Approximate Graph Edit Distance Computation Combining Bipartite Matching and Exact Neighborhood Substructure Distance

Vincenzo Carletti†, Benoit Gaüzère†, Luc Brun‡, and Mario Vento†

†DIEM, Department of Information Engineering, Electrical Engineering and Applied Mathematics, University of Salerno, Italy
{mvento,vcarletti}@unisa.it
‡ GREYC CNRS UMR 6072, ENSICAEN, Caen, France
luc.brun@ensicaen.fr

Abstract. Graph edit distance corresponds to a flexible graph dissimilarity measure. Unfortunately, its computation requires an exponential complexity according to the number of nodes of both graphs being compared. Some heuristics based on bipartite assignment algorithms have been proposed in order to approximate the graph edit distance. However, these heuristics lack of accuracy since they are based either on small patterns providing a too local information or walks whose tottering induce some bias in the edit distance calculus. In this work, we propose to extend previous heuristics by considering both less local and more accurate patterns using subgraphs defined around each node.

1 Introduction

Graph based representations allow to model a wide variety of data [1, 2]. However, since graphs do not lie in euclidean spaces, definition of a graph similarity measure is not a trivial problem. Nonetheless, different approaches have been explored to define similarity or dissimilarity measures between graphs [1, 2]. A first family of approaches is based on graph theory, using for example the size of the maximum common subgraph of two graphs as their similarity measure. A second family of approaches aims to embed graphs into euclidean spaces by extracting a set of predefined features from either graph structures [4] or spectrum of adjacency matrices [5]. In particular, vectorial representations of graphs can be processed using the large set of well known machine learning and statistical pattern recognition methods defined on vectorial space. However, a drawback of this approach is that graphs encode complex objects using nodes and relationships between nodes and a large amount of information is lost when graphs are transformed in vectors. Hence, instead of defining an explicit embedding of graphs, an alternative approach consists in using graph kernels [8] which correspond to a scalar product between two implicit embedding of graphs. Any graph kernel, which can be seen as a similarity measure between graphs, can then be used in any machine learning method which access to data only through scalar products. Another interesting and widely used approach is the graph edit distance. Unlike most of the existing graph similarity measures, this approach aims to define a dissimilarity measure between two graphs directly in the graph space.

The graph edit distance corresponds to a measure of the distortion required to transform one graph into another. The distortion between two graphs G and G' can be encoded by an edit path defined as a sequence of operations transforming G into G'. Such a sequence may include node or edge insertions, removals and substitutions. Given a non-negative cost function c(.) associated to each edit operation, the sum of elementary operation's costs composing the edit path defines its cost. The optimal edit path is then defined as the one having the minimal cost among all possible edit paths transforming G into G'. This minimal cost corresponds then to the graph edit distance between G and G'. More formally, the graph edit distance is defined by the following equation:

$$d_{edit}(G, G') = \min_{(e_1,\ldots,e_k)\in\mathcal{P}(G,G')} \sum_{i=1}^{k} \mathrm{c}(e_i). \tag{1}$$

where $\mathcal{P}(G, G')$ corresponds to all possible edit paths, each edit path consisting in a sequence of edit operations $(e_1, \ldots, e_k)$. Therefore, computing graph edit distance relies on finding an optimal edit path among all possible ones. A common approach consists in traversing the space of all possible edit paths by using an heuristic such as A^* search. This approach, detailed in Section 2, allows to find an exact graph edit distance with the cost of an exponential complexity which restricts it to rather small graphs, typically composed of up to ten nodes.

