in \mathbb{R}_+^{(n+m)\times(n+m)}$ encoding costs associated to edit operations on nodes.

The bipartite GED, proposed by [11], can be understood as an approximation of the problem expressed in Eq. 2. In bipartite approach, only the linear term $L_v(V_1^\epsilon, V_2^\epsilon, \phi)$ is considered. Finding the optimal mapping w.r.t. $L_v(V_1^\epsilon, V_2^\epsilon, \phi)$ corresponds to a Linear Sum Assignment Problem (LSAP) defined as:

$$\mathbf{X}^\star = \underset{\mathbf{X} \in \mathit{\Pi}}{\operatorname{argmin}} \operatorname{vec}(\mathbf{C})^T \operatorname{vec}(\mathbf{X}) \tag{4}$$

where $\operatorname{vec}(\mathbf{X})$ encodes the vectorization of matrix $\mathbf{X}$, i.e. the concatenation of its rows into one single vector. For a sake of clarity, $\operatorname{vec}(\mathbf{X})$ will be denoted $\mathbf{x}$.

Eq. 4 can be resolved using well known algorithms such as Hungarian [6] in $O((n+m)^3)$. The optimal mapping encoded by $\mathbf{X}^\star$ encodes a set of nodes edit operations. Using simple graphs, edges' edit operations are inferred from the nodes' mapping to complete the edit path. Finally, the sum of costs associated to this edit path is taken as the approximation of GED. Note that since this edit path may not be optimal, the computed GED may be an overestimation of the exact one.

Table 1. Edit operations and costs encoded in matrix $\mathbf{C}$

i	j	edit operation	cost $\mathbf{C}_{i,j}$
$\in V_1$	$\in V_2$	substitution of v_i by u_j	$c(v_i \rightarrow u_j)$
$\in V_1$	$\notin V_2$	removal of v_i	$c(v_i \rightarrow \epsilon)$
$\notin V_1$	$\in V_2$	insertion of u_j	$c(\epsilon \rightarrow u_j))$
$\notin V_1$	$\notin V_2$	none	0

Each entry $\mathbf{C}_{i,j}$ of matrix $\mathbf{C}$ encodes the cost of the edit operation induced by mapping the i-th element of V_1^ϵ to the j-th element of V_2^ϵ. Table 1 summarizes the general term $\mathbf{C}_{i,j}$. Computing an optimal mapping using only $\mathbf{C}$ as defined in Table 1 will only take into account information about nodes, ignoring all the structural information encoded by edges. However, since the quality of the GED approximation depends directly from the computed mapping, costs encoded in matrix $\mathbf{C}$ may also estimate costs induced by edit operations on edges. To include this information, methods based on bipartite graph matching [11,4,3] define a cost matrix $\mathbf{C}$ augmented with the costs induced by the mapping of nodes' neighborhoods. The differences between different methods are mainly based on the size of the radius considered around each node. Results presented by [3] have shown that, as expected, the quality of the approximation increases as long as we take into account a larger radius around each node, and thus more structural information. However, we also observe an asymptotic gain for radius greater than 3, hence showing the limit of this method. Obviously, computational times also increase as we increase the radius.

Cost induced by edges' edit operations are inferred from edges' mapping. However, mapping ϕ is only defined between two extended sets of nodes and an edge is defined by a couple of nodes. Therefore, edge mappings can be directly deduced from the mapping of their incident nodes. Let $e = (v_i, v_j) \in E_1$, its mapped edge in V_2^ϵ corresponds to $e' = (\phi(v_i), \phi(v_j))$. If $e' \in E_2$, then the mapping corresponds to an edge substitution, else if $e' \notin E_2$, then the edge has been removed. Reciprocally, if $e' = (u_i, u_j) \in E_2$ and $e = (\phi^{-1}(u_i), \phi^{-1}(u_j)) \notin E_1$, then this edge has been inserted. The cost associated to edges' operations is encoded by a sum of three terms, one for each kind of edit operation:

$$Q_e(V_1^\epsilon, V_2^\epsilon, \phi) = \underbrace{\sum_{\substack{(i,j)\in E_1, \phi(i)=k,\\ \phi(j)=l, (k,l)\in E_2}} c((i,j)\rightarrow(k,l))}_{\text{substitutions}} + \underbrace{\sum_{\substack{(i,j)\in E_1, \phi(i)=k,\\ \phi(j)=l, (k,l)\notin E_2}} c((i,j)\rightarrow\epsilon)}_{\text{removals}} + \underbrace{\sum_{\substack{(k,l)\in E_2, i=\phi^{-1}(k),\\ j=\phi^{-1}(l), (i,j)\notin E_1}} c(\epsilon\rightarrow(k,l))}_{\text{insertions}} \tag{5}$$

