Pattern Recognition Letters 151 (2021) 310–316

Contents lists available at ScienceDirect

Pattern Recognition Letters

journal homepage: www.elsevier.com/locate/patrec

Efficient approximate approach for graph edit distance problem

Adel Dabah[a,*], Ibrahim Chegrane[b], Saïd Yahiaoui[a]

[a] CERIST Research Center on Scientific and Technical Information, Algiers, Algeria
[b] CoBIUS lab, Department of Computer Science, University of Sherbrooke, Canada

ARTICLE INFO

Article history:
Received 12 December 2020
Revised 11 June 2021
Accepted 23 August 2021
Available online 14 September 2021

Edited by: Prof. S. Sarkar

Keywords:
Graph matching
Graph edit distance
Approximate approach
Pattern recognition,

ABSTRACT

Graph Edit Distance (GED) problem is a well-known tool used to measure the similarity/dissimilarity between two graphs. It searches for the best set of edit operations (in terms of cost) that transforms one graph into another. Due to the NP-hardness nature of the problem, the search space increases exponentially making exact approaches impossible to use for large graphs. In this context, there is a huge need for approaches that give near-optimal results in reasonable time. In this paper, we propose a tree-based approximate approach for dealing with GED problem. It operates on a search tree that models all possible solutions of the problem. Since exploring the whole tree is impractical; this approach keeps only the best k nodes at each level of the tree for further exploration. This reduces enormously the execution time without scarifying the solution quality. Experiments using small and medium size data-sets show the low deviation of our results as compared to the optimal results of a Depth First Search algorithm. Moreover, our approach show a strong scalability potential by dealing with large data-sets in low execution time.

© 2021 Elsevier B.V. All rights reserved.

1. Introduction

Graph representation is a powerful tool to model large number of real-world objects from human faces to fingerprints and biology. It allows to overcome many problems related to image representation such-as: scaling, rotation, and translation of images. As a results, they are used widely as a powerful representation of objects in pattern recognition. Thus, a pattern recognition problem becomes a problem of graph matching.

Graph Edit Distance (GED) approach is a well-known technique used to measure the similarity/dissimilarity between two graphs (objects). It was first introduced by [1]. The goal of GED is finding the best set of edit operations, in terms of cost, needed to transform one graph into another [2]. The allowed operations are insertion, deletion and substitution. Due to noise, the graph representation of identical real-world objects may not match exactly. For this reason, GED can be used at the same time to find the *Exact*, and also the *Inexact* matching, where some errors are tolerated.

GED problem is known to be very challenging due to its NP-hardness nature [3], which means that the time needed to compute the minimum distance between two graphs increases exponentially with the number of vertices. As a result, optimal approaches like Depth First Search (DFS) and A-star algorithms are impractical due to theirs prohibitive running time, especially when dealing with large graphs. To deal with such large graphs, using approximate approaches is inescapable.

In this paper, we propose an efficient approximate approach that answers the huge need for approaches that ensure tight bounds for GED problem in a low execution time. Our approach is an adaptation of a detection algorithm used in communication field, known as *K-best algorithm* which is a version of the Sphere Decoder algorithm [4].

The proposed approach operates on a search tree that models all possible edit-paths that transform one graph into another, i.e., all possible solutions of the problem. The tree is built using the branching process over vertices. This latter decomposes a problem (tree node) into several smaller sub-problems which are treated in the same way until reaching leaf nodes (solutions). The proposed approaches mimics Breadth-First Search (BFS) and explores the search tree level by level. At each level, our approach has two phases: A first phase where the approach performs the branching process over all nodes of a given level. After that, a second phase begins by evaluating all resulting nodes from the first phase; and then selects only the best k nodes, in terms of evaluation, for the next level. This process is repeated until reaching the last level where solutions exist. Thus, returning the best one in terms of edit-distance. The interesting fact about this approach is its fixed time complexity. As a fact, the complexity our approach depends on the number of partial edit-distance calculations. This latter depends on the number of selected nodes at each level, and the size

* Corresponding author.
E-mail addresses: adabah@cerist.dz (A. Dabah), Ibrahim.Chegrane@usherbrooke.ca (I. Chegrane), syahiaoui@cerist.dz (S. Yahiaoui).

https://doi.org/10.1016/j.patrec.2021.08.027
0167-8655/© 2021 Elsevier B.V. All rights reserved.

of considered graphs. The obtained results of our approach using small and medium size data-sets, under different cost-settings, show a near optimal results with a very low computational effort. Indeed, in a less than one second, our approach reaches a less than 5% deviation from optimal edit-distances obtained by the DFS approach. This latter takes, in average, one hours for each combination. Thereby, our approach achieves a good balance between the complexity and the quality of the returned solution. Our results for large data-sets, in terms of accuracy and execution time, outperform those of the Beam-search approach [5]. To our knowledge, this latter approach represents the most efficient approach in the literature. Moreover, our approach shows a good scalability potential since it deals with larger data-sets, in a relatively low execution time.

The rest of the paper is organized as follows: Section 2 gives a global view on well-known approaches for GED problem. Section 3 presents basic notations and definitions of the GED problem. Section 4 describes our proposed approximate approach and its components. Section 5 reports computational results, analysis, and discussions of our proposed approach. Finally, conclusions and perspectives are given in Section 6.