In order to tackle this complexity, Riesen and Bunke have proposed in [9] a method to compute an approximation of the graph edit distance in a polynomial time. This method exploits the close relationship between node mapping and edit distance to reduce the complexity. Indeed, any mapping between the set of nodes and edges of two graphs induces an edit path which substitutes all mapped nodes and edges, and inserts or removes the non mapped nodes and edges of the two graphs. Conversely, given an edit path between two graphs such that each node and each edge is substituted only once, one can define a mapping between the substituted nodes and edges of both graphs. The heuristic of Riesen and Bunke [9] builds a mapping between the sets of nodes of two graphs using a bipartite assignment algorithm, and deduces an edit path from this mapping. The cost associated to this possibly non optimal edit path corresponds to an overestimation of the exact edit distance. Obviously, the better the assignment is, the lower the overestimation is and thus more accurate is the approximation. The optimal bipartite assignment algorithm is based on a cost function defined between the neighborhoods of each pair of nodes of the two graphs. The idea behind this heuristic being that a mapping between nodes with similar neighborhoods should induce an edit path associated to a low cost and thus close to the optimal one. However, this heuristic may work poorly when the direct neighbourhood does not allow to easily differentiate the nodes such as when considering unlabeled graphs.

Considering this, we can distinguish two approaches which aim to improve the approximation of graph edit distance: A first one consists in starting from the edit path induced by the mapping computed by the original method of Riesen and Bunke, and improve this edit path by slightly modifying it using for example genetic algorithms [11]. A second approach consists in improving the initial node mapping [3]. The approach proposed in [3] associates to each node of the graph a bag of k-walks starting from this node. The mapping cost of two nodes is then computed by an approximation of the cost of mapping their bag of walks. This approach allows to compute a better approximation than the original approach defined in [9], but this gain may be altered by a low accuracy induced both by the use of walks and an approximation of mapping costs. Therefore, in order to compute a more accurate approximation of graph edit distance, we propose to associate each node of the graph to a subgraph including all nodes within a radius of k edges, such graphs being denoted as k-subgraphs. These patterns may provide a more accurate description about the surroundings of a node than a bag of walks. In addition, we propose to evaluate mapping costs between k-subgraphs using either an exact or an approximated graph edit distance.

Our paper is structured as follows. First, in Section 2, we review the computation of exact graph edit distance using the A^* algorithm. We also discuss about the Beam heuristic which allows to compute an approximate graph edit distance by restricting the search space. Second, in Section 3, we review in details the approximation algorithm proposed by Riesen and Bunke in [9]. Then, in Section 4, we present our proposition to improve the approximation of graph edit distance by computing a mapping cost based on k-subgraphs instead of direct neighborhood or k-walks edit distance. Finally, we present some experiments in Section 5 showing the accuracy gain obtained using our approach.

2 Graph Edit Distance Computation: Exact Approach

The problem of computing the exact graph edit distance between two graphs can be formulated as a search problem inside an appropriate state space, as well as for other graph matching problems [1, 2]. Considering graph edit distance problem, the state space corresponds to the set of all complete and incomplete edit paths transforming the source graph into the target graph. This search is generally performed using A^* algorithm.

A^* is a well known search algorithm, which uses an heuristic function to conduct the search towards an optimal solution inside a search space. It is proven to be complete, i.e. it always find an optimal solution if it exists, and to be the best suited algorithm to perform an heuristic search. A^* algorithm consists in finding an optimal path starting from an initial state s_0 to a goal state s_g in a search space S. A^* begins thus by exploring the search space from s_0. Then, for each step corresponding to a state $s \in S$, the cost corresponding to the path from s_0 to s is encoded by a past-path cost function summing the cost of each previous step, while an estimation of the cost from s to s_g is provided by an heuristic function. The sum of these two functions provides an approximate cost of the path from s_0 to s_g that passes through s. So, given a current state s, it generates a set of successor states s' and puts non explored ones into a frontier set, namely the *Opening* set. Then, the state s' having the lowest cost estimation to reach the goal state s_g is extracted from the *Opening* set and is chosen as the next state. This search process ends when a goal state s_g is reached or the *Opening* set is empty. It is worth noticing that under certain conditions on heuristic function, A^* always finds an optimal solution.

Time complexity of A^* depends on the specific heuristic, but in the worst case, it is exponential with respect to the length of the shortest path. In order to reduce the complexity of A^* search, a limit on the size of the *Opening* set can be imposed. Clearly, in this way, the search algorithm does not guarantee to always find the optimal solution, but only a sub optimal one. Obviously, the probability of finding the optimal solution decreases as the limit size of *Opening* set decreases. This adaptation of A^* is called Beam Search.

