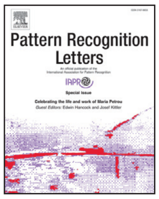
Pattern Recognition Letters 100 (2017) 96–103

Contents lists available at ScienceDirect

Pattern Recognition Letters

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



Graph edit distance contest: Results and future challenges

Zeina Abu-Aisheh[a,*], Benoit Gaüzere[b], Sébastien Bougleux[c], Jean-Yves Ramel[a], Luc Brun[c], Romain Raveaux[a], Pierre Héroux[b], Sébastien Adam[b]

[a] *Université François Rabelais de Tours, 64, avenue Jean Portalis, Tours, 37200, France*
[b] *Normandie Univ, UNIROUEN, UNIHAVRE, INSA Rouen, LITIS, Rouen, 76000, France*
[c] *Normandie Univ, UNICAEN, ENSICAEN, CNRS, GREYC UMR 6072, 6 Bd Maréchal Juin, Caen, 14050, France*

A R T I C L E I N F O

Article history:
Received 8 April 2017
Available online 9 October 2017

MSC:
41A05
41A10
65D05
65D17

Keywords:
Graph edit distance
Pattern Recognition
Binary linear programming
Quadratic assignment
Branch-and-bound

A B S T R A C T

Graph Distance Contest (GDC) was organized in the context of ICPR 2016. Its main challenge was to inspect and report performances and effectiveness of exact and approximate graph edit distance methods by comparison with a ground truth. This paper presents the context of this competition, the metrics and datasets used for evaluation, and the results obtained by the eight submitted methods. Results are analyzed and discussed in terms of computation time and accuracy. We also highlight the future challenges in graph edit distance regarding both future methods and evaluation metrics. The contest was supported by the Technical Committee on Graph-Based Representations in Pattern Recognition (TC-15) of the International Association of Pattern Recognition (IAPR).

1. Introduction

Computing a similarity or a dissimilarity measure between graphs is a major challenge in pattern recognition. One of the most well-known and used approaches to compute a distance between two graphs is the Graph Edit Distance (GED). Computing the GED consists in finding a sequence of graph edit operations (insertions, deletions and substitutions of vertices and edges) which transforms a graph into another with a minimal cost. However, computing the GED is NP-hard. Therefore, in the last four decades, several approaches were proposed to compute approximations in polynomial time [22].

This paper reports the results of the Graph Distance Contest (GDC) which was organized in the context of ICPR 2016. The aim of the contest was to inspect performance and effectiveness of recent methods which compute an exact or an approximate GED. The quality of the output distances as well as the execution times of the methods were used as keys for the inspection. Seven datasets were integrated, each of them being composed of several types of graphs with symbolic or numerical attributes attached to vertices and edges.

GDC was open to any method which computes a sequence of edit operations transforming a graph into another one. All the participants were required to download the datasets to prepare the submission of their programs. All the programs were executed by the organizers on the same computer. Two constraints were put in the contest. First, each submitted method could not exceed 30 s per graph comparison. Second, concerning parallel methods, the number of threads was limited to 4.

This paper is organized as follows: Section 2 describes the methods submitted to GDC. Then, Section 3 specifies the protocol and the datasets used for this contest. Obtained results are presented and discussed in Section 4. Note that a complementary and exhaustive presentation of the results is provided on GDC website http://gdc2016.greyc.fr. Last but not least, Section 5 highlights the bottlenecks of both tested methods and GED performance evaluation metrics, and proposes some possible tracks to go beyond these bottlenecks.

2. Inspected methods

Eight methods proposed by three different research groups were submitted. The beam search algorithm of Neuhaus et al. [21] was also added to the list of inspected methods. All these methods can be globally divided into three categories.

* Corresponding author.
E-mail address: zeina.abu-aisheh@univ-tours.fr (Z. Abu-Aisheh).

https://doi.org/10.1016/j.patrec.2017.10.007
0167-8655/© 2017 Elsevier B.V. All rights reserved.

GED as linear or quadratic assignment problems. The GED can be reformulated as a Quadratic Assignment Problem (QAP) when the set of vertices of both graphs are extended enough by null vertices to represent removal and insertions operations [4,8,22]. Since QAP are NP-hard in general, many approximation algorithms have been developed. In the GED, the notion of bipartite GED has been introduced in [24]. Based on the extended representation of the graphs, the quadratic problem is replaced by a Linear Sum Assignment Problem (LSAP) of their vertices, so that the cost of assigning two vertices is defined as the GED between the star subgraphs centered at these vertices. Richer graph structures were then explored to capture less local dissimilarities [7,10,28]. Given a $(n+m)\times(m+n)$ extended cost matrix constructed from these dissimilarities, the LSAP is solved in these works by Kuhn–Munkres version of the Hungarian algorithm [14,20] in $\mathcal{O}((n+m)^3)$ worst-case time complexity, where n and m denote the order of the graphs. When costs fulfill triangular inequality, there is no removal $(n \le m)$ or no insertion $(n \ge m)$, so the size of the LSAP can be reduced and solved in $\mathcal{O}(\max\{n, m\}^3)$ time complexity with the same Hungarian algorithm [26,27]. The LSAP is also equivalent to a binary linear program (BLP) with $(n+1)(m+1)$ variables, which can be solved in $\mathcal{O}(\min(n, m)^2 \max(n, m))$ by an adaption of Lawler version [15] of the Hungarian algorithm proposed in [5] to this BLP. We denote by *LSAPE*, the bipartite GED computed using this last algorithm on a cost matrix defined from random walks of length 3 initiated on each node [10]. *LSAPE* was one of the inspected methods in GDC.