2. Related work

The NP-hard nature of the GED problem induces a huge running time for optimal approaches such-as the A-star and B&B algorithms. Despite their complexity, these approaches were largely used, but especially for small datasets. Bunke and Allermann [2] were the first to implement and adapt the A-star algorithm to solve the GED problem using the three edit operations: deletion, insertion, and substitution. Riesen et al. [6] improved the performance of the basic A-star algorithm by proposing an efficient estimated cost-function heuristic named *Bipartite heuristic*. This latter sums the cost of the best assignments of unmapped vertices and edges between the two processed graphs. Abu-Aisheh et al. [7] proposed a Depth First approach (DF-GED) to solve optimally the GED problem. The authors' goal is to reduce the amount of used memory space, as compared to the A-star algorithm.

Due to the tremendous execution time of exact approaches, there have been several attempts to lower their complexity, but at cost of losing optimality insurance. Two major approaches exist in the literature. The first one is based on a fixed-time parallel B&B approaches. Abu-Aisheh et al. [8,9] proposed a shared and distributed memory parallelization schemes of their DF-GED algorithm. The idea here is that several parallel instances of the DF-GED explore the search tree simultaneously in depth-first manner. After certain amount of time, the search is stopped and the best solution explored by all parallel instances is returned as an approximate solution for the GED problem.

The second one, known as Beam-Search, is proposed by Neuhaus et al. [5]. This approach lowers the complexity of A-star algorithm by limiting the size of its priority-queue to a given size Z. This induces an elimination of all nodes with a rank greater than Z during the search process. To our knowledge, this last approach reports the best results in the literature. However, its lack of scalability represents an issue for using it for large data-sets.

3. Problem definition

Graph Edit Distance (GED) is a well-known technique used in Graph Matching area to compute the amount of dissimilarity between two graphs. It represents the cost of the best set of edit operations needed to transform one graph into another [2]. The allowed operations are insertion, deletion and substitution, which are applied on both vertices and edges.

A graph is a structure used to model pairwise relations between objects [10]. It is denoted by $G = (V, E, \alpha, \beta)$, where:

- $V = \{v_1, v_2, \ldots, v_n\}$ a set of n vertices.
- $E = \{e_1, e_2, \ldots, e_m\}$ a set of m edges ($E \subseteq V \times V$).
- $\alpha : V \rightarrow L_V$ a labeling function on the vertices.
- $\beta : V \rightarrow L_E$ a labeling function for the edges.

  L_V and L_E can be integers, or symbolic labels.

In this paper, we consider the case of undirected graphs.

3.1. Operations of GED

In GED problem, an edit path, that transforms g_1 into g_2, consists of a set of edit operations. Each edit operation, denoted by o, performs an action on either vertices or edges, or both vertices and edges. In our work, we consider a vertex-centric approach in which edit operations are performed on vertices, and the edges are implied. The edit operations are: insertion, deletion, and substitution.

Let us consider a vertex $v_i \in V_1$ from g_1 and a vertex $u_j \in V_2$ from g_2:

1. $v_i \rightarrow u_j$: Vertex u_i is substituted by vertex u_j.
2. $v_i \rightarrow \epsilon$: Vertex u_i is deleted from g_1.
3. $\epsilon \rightarrow u_j$: $u_j \in V_2$ is inserted into g_1.

3.1.1. Implied edges operations

Since our work is based on vertex-centric approach, the edit operations on edges are implied. Based on edit operations performed on its incident vertices, an edge can be substituted, deleted, or inserted [11]. Let us consider two vertices v, v' from g_1, and two other vertices u, u' from g_2. If we perform the following two edit operations $\{v \rightarrow u\}$ and $\{v' \rightarrow u'\}$; we can have three cases of implied edge:

1. If edges $(e_1 = (v, v'))$ and $(e_2 = (u, u'))$ exist respectively in g_1 and g_2. Then, e_1 is substituted by e_2 in g_1. i.e. $(e_1 \rightarrow e_2)$.
2. If the edge $(e_1 = (v, v'))$ exists in g_1 and there is no edge between u and u' in g_2. In this case, ed_1 is deleted from g_1. i.e. $(e_1 \rightarrow \epsilon)$.
3. If there is no edge between v and v' in g_1 and the edge $(e_2 = (u, u'))$ exists in g_2. In this case, e_2 is inserted in g_1. i.e. $(\epsilon \rightarrow e_2)$.

If a vertex is deleted from g_1, all its incident edges are automatically deleted from g_1. Similarly, if a vertex is inserted in g_1, all its incident edges in g_2 are automatically inserted in g_1,if the other vertices also exist in g_1.

3.2. Cost function

In GED problem, a cost function gives a charge (cost) of each edit operation applied on vertices or edges. It represents an efficient way to integrate domain specific information about object similarity. The cost $c(o)$ of a particular edit operation o is defined with respect to the underlying label alphabets:

$c(u \rightarrow \epsilon) = \theta$, $c(\epsilon \rightarrow v) = \theta$, and $c(u \rightarrow v) = c(v \rightarrow u) = \beta$.

For unlabeled graphs, the substitution cost is generally set to zero (free of charge, i.e., $\beta = 0$), while the insertion and deletion operations are usually defined via a unit cost for both vertices and edges.