A^* algorithm can be used to find an optimal edit path between two graphs, and thus to compute an exact graph edit distance. To this purpose, we have to clarify the structure of the search space and how the heuristic cost function is defined. On one hand, each state $s \in S$, $s \neq s_g$, corresponds to a partial solution, i.e. a partial edit path which transforms a subgraph H of G into a subgraph H' of G'. On the other hand, a goal state s_g corresponds to an optimal and complete edit path transforming G into G'.

The heuristic cost function encodes, for each state $s \in S$, an estimation of the cost required to reach s_g from s. In this paper, we used the heuristic function defined by Riesen and Bunke in [10]. This heuristic allows to find an optimal mapping between nodes and edges of G and G' which have not been previously mapped. This optimal assignment provides a minimal mapping cost of unprocessed sub graphs of the two graphs $G - H$ and $G' - H'$.

As described previously, we can use Beam Search in order to reduce the complexity of A^*. However, as shown in [7], using a limitation on the size of *Opening* set, we may not find an optimal edit path. Therefore, the computed graph edit distance may correspond to an overestimation of the exact graph edit distance.

3 Graph Edit Distance Computation: Approximate Approach

The graph edit distance approximation framework introduced in [9] reduces the search problem associated to exact graph edit distance computation to a Linear Sum Assignment Problem (LSAP) which can be solved in polynomial time. Considering two graphs G and G', the approach proposed by [9] consists in three steps. First, G and G' are subdivided into two sets of elements, where each element is defined as a bag of patterns encoding the local environment of a node of G or G'. Given these two sets, we can define a cost matrix $\mathbf{C}_\epsilon$ which encodes the cost of mapping two elements between the two sets. Second, we resolve the LSAP according to $\mathbf{C}_\epsilon$ using Munkres' algorithm [6]. This algorithm allows to compute an optimal assignment between the two sets associated to each graph which corresponds to a mapping of nodes of the first graph onto nodes of the second graph. Third, we can deduce an edit path from this optimal mapping by inferring node and edge edit operations. The cost associated to this edit path, which may not be optimal, corresponds to an approximation of the graph edit distance.

More formally, let us first consider an input labeled graph $G = (V, E, \mu, \nu)$, where V encodes the set of nodes, E the set of edges, μ the labeling function defined on nodes and ν the labeling function defined on edges. This graph is associated with a set of bags of patterns $B = \{B_i\}_{i=1,\dots,|V|}$. Every bag B_i is associated to a node $u_i \in V$ and characterizes the local structure of G around node u_i. The target graph $G' = (V', E', \mu', \nu')$ and its corresponding bags of structural patterns $B' = \{B'_i\}_{i=1,\dots,|V'|}$ are given analogously. We define a cost $\mathrm{c}(B_i \to B'_j)$ for the substitution of two bags of patterns. In order to define cost for inserting or removing bags of patterns, we introduce an empty bag of patterns ε. Then, costs $\mathrm{c}(B_i \to \varepsilon)$ and $\mathrm{c}(\varepsilon \to B'_j)$ encode respectively removal and insertion of a bag. Given this cost definition, a cost matrix $\mathbf{C}_\varepsilon(B, B')$ encoding costs of substitutions, insertions, and removals of bags of structural patterns is defined as:

$$\mathbf{C}_\varepsilon(B, B') = \begin{bmatrix} \mathbf{C}(B, B') & \mathbf{C}_\varepsilon(B \to \varepsilon) \\ \mathbf{C}_\varepsilon(\varepsilon \to B') & \mathbf{0} \end{bmatrix} \in [0, +\infty]^{(|V|+|V'|)\times(|V'|+|V|)}, \quad (2)$$

where $\mathbf{C}(B, B')_{i,j} = \mathrm{c}(B_i \to B'_j)$, $\mathbf{C}_\varepsilon(B \to \epsilon)_{i,j} = \mathrm{c}(B_i \to \varepsilon)$ if $i = j$ and $+\infty$ elsewhere. Similarly, $\mathbf{C}_\varepsilon(\varepsilon \to B')_{i,j} = \mathrm{c}(\varepsilon \to B'_i)$ if $i = j$ and $+\infty$ elsewhere.