Table 2. General term of matrix $\mathbf{D}$

(i,j)	(k,l)	edit operation	$\mathbf{D}_{ik,jl}$
$\in E_1$	$\in E_2$	substitution of (i,j) by (k,l)	$c((i,j) \to (k,l))$
$\in E_1$	$\notin E_2$	removal of (i,j)	$c((i,j) \to \epsilon)$
$\notin E_1$	$\in E_2$	insertion of (k,l) into E_1	$c(\epsilon \to (k,l))$
$\notin E_1$	$\notin E_2$	none	0

Following the quadratic assignment formulation given by [1], the term $Q_e(V_1^\epsilon, V_2^\epsilon, \phi)$ can be expressed as a quadratic term depending on a matrix $\mathbf{D} \in \mathbb{R}^{(n+m)^2 \times (n+m)^2}$ encoding edges' mapping costs :

$$Q_e(V_1^\epsilon, V_2^\epsilon, \phi) = \mathbf{x}^T \mathbf{D} \mathbf{x} \tag{6}$$

Table 2 summarizes all mapping costs encoded within matrix $\mathbf{D}$. Note that the computation of $\mathbf{x}^T\mathbf{D}\mathbf{x}$ has, in a naive computation, a $\mathcal{O}((n+m)^4)$ computational and memory complexity. However, the vector $\mathbf{x} \in \{0,1\}^{(n+m)^2}$ is sparse since it encodes a mapping and thus only $(n+m)$ terms differs from 0, which reduces the computational complexity if we only process them (Alg. 2, lines 1 and 3). Therefore, with a proper approach, the computation of $(\mathbf{x}^T\mathbf{D}\mathbf{x})$ has thus a $\mathcal{O}((n+m)^2)$ complexity. Note that this complexity is given for $\mathbf{x}$ encoding a permutation matrix. If more than $n+m$ elements of $\mathbf{x}$ are different from 0, the associated complexity will be higher. We also avoid the storage of $\mathbf{D}$ since we can efficiently compute each term trough a function d (Alg.2, line 7) which computes $\mathbf{D}_{ij,kl}$ according to Table 2. The overall memory complexity can thus be reduced to the storage of $n+m$ mappings.

Plugging Eq. 3 and Eq. 6 into Eq. 2, the cost of transforming G_1 into G_2 including both nodes and edge costs according to a mapping ϕ can then be written as:

$$S(V_1^\epsilon, V_2^\epsilon, \phi) = Q_e(V_1^\epsilon, V_2^\epsilon, \phi) + L_v(V_1^\epsilon, V_2^\epsilon, \phi) = \mathbf{x}^T\mathbf{D}\mathbf{x} + \mathbf{c}^T\mathbf{x} \tag{7}$$

Algorithm 1 Computation of $\mathbf{x}^T\mathbf{D}\mathbf{x}(\phi)$

1: **for all** $j,l \mid \mathbf{x_{jl}} \neq 0$ **do** // $\mathcal{O}(n+m)$ if $x \in \Pi$
2: $\quad(\mathbf{x}^T\mathbf{D})_{jl} = 0$
3: $\quad$**for all** $i,k \mid \mathbf{x_{ik}} \neq 0$ **do** // $\mathcal{O}(n+m)$ if $x \in \Pi$
4: $\quad\quad\delta_{ij} \leftarrow (i,j) \in E_1$
5: $\quad\quad\delta_{kl} \leftarrow (k,l) \in E_2$
6: $\quad\quad(\mathbf{x}^T\mathbf{D})_{jl} \leftarrow (\mathbf{x}^T\mathbf{D})_{jl} + \mathrm{d}(\delta_{ij}, \delta_{kl}, c(\cdot \to \cdot)$ // see Table 2
7: $\quad$**end for**
8: **end for**
9: **return** $\mathbf{x}_{k+1}$

with $\mathbf{D}$ and $\mathbf{c} = \text{vec}(\mathbf{C})$ being fully dependant on the two input graphs G_1 and G_2 and $\mathbf{x}$ encoding the mapping ϕ. Given a pair of graphs, Eq. 7 can be rewritten as only depending on $\mathbf{x}$ as $S(\mathbf{x}) = \mathbf{x}^T\mathbf{D}\mathbf{x} + \mathbf{c}^T\mathbf{x}$.