In this paper, we used three cost-settings which are presented in Section 5.

3.3. Graph edit distance

Let us consider a graph $g_1 = (V_1, E_1, \alpha_1, \beta_1)$ as the source graph, and $g_2 = (V_2, E_2, \alpha_2, \beta_2)$ as the target graph. The Graph Edit Distance between g_1 and g_2, denoted by $d(g_1, g_2)$, represents the amount of similarity/dissimilarity between the two graphs. It represents the best set of edit-operations $\{o_1, o_2, \ldots\}$ in terms of cost that transform g_1 into g_2.

In a formal way, The graph edit distance between g_1 and g_2 is defined by:

$$d(g_1, g_2) = min_{\{o_1, o_2, \ldots, o_k\} \in \gamma(g_1, g_2)} \sum_{i=1}^{k} c(o_i)$$

where $\gamma(g_1, g_2)$ denotes the set of all edit paths transforming g_1 into g_2. Each path λ contains a set of edit operations $\lambda = \{o_1, o_2, \ldots, o_{|\lambda|}\}$, and $c(o_i)$ denotes the cost of the edit operation o_i [11].

4. Proposed tree-Based approximate approach for GED problem

In the same way as a Breadth-First Search (BFS), Our approximate approach explores the search tree level by level until reaching leaf-nodes. Therefore, performing the branching and evaluation for all nodes of a given level before moving to the next one. However, instead of considering all resulted nodes from the branching process. We consider only the best K nodes in terms of evaluation for further exploration (next level), which reduces the execution time. Therefore, two phases can be identified in this approach. A first phase where we perform the branching for all nodes of a given level, and a second phase where we compute the evaluation and we select the best k nodes in terms of evaluation for the next level. Theses two phases are repeated until reaching the last level where solutions exist. This approach does not explore the whole search tree which results in a sub-optimal solution. The selected nodes at each level have a lower evaluation. Thus, they are more likely to contain promising paths.

Moreover, Fig. 1 illustrates the different phases of our approximate approach used to solve GED problem.

4.1. Phase 1: Branching

This phase performs the branching process for all nodes of a given level. It decomposes each problem (tree node) into several smaller sub-problems which are treated in the same way until reaching leaf nodes. The branching for GED problem is based on the idea of mapping vertices from the first graph g_1 to vertices of the second graph g_2. i.e. Vertex-centric approach is used. The mapping performs the following classical edit operations: substitution, deletion, and insertion. In a formal way, a search tree node *nd* is characterized by:

- A set of past edit operations known as *path* and denoted by $\lambda = \{o_1, o_2, \ldots, o_k\}$ where o_i is one of the edit operations (substitution, deletion, and insertion).
- A remaining vertices from the two graphs (g_1 and g_2) denoted by R_{V1} and R_{V2} respectively.
- A processed vertices from the two graphs (g_1 and g_2) denoted by P_{V1} and P_{V2} respectively.

In this way, the root node (initial problem) is defined as $|R_{V1}| = |V_1|$, $|R_{V2}| = |V_2|$, $P_{V1} = \emptyset$, and $P_{V2} = \emptyset$.

The branching on a search tree node *nd* generates $|R_{V2}|$ new successors by performing the substitution of a vertex $u \in R_{V1}$ with all vertices in R_{V2}. In other words, each successor node nd_j (where $j \in [1.|R_{V2}|])$ represents the mapping of a vertex $u \in R_{V1}$ by a vertex $u' \in R_{V2}$ is defined as follows:

$$R_{V1}(nd_j) = R_{V1}(nd) \setminus \{u\}$$

$$R_{V2}(nd_j) = R_{V2}(nd) \setminus \{u'\}$$

$$P_{V1}(nd_j) = P_{V1}(nd) \cup \{u\}$$

$$P_{V2}(nd_j) = P_{V2}(nd) \cup \{u'\}$$

$$\lambda(nd_j) = \lambda(nd) \cup \{u \rightarrow u'\}.$$

In addition to the successors generated above, we add a new successor that represents the deletion of the vertex $u \in R_{V1}$ as follow:

$$R_{V1}(nd_j) = R_{V1}(nd) \setminus \{u\}$$

$$R_{V2}(nd_j) = R_{V2}(nd)$$

$$P_{V1}(nd_j) = P_{V1}(nd) \cup \{u\}$$

$$R_{V2}(nd_j) = P_{V2}(nd)$$

$$\lambda(nd_j) = \lambda(nd) \cup \{u \rightarrow \epsilon\}.$$

This process is repeated for all the nodes of a given level. The last level, where solutions (leaf nodes) exist, is reached when $R_{V1} = \emptyset$. If the set of remaining vertices in g_2 is not empty ($R_{V2} \neq \emptyset$); we insert them all in R_{V1}.

The number of nodes generated at the end of this phase depends on the number of nodes in the level and the size of remaining vertices in g_2. Thus, if a level contains k nodes, the number of successors generated is $k * (|R_{V2}| + 1)$.