Given the cost matrix $\mathbf{C}_\varepsilon(B, B')$, we can compute an optimal assignment between the sets B and B' in $O((|V| + |V'|)^3)$ time complexity thanks to the use of Munkres' algorithm [6]. This algorithm allows to compute a mapping between the two sets B and B', which may not be unique, having the lowest cost according to $\mathbf{C}_\varepsilon(B, B')$. Since each bag B_i is associated to a node u_i, the optimal assignment provides an optimal assignment between the nodes of both graphs with respect to the associated bags of patterns. That is, the optimal assignment corresponds to a mapping $\psi : V \cup \{\varepsilon\}^{|V'|} \to V' \cup \{\varepsilon\}^{|V|}$ of the nodes V of G to the nodes V' of G'. Due to the definition of the cost matrix, which allows both insertions and removals of elements, the mapping ψ is composed of different forms of node assignments: $u_i \to u'_j$, $u_i \to \varepsilon$, $\varepsilon \to u'_j$, and $\varepsilon \to \varepsilon$. The mapping ψ can be interpreted as a partial edit path between the graphs G and G' which only includes edit operations on nodes. The complete edit path is obtained by completing this partial edit path with edit operations on edges. Using simple graphs, this set of edit operations can be directly inferred from node edit operations since edit operations performed on nodes induces substitution, insertion or deletion of some edges in order to retrieve the target graph. Hence, the set of edges operations required to transform G into G' is obtained from the set of node operations induced by ψ. The cost of the complete edit path is finally returned as an approximate graph edit distance between graphs G and G'.

This approach proposed by Riesen and Bunke allows to compute an approximate edit distance in a polynomial time complexity with respect to the number of nodes. The low complexity of this approach allows then to use the graph edit distance in pattern recognition problems, and reaches good prediction accuracies [9]. However, the approach proposed by [9] associates to each node a bag of patterns defined as the node itself and its direct neighbourhood, i.e. its incident edges and its adjacent nodes. Using this kind of bags of patterns, this approach lacks of accuracy in some applications where the direct neighbourhood of graphs is not sufficiently discriminant. When considering such graphs, the mapping costs associated to each pair of nodes do not differ sufficiently and the optimal mapping depends more on the initial order of nodes than on the graph's structure. Therefore, in order to improve the accuracy of the approximation of graph edit distance, Gaüzère et al. proposed in [3] to enhance the information associated to each node by considering a bag of walks up to a length k instead of only the direct neighbourhood. This approach follows the same scheme as the one used by [9] and described at the beginning of this section, except that the set of bags of patterns associated to a node is defined as the set of walks starting from this node and having a particular length k. Considering such a bag of patterns allows to extend the amount of information associated to each node by taking into account less local structures. The bag of patterns associated to each node is then more discriminant and leads to a better approximation of the graph edit distance. However, the use of bags of walks induces some drawbacks. First, the set of computed walks suffers from the tottering phenomenon which leads to consider irrelevant patterns. These irrelevant patterns affects the mapping cost and thus the approximation of the graph edit distance. In addition, the mapping cost between two bags of walks is only approximated, which induces another loss of accuracy. Therefore, we propose in the next section to tackle these drawbacks by considering k-subgraphs associated to each node.

4 Approximate GED using K-Graphs

First, let us introduce the concept of k-subgraph (Figure 1). Given a node $u \in G$ and a radius $k \in \mathbb{N}$, we define k-subgraph(u) as the subgraph of G defined by the subset of nodes of G which can be reached from u by a path composed of maximum k edges. Given this concept, each node of the graph can be associated to a sub structure of the graph which encodes a more or less local information, depending on the value of k. However, unlike bags of walks used in [3], k-subgraphs do not suffer from tottering phenomenon.