Better approximations of the solution to QAP are obtained by iterative optimization methods based on relaxation, linear approximation and gradient descent [16,19], as experimented in [4] for the GED with the extended graphs. In particular, the integer projected fixed point (IPFP) algorithm proposed in [16] provides an elegant extension of the bipartite GED by refining the initial solution provided by the LSAP. The algorithm iterates a projection to the closest binary solution and a line search to find the next relaxed continuous solution, until a fixed point is reached. Each projection step consists in solving a LSAP, representing a 1st-order approximation of the GED. In the contest, IPFP algorithm was tested for approximating the binary quadratic program reformulation of the GED as proposed in [6]. This reformulation, named *QAPE* in this paper, considers $(n+1)(m+1)$ variables and IPFP refines the solution obtained by *LSAPE*. At each iteration, the projection becomes a BLP, which is also computed in $\mathcal{O}(\min(n, m)^2 \max(n, m))$ by the Hungarian algorithm proposed in [5]. Note that due to a limitation of *LSAPE*, *QAPE* is restricted to integer cost matrices.

Binary linear programming based approaches. A second family of algorithms consists in using BLP for computing the GED.

In [13], a BLP formulation of the GED has been proposed. This approach searches for the permutation matrix which minimizes the cost of transforming one graph into another graph. The criterion to be minimized takes into account costs for matching vertices, but the formulation does not process graphs whose edges are attributed.

Recently, BLP algorithms have been proposed to compute the GED between attributed graphs (on both vertices and edges). These algorithms were implemented using CPLEX which is one of the best mathematical programming solvers. *F2* in [17], is an exact GED approach, which is an extension of an earlier work proposed by the authors in [18]. *F2* was among the methods that participated in GDC. In this model, two sets of binary variables were associated to vertex-to-vertex matching and edge-to-edge matching, respectively. Two types of constraints were introduced to represent the GED problem. The mapping constraints ensure that each vertex of G_1 is either mapped to exactly one vertex of G_2 or deleted from G_1 and that each vertex of G_2 is either mapped to exactly one vertex of G_1 or inserted in G_1. Second, the graph topology in the mapping of vertices and edges was preserved thanks to topological constraints. A cost is associated to each mapping. The GED is then the minimum sum of mapping costs among the feasible solutions that respect the constraints of the BLP formulation.

In GDC, a parallel extension of *F2* was put forward. This algorithm, named *F24Threads*, is the same as *F2* except that CPLEX was set to run in a parallel manner with 4 threads.

The continuous relaxation of an Integer Linear Program (ILP) is a Linear Program (LP) where the constraints are unmodified but the variables are continuous. A lower bound of *F2*, named *F2LP*, was proposed in [17]. The time complexity could be reached in $O(k^{3.5})$ with the interior point method where k is the number of variables in the model. In GDC, an upper bound was derived from *F2LP* by rounding the continuous variables to the nearest integer. The obtained solution may not be a feasible solution. A feasibility pump heuristic [11] is performed to generate a feasible solution. The basic feasibility pump procedure defines a (linear) distance function $\Delta(x, \tilde{x})$ between two solutions x and $\tilde{x}$. The feasibility pump procedure solves an integer linear program with an auxiliary objective function $\Delta(x, \tilde{x})$ where $\tilde{x}$ is a nearest-integer rounding from the optimal solution of the continuous relaxation. The feasibility pump procedure tries to generate a relaxed optimum which lies as close as possible to Ìâ$\tilde{x}$. This algorithm, also called *F2LP*, was part of GDC.

Branch-and-bound based approaches. The last family that participated in GDC is the branch-and-bound one. The A^*-based algorithm is considered as a foundation work for solving the GED [25]. The computations are achieved by means of an ordered tree that is constructed dynamically at run time by iteratively creating successor vertices. Only leaf vertices correspond to complete matching operations. To overcome the memory problem of A^*, a depth-first algorithm, named *DF*, was proposed in [2]. *DF*, which also among the methods in GDC, contains two main steps: The preprocessing and branch-and-bound steps. In the preprocessing step, a first upper bound is calculated using the bipartite graph matching algorithm [24]. Moreover, the vertices and edges cost matrices are constructed to speed up the algorithm by getting rid of re-calculating the assigned costs when matching the vertices and edges of the two compared graphs. Once the preprocessing step finishes, the exploration of the search space starts in a depth-first way.

The search tree is pruned thanks to a heuristic which estimates the future cost by substituting each of the n vertices of G_1 with any of the m vertices of G_2. To obtain a lower bound of the exact edit cost, the costs of the $min(n, m)$ least expensive vertex substitutions are accumulated. To get rid of solving the minimization problem, the substitution costs are set to zero. Thus, any of the selected substitutions is always cheaper than a deletion or an insertion operation. However, the costs of $max(0, n-m)$ vertex deletions and $max(0, m-n)$ vertex insertions are accumulated. The unprocessed edges of both graphs are handled independently from the vertices.

In GDC, an upper bound, called *DFUB*, was derived from *DF*. *DFUB* has the two main steps of *DF*. However, the branch-and-bound step differs from *DF*. When a first complete solution is achieved, the algorithm terminates and outputs this solution as a final one.

A parallel version of *DF*, named *PDFS* in the contest, was proposed in [3]. A best-first strategy is performed before starting to decompose the search tree into sub-trees. Load balancing occurs when a thread finishes all its assigned problems (i.e., partial matching). *PDFS* terminates when all threads finish the exploration of their assigned problems. Each thread runs the depth-first algorithm on a part of the problem to explore the solution space in parallel and thus to discard misleading partial solutions.