4.2. Phase 2: Evaluation and selection

4.2.1. Evaluation (Ee)

After a first phase where the branching process is performed, the evaluation process calculates the partial edit distance for each resulted node. The evaluation E, in our case, has a linear complexity ($O(n)$) that adds up the cost of all edit-operations performed on a search tree node. More precisely, the evaluation of a search tree node (nd) equals: $E(nd) = Cost(\lambda(nd)) = \sum_{i=1}^{|\lambda|} c(o_i)$.

For example, for a search tree node *nd* with the following path:

$$\lambda = \{(v_1\{B\} \rightarrow u_1\{B\}), \quad (1)$$

$$(v_2\{A\} \rightarrow u_2\{A\}), \quad (2)$$

$$\boldsymbol{((v_2, v_1) \rightarrow \epsilon)}, \quad (3)$$

$$\boldsymbol{(v_3\{A\} \rightarrow u_3\{B\})}, \quad (4)$$

$$((v_3, v_2) \rightarrow (u_3, u_2)), \quad (5)$$

$$(v_4\{A\} \rightarrow u_4\{A\}), \quad (6)$$

$$((v_4, v_1) \rightarrow (u_4, u_1)), \quad (7)$$

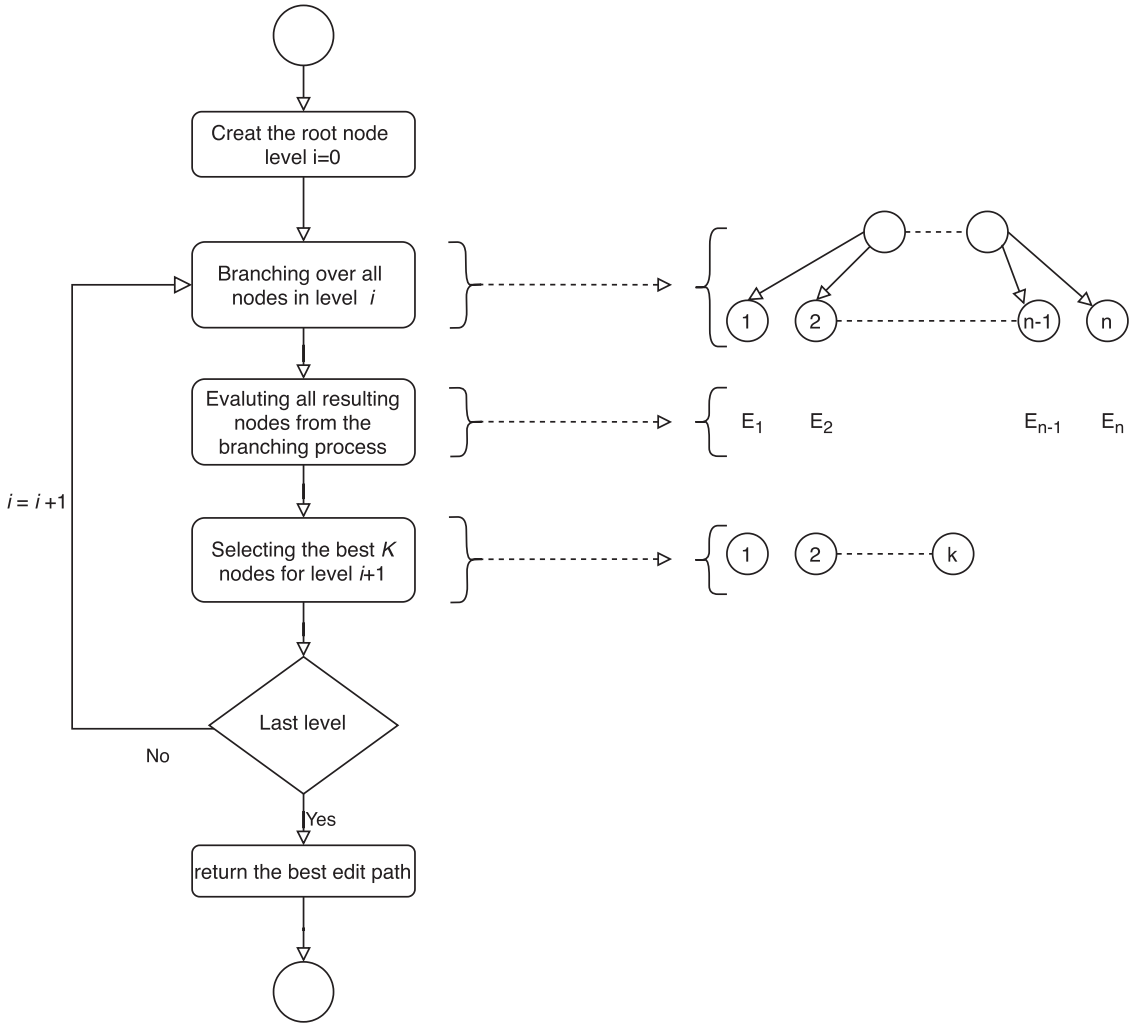

Fig. 1. general scheme of the search tree related to the GED problem.

Table 1
Cost Setting for edit operations on vertices and edges .