Given the Alg. 2, each edge is processed twice, i.e. once as (i, j) and once as (j, i). In case of undirected graphs, it will thus count each edit operation twice. We have $\mathbf{x}^T\mathbf{D}\mathbf{x} = \frac{1}{2}\mathbf{x}^T(\mathbf{D} + \mathbf{D}^T)\mathbf{x}$ and to handle both directed and undirected graphs, we introduce a matrix Δ :

$$S(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\Delta\mathbf{x} + \mathbf{c}^T\mathbf{x} \qquad \text{with } \Delta = \begin{cases} \mathbf{D} & \text{if } G_1 \text{ and } G_2 \text{ are undirected} \\ \mathbf{D} + \mathbf{D}^T & \text{else} \end{cases} \tag{8}$$

Note that $\mathbf{D}$ is symmetric if both G_1 and G_2 are undirected [1]. Hence, Δ is symmetric in both cases. Including the linear part into the quadratic one leads to:

$$S(\mathbf{x}) = \mathbf{x}^T\widehat{\Delta}\mathbf{x}, \text{ with } \widehat{\Delta} = \frac{1}{2}\Delta + \text{diag}(\mathbf{c}) \tag{9}$$

Then, computing the GED of two graphs leads to minimize the quadratic function given in Eq. 9 under the constraint that $\mathbf{x}$ is a permutation matrix. This problem corresponds thus to the QAP:

$$\text{GED}(G_1, G_2) = \min_{\mathbf{x} \in \Pi} \mathbf{x}^T\widehat{\Delta}\mathbf{x} \tag{10}$$

Note that quadratic formulation of GED have already been proposed: [5] proposes a binary quadratic program transformed into a binary linear program but restricted to undirected and unlabeled edges. Slightly after, [10] proposed another quadratic program but where optimization is only focused on node substitutions.

3 Resolution of the QAP

As stated in Introduction, computing an exact GED is intractable for most applications. Reducing the complexity relies thus on finding the best possible mapping, i.e. the one having the lowest mapping cost, in an acceptable computational time. Rather than finding an exact solution of a linear approximation of the problem, another strategy consists in optimizing the QAP associated to GED (Eq. 10). However, since Δ is neither positive nor negative definite, finding a global minimum of Eq.10 is, conversely to LSAP, a NP-complete problem. Since this problem is NP-complete, most of algorithms find thus approximate solutions by relaxing the original problem to the set of doubly stochastic matrices. As we resolve a QAP and not a generic quadratic problem, the solution should be a permutation matrix encoding a mapping. Since most of classic approaches used to resolve quadratic problems do not take this constraint into account, the final projection of the continuous solution to a permutation matrix may alter the approximation of the GED.

Interger-Projected Fixed Point (IPFP) method [7] has originally been proposed to resolve graph matching problems formalized as a maximization of a

Algorithm 2 IPFP($\mathbf{x_0}, \hat{\Delta}, \varepsilon$)

1: $k = 0$
2: **repeat**
3: $\quad \mathbf{b}^\star \leftarrow \text{argmin}_{\mathbf{b} \in \Pi} \quad (\mathbf{x_k}^T \hat{\Delta})$
4: $\quad \alpha \leftarrow (\mathbf{x_k}^T \hat{\Delta})\mathbf{b}^\star - 2S(\mathbf{x_k}) + \mathbf{c}^T \mathbf{x_k}$
5: $\quad \beta \leftarrow S(\mathbf{b}^\star) + S(\mathbf{x_k}) - (\mathbf{x_k}^T \hat{\Delta})\mathbf{b}^\star + \mathbf{c}^T \mathbf{x_k}$
6: $\quad t^\star \leftarrow -\alpha/2\beta$
7: $\quad \mathbf{x_{k+1}} \leftarrow \mathbf{x_k} + t^\star(\mathbf{b}^\star - \mathbf{x_k})$
8: $\quad k \leftarrow k + 1$
9: **until** $||\mathbf{x_k} - \mathbf{x_{k-1}}|| < \varepsilon$
10: **return** $\text{argmax}_{b \in \Pi} \mathbf{x}_k^T \mathbf{b}$

QAP. Conversely to classical methods used to resolve quadratic problems, IPFP allows to approximate the optimal solution by a gradient descent approach taking into account the nature of the solution. The algorithm mainly iterates over two steps: (i) compute a gradient direction which maximizes a linear approximation of the QAP in discrete domain and (ii) maximize the QAP in the continuous domain between the previous solution and the one found at step (i).