A modification of A^*, called beam search (*BS*), was proposed in [21]. The purpose of *BS*, is to prune the search tree while searching

Table 1
Characteristics of symbolic and numeric datasets.

Datasets	Reference	# graphs	avg. (max) #nodes
Alkane	[12]	150	8.9 (10)
Acyclic		185	8.2 (11)
PAH		94	20.7 (28)
MAO		68	18.4 (27)
CMU	[29]	111	30 (30)

Table 2
Cost parameters for symbolic datasets.

	Vertices			Edges		
	c_s	c_d	c_i	c_s	c_d	c_i
Setting 1	2	4	4	1	1	1
Setting 2	2	4	4	1	2	2
Setting 3	6	2	2	3	1	1

for a satisfactory solution. Instead of exploring all the search space, at each iteration, the x most promising partial solutions are kept in the set of promising candidates. In this contest, x was set to 100.

3. Protocol and datasets

Our evaluation is conducted on 4 Quad-Core AMD Opteron processor 8350, cadenced at 2.0GHz together with 16GB memory. The maximum number of threads was limited to 4 (i.e., for *F24threads* and *PDFS*). The time constraint used in GDC was fixed to 30 s. That is, the methods that needed more than 30 s were stopped, and the best answer found so far was outputted. Note that *F2, F24threads, DF* and *PDFS* are exact algorithms without time constraints.

3.1. Datasets and cost functions

Several datasets composed of graphs with symbolic and numeric attributes were used in the experiments. Table 1 synthesizes the characteristics of the first 5 datasets.

The first 4 datasets are composed of chemical compounds. Alkane and PAH datasets correspond to molecules only composed of carbons and may thus be considered as unlabeled graphs. On the other hand, MAO and Acyclic are composed of various types of atoms (carbons, oxygen, etc.). For these four datasets, all graph pairwise comparisons were computed. The cost of editing an element (a vertex or an edge) is given by:

$$c(a,b) = \delta_{f(a)=f(b)} c_s, \quad c(a,\epsilon) = c_d, \quad c(\epsilon,b) = c_i \tag{1}$$

where a and b are elements, $f(a)$ is the label of vertex a or the bound type of edge a. Note that $\delta_r = 1$ if r is true or 0 else, c_s is the cost of substituting elements with a same attribute, and c_d and c_i denote respectively the cost of deleting and inserting an element. Three different combinations of these parameters have been tested (see Table 2): Vertex insertion/deletion more expensive than vertex substitution (setting 1), vertex/edge insertion/deletion more expensive than vertex/edge substitution (setting 2), and vertex/edge insertion/deletion cheaper than node/vertex substitution (setting 3). These three settings aim at favoring either substitution or deletion/insertion edit operations and may exhibit if the performances of a same algorithm are constant upon different cases.

CMU dataset was made up of 660 graph pairs which were constructed from 111 images of a toy house captured from 9 different viewpoints. Each house was represented by 30 unattributed vertices while edges were attributed by the euclidean distance (integer-valued) between them. Vertices substitutions were fixed to a zero cost and to $+\infty$ for deletions and insertions, hence forbidding them. Edit operations on edges were penalized by the cost function:

$$c(e_i,e_j) = 0.5\,|d(e_i) - d(e_j)|, \quad c(e,\epsilon) = c(\epsilon,e) = 0.5\,d(e) \tag{2}$$

where $d(e)$ is the distance associated to edge e.

To analyze the behavior of the submitted approaches when the number of vertices increases, the well-known Mutagenicity (or MUTA) and GREC datasets [23] were considered but divided into disjoint subsets containing graphs with a same number of vertices, as proposed in [1]. The subsets are composed of graphs with 5, 10, 15 and 20 vertices, for GREC, and 10, 20, ..., 70 vertices for MUTA. Each subset is composed of 10 graphs, hence leading to 100 pairwise comparisons for each one. Since MUTA is composed of chemical compounds, we used cost functions defined by Eq. (1). The graphs of GREC have several attributes (both symbolic and numerical) defined on vertices and edges. The cost functions of GREC were defined as follows:

$$c(e_i,e_j) = \begin{cases} 0, & \text{if } f(e_i) = f(e_j) = 1 \text{ and } T(e_i) = T(e_j) \\ 15, & \text{if } f(e_i) = f(e_j) = 1 \text{ and } T(e_i) \neq T(e_j) \\ 0, & \text{if } f(e_i) = f(e_j) = 2 \\ 7.5, & \text{otherwise} \end{cases}$$

$$c(v_i,v_j) = \begin{cases} 0.5\,d_{euc}(pos(v_i), pos(v_j)), & \text{if } T(v_i) = T(v_j) = 1 \\ 90\,, & \text{otherwise} \end{cases} \tag{3}$$

$$\begin{aligned} c(e,\epsilon) &= c(\epsilon,e) = 7.5\,f(e) \\ c(v,\epsilon) &= c(\epsilon,v) = 45 \end{aligned}$$

where f is the frequency associated to edge e. T refers to the type of edges/vertices and pos is the x, y position of vertex v.

3.2. Performance evaluation metrics

Let $\mathcal{S}$ be a graph dataset and let $\mathcal{M}$ denote the set of GED methods listed in Section 2. Given a method $m \in \mathcal{M}$, we computed all the pairwise comparisons $d(G_i, G_j)^m$ (except on CMU as explained in Section 3.1), where $d(G_i, G_j)^m$ is the edit distance computed using method m on the graph pair $(G_i, G_j) \in \mathcal{S}^2$ under a time limit of 30 s.