Cost-Settings	Edit-operations	Cost for Vertices	Cost for Edges
Setting 1 (ST1)	Substitution	2	1
	Deletion	4	1
	Insertion	4	1
Setting 2 (ST2)	Substitution	2	1
	Deletion	4	2
	Insertion	4	2
Setting 3 (ST3)	Substitution	6	3
	Deletion	2	1
	Insertion	2	1

$$((v_4, v_2) \rightarrow \epsilon), \tag{8}$$

$$((v_4, v_3) \rightarrow (u_4, u_3))\} \tag{9}$$

Its evaluation, using cost-setting one in Table 1, is equal to: $E(nd) = 0+0+1+2+0+0+0+1+0$. i.e. $E(nd) = 4$. Note that the GED between a graph and itself is equal to zero. Thus, we apply the substitution cost only when the labels are different as in O_4.

One of the best motivations of this approach is its fixed complexity. Indeed, the complexity of our approach depends mainly on the number of times we compute the evaluation algorithm. i.e., Partial Edit Distance (PED). The factors involved in the complexity are: (1) number of selected nodes (K) at each level. (2) the size of graphs which induces both the number of levels and the number of generated nodes. (3) the complexity of the evaluation algorithm itself, which has a linear complexity is in our case. If we consider two graphs with n vertices ($|V_1| = |V_2| = n$). The whole number of generated nodes (i.e. PED calculation) by this approach is: $k * (n^2 + 3n)/2$.

Moreover, the complexity can be further reduced if we use the evaluation of a node (nd) to evaluate its successors nd_j. Since $\lambda(nd_j) = \lambda(nd) \cup \{o_1, \ldots, o_m\}$, $E(nd_j) = E(nd) + \sum_{i=1}^{m} c(o_i)$, where $m >= 1$.

Even thought it is not covered in the scope of this paper, adding a heuristic to estimated the remaining cost can have a positive influence on the accuracy. However, it may increase significantly the execution time.

4.2.2. Selection

This step aims to select nodes that are more likely to contain promising paths. After the evaluation of all resulted nodes from the first phase. We select (keep), in this step, the K best nodes according to their evaluations. In practice, this step starts by sorting the resulted nodes in ascending order according to their evaluation, and then, remove all the nodes with a rank greater than k.

This process can induce a sorting overload when the number of selected node is very high, in particular for real-time use-cases.

The value of parameter K affects both the execution time and the accuracy in terms of edit distance. On one hand, a large number of kept selected nodes allows the approach to converge to the optimal edit distance. However, the execution time of the approach becomes very significant. On the other hand, a low number of selected nodes allows a fast execution time, at the expense of edit-distance accuracy. To sum-up, the value of parameter K must be selected according to the application (use-case) and accuracy that we want to achieve.

At the end of this step, if the last level of the tree is reached, the path with the best evaluation is returned as an approximate solution for the GED problem. Otherwise, the best k nodes are used as an input for our first phase.

To increase the accuracy of this approach, a dynamic value of parameter K seems like a smart choice. i.e. A large value in early stages of the tree will be a very wise choice for the following reasons: (1) The gap in nodes' evaluation is very small in early stage of the tree. (2) the number of successors, for each node, decreases when moving from one level to the next one. As a fact, a GED problem between g_1 and g_2 ($GED(g_1, g_2)$) induces a search tree with a fixed number of levels, which is equal to $|V_1|$. However, the number of successors varies from one level to another. Indeed, the number of successors depends on the size of unprocessed vertices in g_2 ($|R_{V2}|$). Therefore, we have $|V_2|+1$ successors in the first level vs. 1 successors in last stages of the tree. This is what motivates the choice of increasing the number of selected nodes in early stages of the tree. However, the influence of this technique will not be covered in the scoop of this paper.

5. Performance evaluation

In this section we investigate the ability of our approximate approach in solving the GED problem using well-known literature data-sets and cost-settings. We begin by showing the impact of increasing the number of selected nodes (parameter k) on both the returned edit distance and time needed to find it. After that, we will validate our approach by computing deviation of our results from the optimal results given by a DFS algorithm. The last part of this section shows the performance of our approach for large data-sets with up to hundred nodes.

Computing Platform: We performed our experiments using a node from IBNBADIS cluster located at CERIST research center. This node contains two Intel(R) Xeon(R) processors (E5-2650) with 8 CPU-cores and 2.00 GHz speed for each. It also has 32 Gb of memory space running under Linux operating system. Our code is written in JAVA 1.8.

Datasets: We performed our experiments using the most used literature datasets which are : MAO, Alkane, Acyclic, and GREC.

- MAO, Alkane, and Acyclic are part of the GREYC's Chemistry databases of molecules. Theses datasets can be obtained using the following link: https://brunl01.users.greyc.fr/CHEMISTRY/.
- GREC is a subset of the IAM graph database repository proposed by [12]. This dataset can be obtained using the following link: http://www.fki.inf.unibe.ch/databases/iam-graph-database.

Most datasets are in Graph exchange Language (GXL) format[1]

Cost Function: The cost function represents the weighting of each edit operation (substitution, deletion, or insertion) on both vertices and edges. To study the impact and the robustness of our approach, three cost setting have been used in our experiments.

[1] GXL is defined as an XML sub-language, which offers support for exchanging instance graphs together with their appropriate schema information in a uniform format (http://www.gupro.de/GXL/).