IPFP has been adapted from its original definition to GED computation [1,13] by considering a minimization rather than a maximization and by considering the matrix $\hat{\Delta}$. The algorithm corresponding to this adaptation is shown in Alg. 3. Line 3 corresponds to the computation of the gradient direction. This step corresponds to a LSAP and can be efficiently computed using an Hungarian algorithm such as in bipartite GED framework. Then, a line search is performed to minimize the quadratic objective function in continuous domain. This optimization relies on the computation of a step size $t^\star$ (line 6) which can be computed analytically. Finally, the solution at convergence is projected onto the set of mapping matrices (line 10). Note that the stochastic matrix $\mathbf{x_k}$ is projected by line 10 onto its closest permutation matrix. However, the distance betwenn $\mathbf{x_k}$ and its projection can not be bounded a priori. In [1,13], we reported the results obtained by using this approach and, as expected, it reaches a better accuracy than LSAP based methods. However, due to the non convexity of the objective function, the accuracy of the approximation is strongly dependant on the initialization $\mathbf{x_0}$.

Graduated NonConvexity and Concativity Procedure (GNCCP) [9,16] is a path following algorithm which consists in approximating the solution of a QAP by using a convex-concave relaxation. This approach brings several advantages over IPFP. First, no initial mapping is required. This may avoid the accuracy variations induced by the initialization step observed with IPFP [13]. Second, this algorithm converges towards a mapping matrix [9]. This second property allows to avoid the projection step required at the end of IPFP algorithm which may alter the accuracy of the approximation. GNCCP algorithm is based on a weighted sum of a convex and a concave relaxation of Eq. 9:

$$S_\zeta(\mathbf{x}) = (1 - |\zeta|)S(\mathbf{x}) + \zeta \mathbf{x}^T \mathbf{x} \tag{11}$$

Algorithm 3 GNCCP($\mathbf{x}_0, \mathbf{c}, \Delta, k_{\max}$)

1: $\zeta = 1, d = 0.1$, $\mathbf{x} = \mathbf{0}$
2: **while** $\zeta > -1$ & $\mathbf{x} \notin \mathbb{A}$ **do**
3: // Resolution of relaxed QAP according to ζ using Frank-Wolfe like algorithm
4: $\mathbf{x} \leftarrow \text{argmin}_{\mathbf{x} \in \Pi}(1 - |\zeta|)(\frac{1}{2}\mathbf{x}^T \hat{\mathbf{\Delta}} \mathbf{x}) + \zeta \mathbf{x}^T \mathbf{x}$
5: $\zeta \leftarrow \zeta - d$
6: **end while**
7: **return** $\mathbf{x}_{k+1}$

For $\zeta = 1$, $S_\zeta(\mathbf{x})$ is equal to $\mathbf{x}^T\mathbf{x}$ which corresponds to a fully convex objective function. Conversely, for $\zeta = -1$, $S_\zeta(\mathbf{x}) = -\mathbf{x}^T\mathbf{x}$ which now defines a concave function. GNCCP algorithm starts with $\zeta = 1$, and since the problem is convex, the initialization does not alter the result and will not influence the quality of the approximation as in [13]. Then, the algorithm, detailed in Alg. 3, smoothly interpolates convex and concave relaxations by passing from $\zeta = 1$ to $\zeta = -1$ with steps of size d until convergence is reached by having a mapping matrix or $\zeta = -1$. Note that the minimum of the concave relaxation is a mapping matrix [17,16]. Hence, conversely to IPFP, we do not need to perform a final projection step which may alter the approximation (end of Alg. 3).

For a given ζ, each iteration of GNCCP minimizes the quadratic functional defined in Eq. 11 using a Frank-Wolfe like algorithm. In this paper, we adapt IPFP algorithm used in [13] to perform this step and we remove last projection step (Alg. 3, line 10). However, since we optimize the relaxed objective function (Eq. 11), we have to update the linear subproblem and line search step. In our modified version of Alg. 3, the gradient direction $\mathbf{b}^\star$ has now to minimize $S_\zeta(\mathbf{x})$:

$$\frac{\partial S_\zeta(\mathbf{x})}{\partial \mathbf{x}} = (1 - |\zeta|)\frac{\partial S(\mathbf{x})}{\partial \mathbf{x}} + \zeta 2\mathbf{x}^T = (1 - |\zeta|)(\mathbf{x}^T \hat{\Delta}) + \zeta 2\mathbf{x}^T \tag{12}$$