A projection on a two-dimensional space ($\mathbb{R}^2$) was achieved by using *time-score* (Eq. (8)) and *deviation-score* (Eq. (7) which measure performance in terms of computational time and accuracy, respectively. Note that both criteria must be minimized. Given a dataset, mean deviation and mean computational time were derived as follows:

$$\overline{dev_{\mathcal{S}}^m} = \frac{1}{|\mathcal{S}| \times |\mathcal{S}|} \sum_{i=1}^{|\mathcal{S}|} \sum_{j=1}^{|\mathcal{S}|} dev(G_i,G_j)^m \quad \forall m \in \mathcal{M} \tag{4}$$

$$\overline{time_{\mathcal{S}}^m} = \frac{1}{|\mathcal{S}| \times |\mathcal{S}|} \sum_{i=1}^{|\mathcal{S}|} \sum_{j=1}^{|\mathcal{S}|} time(G_i,G_j)^m \quad \forall m \in \mathcal{M} \tag{5}$$

where $time(G_i, G_j)^m$ is the run time required by m to compute $d(G_i, G_j)^m$. The deviation $dev(G_i, G_j)^m$ measures the distance between $d(G_i, G_j)^m$ and the best known solution R_{G_i,G_j} (either optimal or not). Note that for a given pair of graphs G_i and G_j, the distance provided in the ground truth is considered as R_{G_i,G_j} if it is either that optimal distance or if it is lower than all the distances $d(G_i, G_j)^m$ $\forall m \in \mathcal{M}$. In the case where a method m obtained a lower $d(G_i, G_j)^m$ than the one in the ground truth, $d(G_i, G_j)^m$ is considered as R_{G_i,G_j}. Moreover, since we compute graph edit distance from an edit path encoded as a mapping, the approximation computed by a method is always an overestimation if it is not the exact graph

Table 3
Methods included in GDC.

Acronym	Reference	Details
BS-100	[21]	Beam-search of size 100
LSAPE	[5]	Linear Sum Assignment Problem with Edition
QAPE	[6]	Quadratic Assignment Problem with Edition
F2	[17]	Exact binary linear programming formulation
F24threads	This paper	Parallel version of *F2*
F2LP	This paper	Upper bound of *F2*
DF	[2]	Depth-first algorithm
DFUB	This paper	Upper bound of *DF*
PDFS	[3]	Parallel version of *DF*

edit distance. Therefore, the best approximation is defined as the lowest one. Deviation is defined as follows:

$$dev(G_i, G_j)^m = \frac{d(G_i, G_j)^m - R_{G_i,G_j}}{R_{G_i,G_j}}, \ \forall(i, j) \in [\![1, |\mathcal{S}|]\!]^2 \tag{6}$$

To obtain comparable results between databases, mean deviations and times were normalized:

$$\overline{deviation\_score^m} = \frac{1}{\#subsets} \sum_{\mathcal{S} \in subsets} \frac{\overline{dev^m_{\mathcal{S}}}}{max\_dev_{\mathcal{S}}} \tag{7}$$

$$\overline{time\_score^m} = \frac{1}{\#subsets} \sum_{\mathcal{S} \in subsets} \frac{\overline{time^m_{\mathcal{S}}}}{max\_time_{\mathcal{S}}} \tag{8}$$

where $max\_dev_{\mathcal{S}}$ and $max\_time_{\mathcal{S}}$ denote respectively the maximal mean deviation and the maximal mean execution time obtained among all methods on dataset $\mathcal{S}$.

The computational time, measured in seconds, includes all inherited costs computations and graphs parsing. We also count if an exact edit distance has been computed by a given method m on each graph pair. This metric is obviously only measured on datasets which exact graph edit distance has been computed.

4. Results and discussion

In this section, we present and analyze the results obtained by the submitted methods on all datasets. Table 3 summarizes the GED methods that were included in GDC.

For the sake of clarity, we synthesize our different conclusions via figures. For exhaustive and numerical results, we refer the interested reader to the contest website: http://gdc2016.greyc.fr.

4.1. Symbolic datasets

We ran all the methods on MUTA using the 3 Settings described in Section 3.1. Since the results using Settings 1 and 2 were similar, we only report the results obtained using Settings 2 and 3.

Fig. 1(a) sums up the running time (x-axis) and deviation scores (y-axis) on MUTA. This plot allows to see the trade-off between the running time required to compute an edit distance and the accuracy of this approximation. One can see that the average deviation of *F24threads* was the lowest (using Settings 2 and 3), which thus corresponds to the best method in terms of accuracy. Considering the trade-off between speed and quality, *QAPE* was the best candidate.

Fig. 1(b) shows the percentage of computations of exact graph edit distance (with or without optimality proof) using Settings 2 and 3. One can see that the number of optimal solutions found decreases as the size of graphs increases. However, *F2* and *F24threads* were able to find more optimal solutions than the other methods for a given size of graphs. The minimum number of found optimal solutions was 10 which corresponds to matching each graph with itself which is an easier problem.