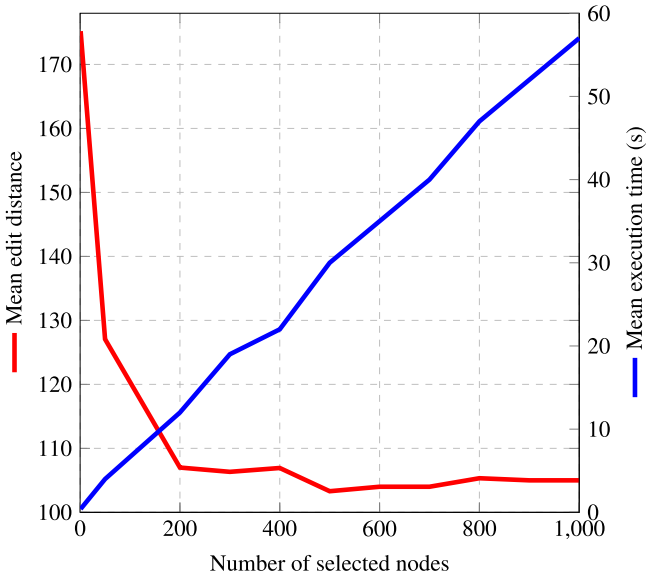

Fig. 2. Evolution of mean edit distance and execution time for MUTA-L data-set when increasing number of selected nodes using cost setting one.

These settings were first proposed in the graph matching competition organized in 2016 (https://gdc2016.greyc.fr/#ged). As described in Table 1, each setting favorites either substitution (Sub) or deletion/insertion (Del/Ins) edit operations.

5.1. Impact of increasing the number of selected nodes

In order to study the impact of increasing the number of selected nodes, we selected five large graphs from Muta data-sets denoted by MUTA-L; where the size of the graphs is between 104 and 109 vertices. Fig. 2 shows the variation of the mean edit distance and mean execution time of our approximate approach for solving 15 graph-combinations of MUTA-L subset using cost setting one.

The first point that can be highlighted in Fig. 2 is the scalability potential of our approach to deal with large graphs in reasonable time. Indeed, our approach can return an acceptable edit-path for large graphs with relatively a small number of explored nodes. i.e. Partial edit-distance calculations. For Muta-L subset which has 106 vertices in average, our approach with one selected node (k=1) explores 106 nodes (one at each level) and performs around 5777 PED calculations. This is relatively a very small number as compared to the prohibitive number of explored nodes in exact approaches. By increasing the number of selected nodes in the tree levels, the precision in terms of edit distance increases as well as the execution time. In more details, we can notice that the mean edit distance returned for the 15 combinations decreases rapidly (precision increases) when increasing the number of selected nodes before reaching stabilization starting from 500 selected nodes. For this reason, the incoming experiments for large data-sets are performed using 500 nodes. For small data-sets a much larger number of selected nodes can be used since the execution time is very low for such data-sets.

For the complexity of our approach, we can see from the same figure that the execution time increases in a linear fashion when increasing the number of selected nodes. This represents one of the best motivation to choose this approach as compared to other literature approaches. Indeed, the linear increase is the result of the fixed complexity of our approach. This later depends on the graph sizes and the number of selected nodes at each level of the

Table 2
Deviation of our approximate approach from the optimal results of a DFS approach.

datasets	Size	NB-comb.	K-best$_{5000}$				DFS
			ST 1 Dev %	ST 2 Dev %	ST 3 Dev %	Time (10^{-3}S)	Time (10^{-3}S)
Acyclic	10	49,962	3.08	4.85	4.81	131	4954
Mao 200	5–20	6837	1.63	2.72	2.62	850	44,711
Alakane	10	33,528	1.61	2.37	2.37	145	418
Muta	10	138	0.00	0.38	1.62	262	795
Grec	5	138	0.00	0.00	1.51	9	10
	10	138	0.00	0.00	4.41	330	2970
	15	138	3.59	8.24	11.14	894	24,570,873
Average			1.41	2.65	4.07	375	3,517,819

search tree. i.e. doubling the number of selected nodes will certainly double the execution time.

5.2. Performance evaluation for small and medium size data-sets

This sub-section aims to measure the relative error (edit distance deviation) and execution time of our approach as compared to the optimal results of a DFS algorithm. In other words, how close the k-best results are to the optimal results. Table 2 summarizes the obtained results.

For Mao dataset, we created a sub-dataset (Mao$_{200}$) that contains the first 200 graphs of Moa dataset. For all the other datasets (Acyclic, Alkane, Grec 5, Grec 10, Grec 15, and MUTA 10) we tested all possible combinations in each dataset.

For each dataset, Columns Size and NB-conb. show respectively graph-size and the number of graph-combinations tested. Moreover, For each setting, Column Dev %, reports the average deviation of the k-best results for all tested graph-combinations. The deviation is defined by the percentage of error from the optimal edit-distance obtained by a DFS algorithm :

$$Dev = \frac{Kbest - DFS}{DFS} * 100$$

Finally, Time Columns report the average time needed by each approach for one graph-combination. To speedup the time needed by the DFS algorithm, especially for medium size data-sets, we used the k-best results as upper-bounds.