Then, line 3 of Alg. 3 is updated with:

$$\mathbf{b}^\star \leftarrow \underset{\mathbf{b} \in \Pi}{\text{argmin}} \quad [(1 - |\zeta|)(\mathbf{x}^T \hat{\Delta}) + 2\zeta \mathbf{x}^T]\mathbf{b} \tag{13}$$

Given the gradient direction $\mathbf{b}^\star$, line search step has also to be updated to minimize $S_\zeta(\mathbf{x})$ rather than $S(\mathbf{x})$. After some calculus, lines 4 and 5 are updated with:

$$\alpha_\zeta \leftarrow [(1 - |\zeta|)(\mathbf{x_k}^T \hat{\Delta}) + 2\zeta \mathbf{x_k}^T]\mathbf{b}^\star - 2S_\zeta(\mathbf{x_k}) + (1 - |\zeta|)\mathbf{c}^T\mathbf{x_k} \tag{14}$$

$$\beta_\zeta \leftarrow S_\zeta(\mathbf{b}^\star) + S_\zeta(\mathbf{x_k}) - [(1 - |\zeta|)(\mathbf{x}^T \hat{\Delta}) + 2\zeta \mathbf{x_k}^T]\mathbf{b}^\star + (1 - |\zeta|)\mathbf{c}^T\mathbf{x_k} \tag{15}$$

Note that some of the terms of Eq. 14 and 15 have been already computed: $[(1 - |\zeta|)(\mathbf{x}^T \hat{\Delta}) + 2\zeta \mathbf{x_k}^T]\mathbf{b}^\star$ corresponds to the optimal cost computed by LSAP resolver on line 3 of Alg. 3, $\mathbf{c}^T\mathbf{x_k}$ to the linear term included into $\hat{\Delta}$ and computed for the linear subproblem and $S_\zeta(\mathbf{x_k})$ corresponds to the score of objective function computed at previous iteration. Finally, step size $t^\star$ is computed in the same way as in Alg. 3 with $t^\star = -\alpha_\zeta / 2\beta_\zeta$ (line 6).

Table 3. Accuracy and complexity scores. d is the average edit distance, e the average error and t the average computational time.

Algorithm	Alkane			Acyclic			PAH	
	d	e	t	d	e	t	d	t
A^*	15	-	1.29	17	-	6.02	-	-
LSAP [11]	35	18	$\simeq 10^{-3}$	35	18	$\simeq 10^{-3}$	138	$\simeq 10^{-3}$
LSAP Random Walks [4]	33	18	$\simeq 10^{-3}$	31	14	$\simeq 10^{-2}$	120	$\simeq 10^{-2}$
LSAP K-graphs[3]	26	11	2.27	28	9	0.73	129	2.01
$IPFP_{\text{Random init}}$	22.6	7.1	0.007	23.4	6.1	0.006	63	0.04
$IPFP_{\text{Init [4]}}$	20.5	5	0.006	20.7	3.4	0.005	52.5	0.037
$GNCCP$	16.7	1.2	0.46	18.8	1.5	0.33	41.8	6.24

4 Experiments

Our method based on GNCCP is tested on 3 real world chemical datasets[4]. Similarly to our previous papers using these datasets [3,4,13], edge and node substitutions costs have been set to 1 and 3 for insertions and deletions. To avoid bias due to arbitrary order of nodes in data files, each graph adjacency matrix is randomly permuted. We compare our method to others methods computing GED. First, we used an exact GED computed using $A^\star$ but only available for small graphs (line 1). Second, we show results obtained by LSAP based methods using different information around each node: incident edges and adjacent nodes [11] (line 2), bag of random walks up to 3 edges (line 3) [4], subgraphs up to a radius 3 (line 4) [3]. We also show results obtained by IPFP approach, detailed in Section 3, with two different initializations: a random one (line 5) and one using [4] (line 6). Finally, results obtained by our method with $d = 0.1$ are displayed (line 7). First of all, Table 3 shows clearly that our method allows to reach the best accuracy. Considering the two first datasets Alkane and Acyclic for which we have a ground truth computed using $A^\star$ (line 1), average error of approximation is divided by over 2 for acyclic and by over 4 for alkane, hence showing a very good approximation of GED. These improvements allow to reduce the relative error to about 10%. Concerning PAH dataset, the quality of the approximation must be deduced from average distance since we can not compute the exact GED. Since the edit distance is always overestimated, the lowest edit distance may correspond to a better approximation. The quality of our approximation is also validated on PAH dataset since the average edit distance is reduced by about 20%. Despite the good approximation offered by GNCCP, we also observe an important increase of computationnal times. This phenomenon is not surprising since GNCCP iterates over IPFP. For $d = 0.1$, we may perform about 20 iterations in the worst case (Alg. 3). In practice, we observe an average of 11 iterations before reaching convergence. Nonetheless, computational times increase by a factor 40. This increase is due to the computation of quadratic term $\mathbf{x}^T \Delta \mathbf{x}$. Indeed, for ζ far from 0, the matrix $\mathbf{x}$ may contains a lot of entries different from 0. Therefore, the complexity stated from Alg. 2 no longer holds.