Fig. 1(c) depicts the average running time obtained on MUTA subsets using Settings 2 and 3. We can note that the BLP approaches (i.e., *F2, F24threads* and *F2LP*) required more time when using Setting 2 than using Setting 3. This phenomenon, not observed on other methods, was induced by the topological constraints of the *F2*-based methods which take into account substitutions and not deletions and insertions. Thus, the more insertions and deletions are required, the easier the constraints are respected. One could also notice that on Muta-60 and Muta-70, the running time of *F2* and *F24thread* exceeded 30 s. This was due to the fact that the chosen mathematical solver cannot take the parsing phase into account when configuring the time constraint. Thus, to go beyond this problem, *F2*'s participants decided to limit the time constraint of these two methods to 28 s (instead of 30 s), leaving only 2 s to parse graphs. However, when graphs exceeded 50 vertices, the parsing time needed more than 2 s to compute which induced computational times exceeding the time limit.

Fig. 1(d) depicts the average deviation according to graph size. Using Setting 2, one can see that *F24threads* was the best method up to 40 vertices. On MUTA-50, 60 and 70, *QAPE* outperformed *F24threads* since the latter was unable to output satisfactory solutions. On the other hand, using Setting 3, *F24threads* obtained the lowest estimations of the GED even on graphs whose number of vertices was greater than 40. The algorithm that obtained the second best estimations was *QAPE*.

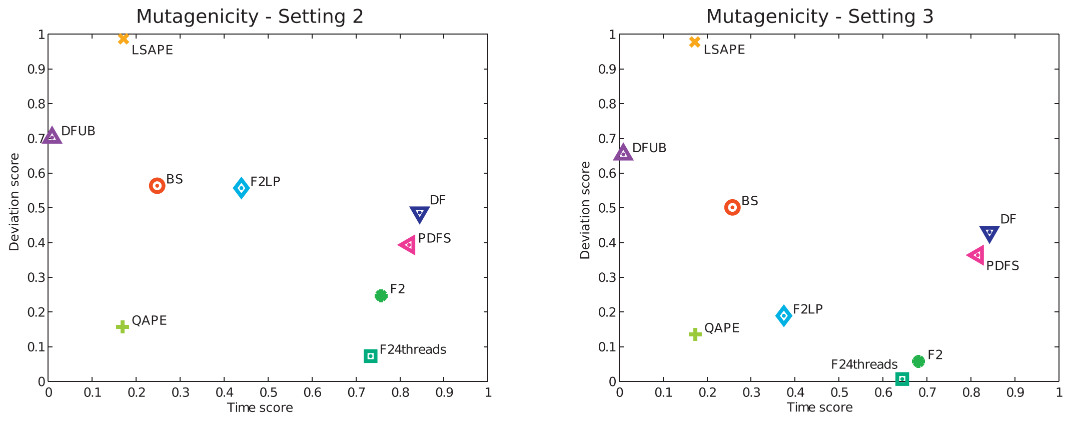
Concerning the 4 chemical datasets (i.e., Acyclic, Alkane, MAO and PAH), Settings 1, 2 and 3 did not significantly alter the behavior of the methods, thus in this paper, only the results of Setting 1 are reported. Fig. 2 shows the trade-off between deviation and time scores on the 4 chemical datasets. Similarly to MUTA subsets, the more accurate approximations were obtained by *F24threads* and *F2*. For a compromise between time and accuracy, *QAPE* was a good candidate.

PAH represented the most challenging dataset since it is composed of large unlabeled graphs. From Fig. 2(d), we can observe the difference in terms of computational time between *F24threads, F2, DF* and *PDFS*. Note that, in this case, *F24threads* took advantage of parallelism, which allowed this method to compute exact GEDs more often than the other methods. On this dataset, one can also notice a large difference in computation time between *BS, LSAPE, QAPE*, and *DFUB* methods on the one hand and *F2*, on the other hand. This point might be explained by the fact that *BS, LSAPE, QAPE* and *DFUB* aim at finding an approximate solution while *F2, F24thread, DF* and *PDFS* search for an optimal one. The important execution time needed by *PDFS* was due to the integrated misleading heuristic (Section 2) which was unable to prune the search tree as fast as possible. However, on simpler datasets, executions time of most methods was below 1 s.

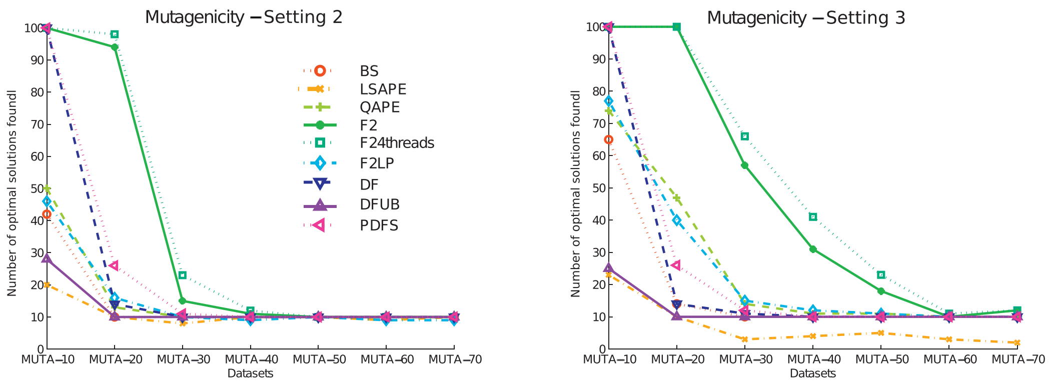
On Alkane, Acyclic and MAO, composed of smaller graphs than PAH, *F2* and *F24threads* were able to compute an optimal solution for any pair of graphs within 30 s (Fig. 2). Note that *DF* and *PDFS* methods were also very close to *F2* and were faster than *F2* and *F24threads* on Alkane. Considering PAH, *F24threads* obtained approximately 84%% of optimal solutions, followed by *F2* with approximately 62%% of optimal solutions.