We can observe from Table 2 that the k-best approach has in average 2.66% error rate from optimal results. This means that the k-best results are very close to the optimal results in terms of edit distance. But the most important aspect of k-best algorithm is its low execution time. As a fact, the k-best approach takes in average only 375 milliseconds, whereas the DFS algorithm takes one hour execution time in average. Thus a huge difference in execution time, but a near optimal results in terms of edit distance. The k-best low execution time is explained by the fixed number of explored paths. The DFS algorithm explores the entire search space, which may contain up to several billions paths for medium size data-sets. This engenders a prohibit execution time. The k-best low error percentage is the results of the selection process. This latter selects at each level of the tree, the best nodes according to their evaluation since they have more chance to reach good solutions, i.e., reaching near optimal path. We can also see that *setting 3* influences the accuracy of k-best approach since the error-deviation increases from 1.41% for setting 1 to 4% for *setting 3*. For this reason, it is important to test several cost-configurations to show the real accuracy of an approach. Indeed, *setting 3* favors deletion/insertion instead of substitution. Thus, the optimal edit-path may have a lot of deletion/insertion operations, especially for implied edges. The k-best approach can miss the selection of the node that contains the optimal path if its partial evaluation is slightly bigger than the other nodes. Moreover, using sophisticated selection methods can further improve the accuracy of this approach.

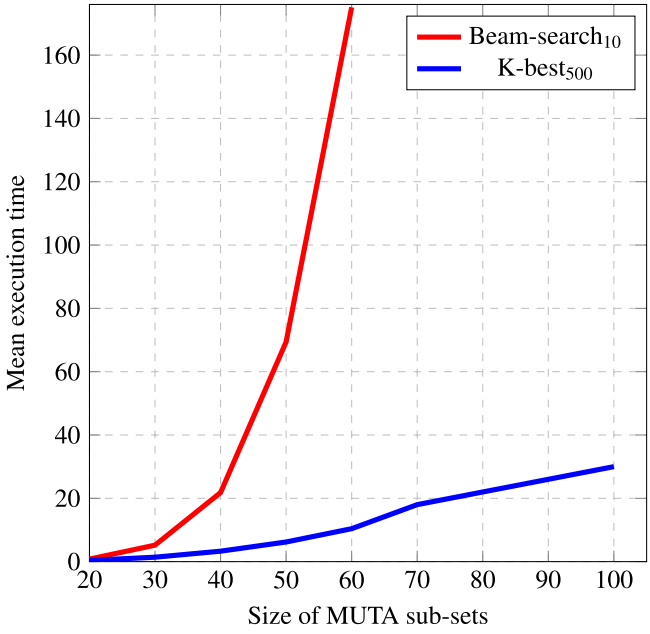

Fig. 3. Evolution of mean execution time when increasing the size of MUTA data-set using cost setting one.

The other interesting observation is the ability of the k-best approach to achieve 0% deviation in Grec 5, Grec 10, and Muta 10 data-sets. i.e. For more than 1200 combinations, our approach reached the optimal edit distances obtained by a DFS algorithms in almost all combinations.

When increasing the size of graphs in Grec dataset, we can see the linear behaviour of k-best approach as explained later in Fig. 3. On the opposite, the execution time of the DFS approach with very tight upper-bounds increases in an exponential way. It takes in average almost seven hours for each combination in Grec-15 data-set. This is huge time as compared to k-best approach which takes only 894 milliseconds in average for each combination. Thus, demonstrating a strong scalability potential.

5.3. Performance evaluation for large graphs

Since optimal approaches can not deal with large data-sets in a reasonable time, we compared the results of our approach with the results of the Beam-search algorithm [5]. To the best of our knowledge, this algorithm represents the most efficient approximate approach for GED problem in the literature. It consists in its basic version to limit the size of the A-star priority queue to a certain size. The larger the size of beam-search queue, the more accurate

Table 3
K-best results versus Beam-search approach for large data-sets using the three cost-settings in Table 1.

Dataset	Size	NB	Setting 1		Setting 2		Setting 3		Time (S)	
			$kbest_{500}$	BS_{10}	$kbest_{500}$	BS_{10}	$kbest_{500}$	BS_{10}	$kbest_{500}$	BS_{10}
GREC	20	55	31.2	37	38.9	53.45	53.6	55.4	00.4	01.7
	Mix	55	35.7	36.9	43.2	47.0	33.1	34.1	00.1	00.4
CMU	30	55	153.3	133.5	147.2	188.3	192.3	192.4	02.2	239
MUTA	20	55	21.9	23.6	30.1	39.6	32.6	37.6	00.4	01.1
	30	55	31.0	36.8	47.2	65.4	41.6	50.6	01.5	06.4
	40	55	51.8	49.9	81.3	87.1	61.4	69.6	03.5	27.4
	50	55	61.2	68.8	105.0	121.4	65.6	83.5	06.2	49.6
	60	55	70.8	79.5	119.8	160.3	77.23	99.2	11.1	250
	70	55	88.3	-	152.7	-	96.0	-	18.3	-
	Mix	55	131.7	140.8	171.7	192.7	99.6	115.6	05.6	297
Average			65.4	67.4	93.7	106.1	75.3	82	3.45	97

1- $Kbest_{500}$: Mean edit distance of kbest algorithm with 500 selected nodes at each level.
2- BS_{10}: Mean edit distance of the Beam-search algorithm with a priory queue of size ten.

it becomes. However, the time of this approach will also increase drastically. To ensure a balance between accuracy and execution time (less than 600 seconds), we fixed the size of the beam-search queue to ten ($Beam\text{-}search_{10}$).