[4] Datasets are available at `https://iapr-tc15.greyc.fr/links.html`

5 Conclusion

Considering the relationship between graph edit distance and nodes' mapping, we present in this paper a quadratic assignment problem formalizing the graph edit distance with substitutions, insertions and removals of nodes and edges. Following a previous optimization scheme based on a Frank Wolfe algorithm, we adapt a convex concave relaxation framework to the resolution of the QAP associated to edit distance. This approach does not require any initialization step and converges towards a mapping matrix. These two properties allow to compute a more reliable and robust approximation of the graph edit distance. The experiments conducted on real world chemical datasets valid our hypothesis.

References

1. S. Bougleux, L. Brun, V. Carletti, P. Foggia, B. Gaüzère, and M. Vento. A quadratic assignment formulation of the graph edit distance. Technical report, Normandie Univ, NormaSTIC FR 3638, France, 2015.
2. H Bunke and G Allermann. Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters*, 1(4):245–253, 1983.
3. V. Carletti, B. Gaüzère, L. Brun, and M. Vento. Approximate graph edit distance computation combining bipartite matching and exact neighborhood substructure distance. In *GbRPR*, volume 9069, pages 168–177. Springer, 2015.
4. B. Gaüzère, S. Bougleux, K. Riesen, and L. Brun. Approximate Graph Edit Distance Guided by Bipartite Matching of Bags of Walks. In *Structural, Syntactic and Statistical Pattern Recognition*, volume 8621, pages 73–82. Springer, 2014.
5. D. Justice and A. Hero. A binary linear programming formulation of the graph edit distance. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(8):1200–1214, 2006.
6. H.W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quaterly*, 2:83–97, 1955.
7. M. Leordeanu, M. Hebert, and R. Sukthankar. An integer projected fixed point method for graph matching and map inference. In *Advances in neural information processing systems*, pages 1114–1122, 2009.
8. C. Lin. Hardness of approximating graph transformation problem. In *5th Annual International Symposium on Algorithms and Computation*, volume 834 of *LNCS*, pages 74–82. Springer-Verlag, Berlin, 1994.
9. Z.-Y. Liu and H. Qiao. GNCCP–Graduated NonConvexity and Concavity Procedure. *Pattern Anal. Mach. Intell.*, 36(6):1258–1267, 2014.
10. M. Neuhaus and H. Bunke. A quadratic programming approach to the graph edit distance problem. In *GbRPR*, volume 4538 of *LNCS*, pages 92–102. Springer, 2007.
11. K. Riesen and H. Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision Comp.*, 27:950–959, 2009.
12. K. Riesen, S. Emmenegger, and H. Bunke. A novel software toolkit for graph edit distance computation. In *Graph-based Representations in Pattern Recognition*, volume 7877 of *LNCS*, pages 142–151. Springer, 2013.
13. L. Brun S. Bougleux, B. Gaüzère. Graph edit distance as a quadratic program. In *ICPR 2016.*, 2016. Submitted.
14. A. Sanfeliu and K.-S. Fu. A distance measure between attributed relational graphs for pattern recognition. *Systems, Man. and Cybernetics*, 13(3):353–362, 1983.

15. S. Serratosa. Speeding up fast bipartite graph matching through a new cost matrix. *Int. Journal of Pattern Recognition*, 29(2), 2015.
16. M. Zaslavskiy, F. Bach, and J.-P. Vert. A path following algorithm for the graph matching problem. *Pattern Anal. Mach. Intell.*, 31(12):2227–2242, 2009.
17. F. Zhou and F. De la Torre. Factorized graph matching. In *CVPR*, pages 127–134, 2012.