4.2. Numeric datasets

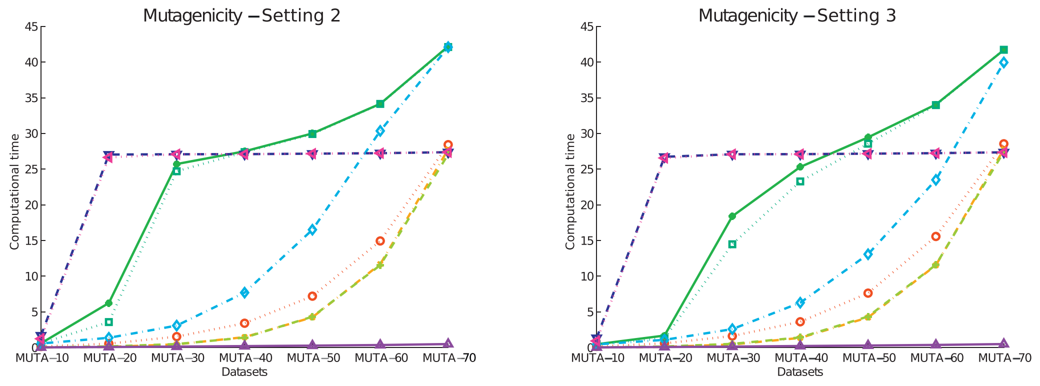
Regarding the numeric datasets (i.e. CMU and GREC), *LSAPE* and *QAPE* were not integrated in this part of GDC since their current implementation could only match graphs the attributes of which are symbolic. Fig. 3(a) shows the trade-off between computation time and deviation on GREC. As one could see, all the methods (except *DFUB*) had a small deviation. GREC is composed of graphs having at most 20 vertices, which thus constitutes a tractable problem for all methods. Note that *F2* and *F24threads* always found the



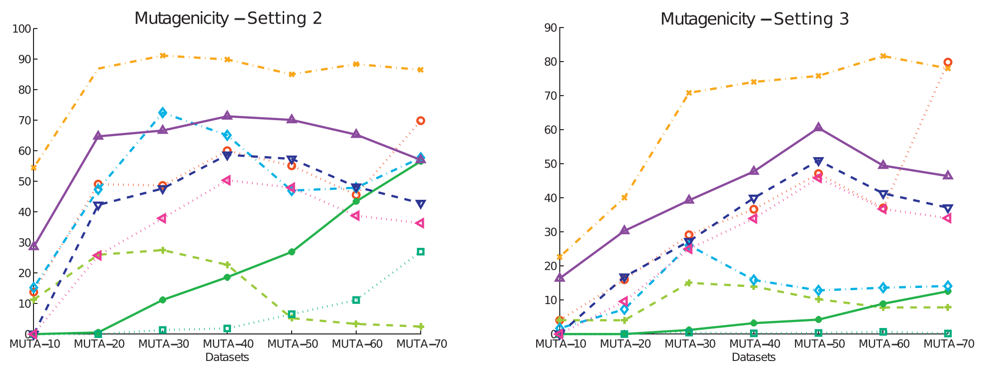
(a) Average speed-deviation scores on MUTA subsets



(b) Number of optimal solutions according to graph sizes on MUTA subsets



(c) Computation time according to graph sizes on MUTA subsets



(d) Deviation according to graph sizes on MUTA subsets

Fig. 1. Scores obtained on MUTA Datasets.