Table 3 summarizes the obtained results, Its first three columns shows the dataset' name, size, and number of tested combinations. For each setting and method, Columns *d* and *t(s)* report respectively the mean edit distance and execution time obtained for the 55 combinations tested for each dataset.

Cost-settings two and three in Table 3 shows that the results of our approach dominate all beam-search results for all data-sets. Indeed, our approach is in average 11% more accurate and up to 30 times faster as compared to beam-search approach. Moreover, cost-setting one seems to have an impact on the results. In fact, Beam-search approach outperforms our approach in two sub-sets over ten. i.e. Beam-search is better in accuracy in CMU-30 and MUTA-40 subsets, but around 7 times slower. In average for cost-setting one, our approach is 3% more accurate and 7 times faster.

Our approach shows a great scalability potential as compared to the Beam-search approach. This later was not able to report results in 600 seconds for MUTA-70 subset, which indicates a scalability problem for this later approach. There are two factors that increase the execution time for our approach. The first factor is the number of selected nodes (see Fig. 2) which increases the execution time in a linear fashion. The second factor that also increases the complexity is the size of the graphs. Fig. 3 reports, for both approaches, the execution time evolution when increasing the size of data-sets. We can see from Fig. 3, that the time needed by our approach increases in nearly linear way which confirms the scalability potential of our method. For the beam-search method, the execution time seems increasing in an exponential way which makes it not scalable and hard to use for larger data-seats.

6. Conclusion and perspectives

In this paper we proposed an approximate approach for Inexact-Graph Edit Distance problem (GED). This problem consists to measure the amount of similarity/dissimilarity between two graphs where some error is tolerated. Its importance comes from its various applications especially in areas related to pattern Recognition. However, solving this problem optimally is impractical due to its NP-harness nature, especially when dealing with large data-sets. In order to overcome this drawback, we proposed an efficient approximate tree-based approach. This latter mimics the Bread-first-search (BFS) approach and explores the search tree level by level until reaching the last one where solutions exist. However, instead of considering all nodes of a given level (BFS), our approach selects and keeps only some of the nodes and remove the others. Thus, allowing only some of the nodes to pass to the next level. The kept nodes are chosen according to their evaluation score, since they have more chance to contain promising paths. The interesting fact of this approach is it fixed complexity which can be easily computed using the number of kept nodes at each level and the size of considered graphs. Experiments on well-known data-sets showed the high accuracy of our approach at a very low computational cost. Indeed, the results indicated the low error-deviation of our results from the optimal results of a DFS algorithm. Moreover, our approach showed a good scalability potential by reporting tight upper-bounds for large data-sets. In future works, we will investigate the use dynamic selection process and the use of estimated-cost heuristics to enhance the accuracy of this approach.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] A. Sanfeliu, K.-S. Fu, A distance measure between attributed relational graphs for pattern recognition, IEEE Trans Syst Man Cybern (3) (1983) 353–362.
[2] H. Bunke, G. Allermann, Inexact graph matching for structural pattern recognition, Pattern Recognit Lett 1 (4) (1983) 245–253.
[3] Z. Zeng, A.K. Tung, J. Wang, J. Feng, L. Zhou, Comparing stars: on approximating graph edit distance, Proceedings of the VLDB Endowment 2 (1) (2009) 25–36.
[4] K.-w. Wong, C.-y. Tsui, R.-K. Cheng, W.-h. Mow, A vlsi architecture of a k-best lattice decoding algorithm for mimo channels, in: 2002 IEEE International Symposium on Circuits and Systems. Proceedings (Cat. No. 02CH37353), volume 3, IEEE, 2002. III–III
[5] M. Neuhaus, K. Riesen, H. Bunke, Fast suboptimal algorithms for the computation of graph edit distance, in: Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR), Springer, 2006, pp. 163–172.
[6] K. Riesen, S. Fankhauser, H. Bunke, Speeding up graph edit distance computation with a bipartite heuristic, MLG, 2007.
[7] Z. Abu-Aisheh, R. Raveaux, J.-Y. Ramel, P. Martineau, An exact graph edit distance algorithm for solving pattern recognition problems, in: 4th International Conference on Pattern Recognition Applications and Methods 2015, 2015.
[8] Z. Abu-Aisheh, R. Raveaux, J.-Y. Ramel, P. Martineau, A parallel graph edit distance algorithm, Expert Syst Appl 94 (2018) 41–57.
[9] Z. Abu-Aisheh, R. Raveaux, J.-Y. Ramel, P. Martineau, A distributed algorithm for graph edit distance, DBKDA 2016 (2016) 76.
[10] D.B. West, et al., Introduction to graph theory, 2, Prentice hall Upper Saddle River, 2001.
[11] K. Riesen, H. Bunke, Approximate graph edit distance computation by means of bipartite graph matching, Image Vis Comput 27 (7) (2009) 950–959.
[12] K. Riesen, H. Bunke, Iam graph database repository for graph based pattern recognition and machine learning, Structural, Syntactic, and Statistical Pattern Recognition (2008) 287–297.