Following the graph edit distance approximation scheme described in section 3, we propose to define the bag of patterns associated to each node $u \in V$ of a graph $G = (V, E, \mu, \nu)$ as k-subgraph(u). Then, in order to compute the optimal assignment between two sets of patterns, we propose to define matching cost c($u \rightarrow v$) as the exact graph edit distance between k-subgraph(u) and k-subgraph(v). However, as explained in previous paragraph, k-subgraph(u) and k-subgraph(v) are respectively centered around nodes u and v. In addition, the matching cost c($u \rightarrow v$) must encode the matching ability of u and v and not only the similarity of k-subgraph(u) with k-subgraph(v). Therefore, we propose

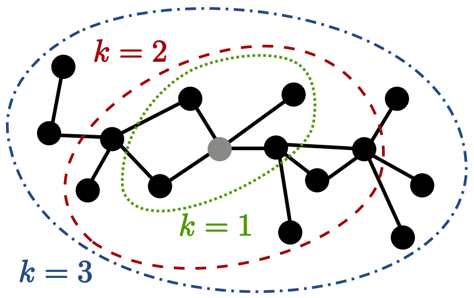

Fig. 1. Examples of k-graphs associated to a central node (in light grey).

to restrict the set of possible mappings in such a way that the two central nodes u and v are mapped together. Using such a constraint, we force a substitution operation $(u \to v)$ to be the only node edit operation performed on u and v. This restriction allows also to slightly reduce the complexity required to compute the exact graph edit distance by pruning a part of possible edit paths. Therefore, given two nodes $u \in G$ and $v \in G'$, the cost of the substitution $\mathrm{c}(u \to v)$ is defined as: $\mathrm{c}(u \to v) = \mathrm{d}_{(u,v)}(\text{k-subgraph}(u), \text{k-subgraph}(v))$ where $\mathrm{d}_{(u,v)}$ corresponds to the graph edit distance between k-subgraph(u) and k-subgraph(v) with restriction on node edit operations involving u and v. The enumeration of k-subgraphs requires to perform a depth first search from each node which is associated to a complexity in $\mathcal{O}(nd^k)$ where n is the number of nodes of the graph, d its maximum degree and k the radius associated to k-subgraph. Therefore, the computational effort is polynomial with the maximum degree of graphs and is linear with the size of the graphs. It is worth noticing that this complexity is only linear for graphs having a bounded degree. In addition, our proposition induces to compute a graph edit distance for each pair of nodes of the graphs to be compared. Considering two graphs having n nodes, these operations induce n^2 graph edit distance computations. Hopefully, these graph edit distances are only computed between graphs of a limited size for reasonable values of k, which limits computational time. However, in order to reduce this computational time, we may use the Beam search algorithm (Section 2) and limit the queue size.

5 Experiments

Following the same test protocol as in [3], we tested our heuristic on 4 graph datasets[1] encoding molecular graphs. For all these experiments and as in [3], insertion/removal costs have been arbitrarily set to 3 for both edges and nodes and substitution cost to 1 for edges and nodes, regardless of node's or edge's labels. Graphs included within the 4 datasets have different characteristics: Alkane and PAH are only composed of unlabeled graphs whereas MAO and Acyclic correspond to labeled graphs. In addition, Alkane and Acyclic correspond to acyclic

[1] These datasets are available at `http://iapr-tc15.greyc.fr/links.html`

graphs having a low number of nodes (8 to 9 nodes in average) whereas MAO and PAH correspond to larger cyclic graphs (about 20 nodes in average).

Tables 1 and 2 show a comparison of the accuracy of our proposition with two state of the art methods: the original one from Riesen [9] and an improvement of this approach using k-walks [3], both detailed in Section 3. As in [3], chosen k is the one obtaining the most accurate results. First, Table 1 shows the percentage of entries of edit distance's matrix corresponding to accuracy gain (i.e. computed edit distance is lower), accuracy loss or equivalent accuracy obtained by our approach versus the ones obtained by [9] and [3]. These percentages are displayed for different ways of computing graph edit distance between k-subgraphs: exact graph edit distance (A^*) and Beam search using a queue limit of 1000 (Beam, 1000) or 100 states (Beam, 100). On one hand, we can note that our approach provides always a better approximation of graph edit distance for 63% to 99% of molecules' pairs when compared to the approach of Riesen and 40% to 76% when compared to the approach based on k-walks. These observations are still valid even if we use Beam search algorithm in order to reduce the computational time required by our approximation. On the other hand, we observe an accuracy loss for only $<1\%$ to 15% when compared to the approach of Riesen and 8% to 41% when compared to the second approach. Note however that the comparison on PAH dataset suffers from the fact that k is limited to 2 for A^* algorithm, versus walks composed of up to 5 nodes in [3]. This limitation is induced by the high computational time required by A^* algorithm. However, we can note that using a faster algorithm which allows to consider larger k-subgraphs, we obtain a better approximation of our graph edit distance. Finally, these two first experiments shows a clear gain on the accuracy of our approximation compared to both state of the art approaches.