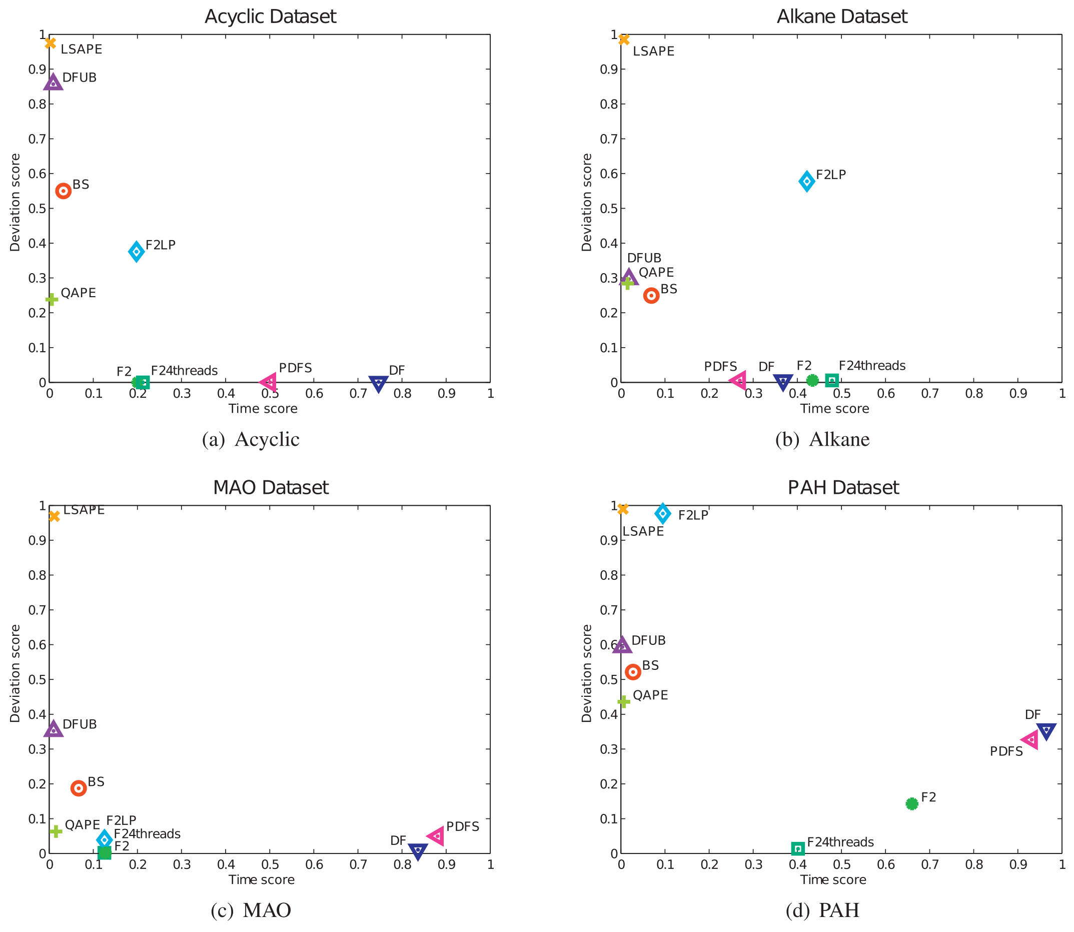


Fig. 2. Speed versus deviation scores on chemical datasets using Setting 1.

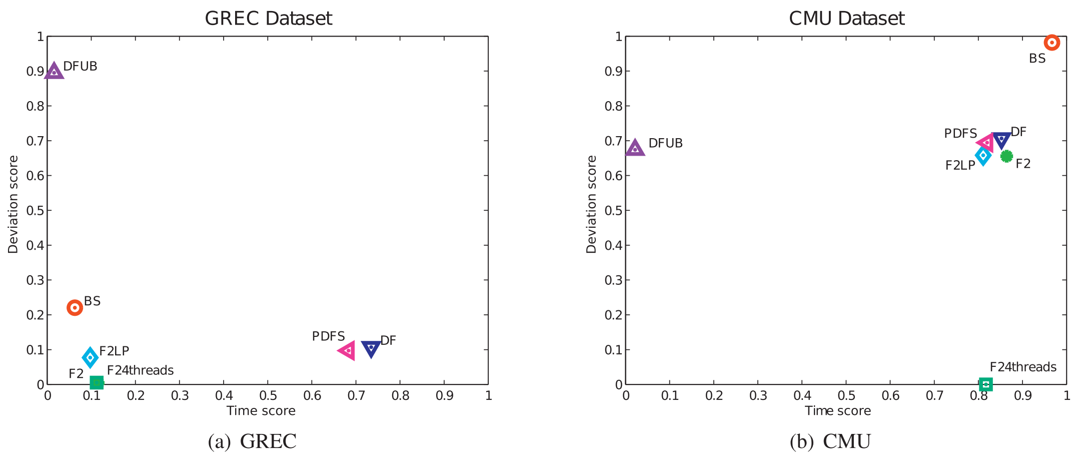


Fig. 3. Average deviation-running time scores.

optimal solutions. On the other hand, CMU was more challenging (see Fig. 3(d)). Indeed, most of the methods obtained a high deviation on this database, except for *F24threads* which had the best results in a comparable computation time with the other methods. Only *DFUB* was significantly faster but at a cost of accuracy. Even if *F24threads*'s deviation score was 0%%, it was only able to find approximately 51%% of optimal solutions. Such an observation emphasizes the complexity of the GED problems on this dataset. As a conclusion, *F24threads* represented a good compromise between deviation and running time on both CMU and GREC.

4.3. General conclusions

As a conclusion of GDC, among the exact methods (i.e., *F2, F24threads, DF* and *PDFS*), which output exact solutions when they are not restricted by time constraints, *F24threads* constitutes the best alternative. Indeed, this method always obtained the exact solution of GED on small graphs and still obtains a good percentage of exact edit distances when dealing with graphs having more than 20 vertices.

On the other hand, approximate GED methods (i.e. *LSAPE, QAPE, F2LP, DFUB* and *BS*) aim at providing a good approximation in a reduced computational time. Among these methods, *QAPE* usually obtained the best GED approximation within a reasonable computational time. Moreover, *QAPE* may find the optimal solution for many pairs of graphs. However, it cannot give any guarantee of the optimality of its result.

5. Challenges in graph edit distance

In this section, we present and discuss perspectives of evaluated methods and GED challenges upcoming in the near future. Regarding the 3 inspected GED categories, the branch-and-bound based algorithms are highly dependent on the lower bound. These algorithms could be improved by proposing other promising lower bounds with the help of machine learning techniques. On the other hand, currently, the implementation of the assignment-based algorithms cannot handle graphs with numeric attributes.

The BLP-based approaches were solved by off-the-shelf mathematical solvers for which the exploration of the solution space is not mastered by the user. It might be interesting to explore how the solving could benefit from the knowledge brought by approximate methods that can be computed in a small time. On the one hand, the early injection of upper and lower bounds provided by approximate methods could help to drastically prune the tree of solutions. On the other hand, even if they are not optimal, the assignments provided by approximate methods could be good start points in the search for an optimal solution.

Both exact and approximate GED approaches have common challenges, particularly in reducing computational time and consequently matching larger graphs. Considering approximate approaches, one direction may be to keep a nearly constant approximation error to obtain a reasonable confidence interval on the computed edit distance. Note that BLP-based approaches reached the highest accuracy on almost all datasets in our experiments. On this basis, it would be interesting to use other Operations Research techniques to solve GED. For instance, heuristics based on mathematical programming could be investigated to break down a given problem into a sequence of subproblems solved optimally.