Same conclusions can be drawn from Table 2 which shows, for each dataset and each method, the average edit distance ($\bar{d}$) together with the average error of our approximation with respect to the exact graph edit distance and the average time required to compute graph edit distance for a pair of graphs. The exact graph edit distance, and thus the average error, is not available for MAO and

	Dataset	A^*			Beam, 1000			Beam, 100		
		Gain	**Loss**	=	**Gain**	**Loss**	=	**Gain**	**Loss**	=
[9]	**Alkane**	63%	17%	20%	60%	20%	20%	56%	23%	21%
	PAH	68%	17%	15%	81%	9%	9 %	81%	10%	9%
	MAO	99%	$<1\%$	$<1\%$	98%	2%	$<1\%$	93%	7%	$<1\%$
	Acyclic	75%	15%	10%	72%	18%	11%	68%	21%	11%
[3]	**Alkane**	63%	17%	21%	59%	19%	22%	54%	21%	26%
	PAH	40%	41%	18%	63%	23%	14%	61%	25%	14%
	MAO	76%	8%	16%	69%	16%	16%	47%	43%	10%
	Acyclic	59%	22%	18%	55%	25%	19%	51%	30%	19%

Table 1. Accuracy comparison of approach versus [3] and [9].

Method	**Alkane**			**Acyclic**			**MAO**		**PAH**	
	$\bar{d}$	$\bar{e}$	$\bar{t}$	$\bar{d}$	$\bar{e}$	$\bar{t}$	$\bar{d}$	$\bar{t}$	$\bar{d}$	$\bar{t}$
A*	15		1.29	17		6.02				
[9]	35	18	10^{-2}	35	18	10^{-3}	105	10^{-3}	138	10^{-3}
[3]	33	18	10^{-3}	31	14	10^{-2}	49	10^{-2}	120	10^{-2}
KG, A^*	26	11	2.27	28	9	0.73	44	6.16	129	2.01
KG, Beam 1000	27.3	12.6	0.46	28.6	9.9	0.13	47	5.74	113	19.39
KG, Beam 500	27.6	12.1	0.58	28.7	9.9	0.21	54	6.43	113	19.74
KG, Beam 100	28.4	12.9	0.22	29.7	10.8	0.12	60	1.84	115	4.79
KG, Beam 50	28.8	13.2	0.15	30	11.2	0.09	76	1.10	115	2.73

Table 2. Average edit distance ($\bar{d}$), average error ($\bar{e}$) and average time in seconds ($\bar{t}$) for each method and each dataset (KG = our method).

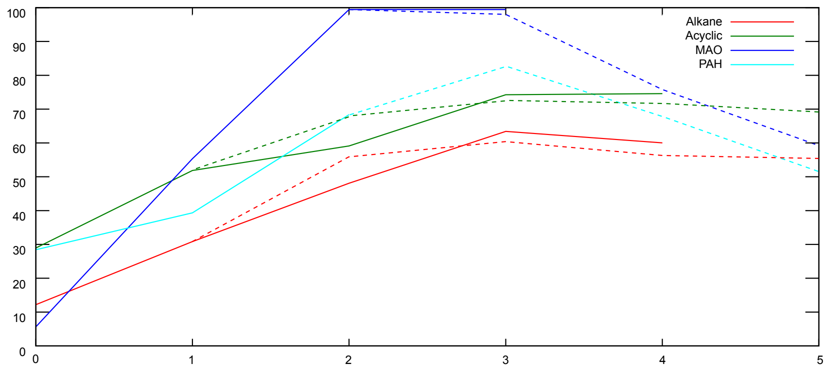

Fig. 2. Gain in accuracy compared to Riesen [9] obtained for each dataset versus the size of considered paths (k). Smashed lines represent the gain using beam search algorithm instead of A^* algorithm.

PAH datasets since it requires too much time to be computed. First, we can note that our approaches require higher computational times with respect to other approximation frameworks. This observation is coherent with the fact that we have to compute, for each pair of nodes to be matched, a graph edit distance between rather large graphs when k is equals to 3 or 4. In addition, computation times obtained for line 3 corresponds to a Matlab/`C++` implementation whereas other lines have been computed using a Java implementation [10]. The results shown in these two tables show the gain in accuracy induced by using k-subgraphs to compute a matching cost between nodes instead of using direct neighbourhood [9] or k-walks [3]. In addition, we can note that taking into account a large radius for k-subgraphs increases the accuracy of our edit distance approximations (Figure 2). Conversely to the observation stated in [3], in our framework, considering a large radius does not induce irrelevant patterns and thus the accuracy does not decrease. The decrease in gain accuracy observed for smashed lines for larger k is due to the fact that the limit on the number of states does not allow to find an optimal edit path. This may occur more often as the number of possible edit paths increases.

6 Conclusion

In this article, we have proposed a new heuristic to enhance the approximation of graph edit distance using the framework based on optimal bipartite graph matching. Our proposition aims to use less local and more discriminant sub structures, called k-subgraphs, associated to each node. Despite the high computational time induced by our proposition, our approach is still less time consuming than computing an exact graph edit distance and we obtain a better approximation accuracy than previous methods, hence showing the relevancy of considering larger and exact patterns with respect to previous propositions.

References

1. Conte, D., Foggia, P., Sansone, C., Vento, M.: Thirty years of graph matching in Pattern Recognition. International Journal of Pattern Recognition and Artificial Intelligence 18(3), 265–298 (2004)
2. Foggia, P., Percannella, G., Vento, M.: Graph Matching and Learning in Pattern Recognition on the last ten years. Journal of Pattern Recognition and Artificial Intelligence 28(1) (2014)
3. Gaüzère, B., Bougleux, S., Riesen, K., Brun, L.: Approximate Graph Edit Distance Guided by Bipartite Matching of Bags of Walks. In: Structural, Syntactic and Statistical Pattern Recognition, Lecture Notes in Computer Science, vol. 8621, pp. 73–82. Springer (2014)
4. Gibert, J., Valveny, E., Bunke, H.: Graph embedding in vector spaces by node attribute statistics. Pattern Recognition 45(9), 3072–3083 (2012)
5. Luo, B., Wilson, R.C., Hancock, E.R.: Spectral embedding of graphs. Pattern Recognition 36(10), 2213 – 2230 (2003)
6. Munkres, J.: Algorithms for the assignment and transportation problems. Journal of the Society for Industrial and Applied Mathematics 5(1), 32–38 (1957)
7. Neuhaus, M., Riesen, K., Bunke, H.: Fast suboptimal algorithms for the computation of graph edit distance. In: Structural, Syntactic and Statistical Pattern Recognition, pp. 163–172. Springer (2006)
8. Ramon, J., Gärtner, T.: Expressivity versus efficiency of graph kernels. In: First International Workshop on Mining Graphs, Trees and Sequences. pp. 65–74 (2003)
9. Riesen, K., Bunke, H.: Approximate graph edit distance computation by means of bipartite graph matching. Image and Vision Computing 27, 950–959 (2009)
10. Riesen, K., Emmenegger, S., Bunke, H.: A novel software toolkit for graph edit distance computation. In: Graph-based Representations in Pattern Recognition, LNCS, vol. 7877, pp. 142–151. Springer (2013)
11. Riesen, K., Fischer, A., Bunke, H.: Improving Approximate Graph Edit Distance Using Genetic Algorithms. In: Structural, Syntactic and Statistical Pattern Recognition, Lecture Notes in Computer Science, vol. 8621, pp. 63–72. Springer (2014)