Regarding GDC, the time constraint was arbitrary fixed to 30 s. One could relax or increase this time constraint to explore how methods are acting in different conditions, or, more precisely, study the deviation as a function of time. For instance, such a study will be very interesting for BLP-based methods to find a good trade-off according to accuracy vs. time constraints of users.

Generally, in GED, there is a lack of challenging and real-world datasets. As demonstrated in this paper, GED computation on GREC is easily solved by most of tested methods. On the other hand, MUTA, PAH and CMU were more challenging. It is of great interest to have more various types of datasets (e.g., big, dense and irregular graphs) that are dedicated to real-world applications (e.g., documents, social networks and object tracking). Moreover, in GDC, the ground truth is not always provided. It would be better to have datasets with their ground truth so as to be compare computed approximations to exact graph edit distance and not the best approximation.

Based on the aforementioned challenges, one could see that there exist different ways to improve GED computation. However, beyond that, one should take into account the reason why we need to compute GED (e.g., classification, matching or clustering problems). In this paper, we focused on just one challenge of GDC (i.e. at the matching level) and not on its use to resolve a given problem. A future contest should focus on the evaluation of GED approximations to classification or clustering problems. Moreover, each kind of problems could raise the issue of learning cost functions. Since GED is a minimization problem, the selected functions as well as the parameter values are not necessarily adapted with the user's provided solution. In other words, the user's vertex-to-vertex matching may not be the minimum solution found by a GED method. Recently, Cortés and Serratosa [9] have proposed to learn the parameters of cost functions based on the vertex-to-vertex matching provided by users. This approach is interesting since it shows the applicability of GED methods in real-world applications. On this basis, one could add visualizing the matching results as a performance evaluation metric to judge whether or not the matching results are relevant after taking the users' point of view into account.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.patrec.2017.10.007.

References

[1] Z. Abu-Aisheh, R. Raveaux, J. Ramel, A graph database repository and performance evaluation metrics for graph edit distance, in: Graph-Based Representations in Pattern Recognition, 2015, pp. 138–147.

[2] Z. Abu-Aisheh, R. Raveaux, J. Ramel, P. Martineau, An exact graph edit distance algorithm for solving pattern recognition problems, in: ICPRAM, 2015, pp. 271–278.

[3] Z. Abu-Aisheh, R. Raveaux, J. Ramel, P. Martineau, A Parallel Graph Edit Distance Algorithm, Technical Report hal-01476393, Univ. de Tours, 2017.

[4] S. Bougleux, L. Brun, V. Carletti, P. Foggia, B. Gaüzère, M. Vento, Graph edit distance as a quadratic assignment problem, Pattern Recognit. Lett. 87 (2017) 38–46.

[5] S. Bougleux, B. Gaüzère, L. Brun, A hungarian algorithm for error-correcting graph matching, in: IAPR-TC-15 International Workshop on Graph-Based Representations in Pattern Recognition, in: LNCS, vol. 10310, Springer International Publishing, 2017, pp. 118–127.

[6] S. Bougleux, B. Gaüzère, L. Brun, Graph edit distance as a quadratic program, in: International Conference on Pattern Recognition, 2016, pp. 1701–1706.

[7] V. Carletti, B. Gaüzère, L. Brun, M. Vento, Approximate graph edit distance computation combining bipartite matching and exact neighborhood substructure distance., in: Graph-Based Representations in Pattern Recognition, in: LNCS, vol. 9069, 2015, pp. 168–177.

[8] X. Cortés, F. Serratosa, Active-learning query strategies applied to select a graph node given a graph labelling, in: IAPR-TC-15 International Workshop on Graph-Based Representations in Pattern Recognition, in: LNCS, vol. 7877, Springer Berlin Heidelberg, 2013, pp. 61–70.

[9] X. Cortés, F. Serratosa, Learning graph matching substitution weights based on the ground truth node correspondence, Int. J. Pattern Recognit. Artif. Intell. 30 (2) (2016).

[10] B. Gaüzère, S. Bougleux, K. Riesen, L. Brun, Approximate graph edit distance guided by bipartite matching of bags of walks, in: Structural, Syntactic, and Statistical Pattern Recognition, in: LNCS, vol. 8621, 2014, pp. 73–82.

[11] F. Glover, M. Laguna, General purpose heuristics for integer programming—part i, J. Heuristics 2 (4) (1997) 343–358, doi:10.1007/BF00132504.

[12] IAPR TC 15, GREYC Datasets, 2013.

[13] D. Justice, A. Hero, A binary linear programming formulation of the graph edit distance, IEEE Trans. Pattern Anal. Mach. Intell. 28 (8) (2006) 1200–1214.

[14] H.W. Kuhn, The hungarian method for the assignment problem, Nav. Res. Logist. 2 (1–2) (1955) 83–97.

[15] E. Lawler, Combinatorial Optimization: Networks and Matroids, Holt, Rinehart and Winston, New York, 1976.

[16] M. Leordeanu, M. Hebert, R. Sukthankar, An integer projected fixed point method for graph matching and map inference, in: Advances in Neural Information Processing Systems, vol. 22, 2009, pp. 1114–1122.

[17] J. Lerouge, Z. Abu-Aisheh, R. Raveaux, P. Héroux, S. Adam, New binary linear programming formulation to compute the graph edit distance, Pattern Recognit. 72 (2017) 254–265.

[18] J. Lerouge, Z. Abu-Aisheh, R. Raveaux, P. Héroux, S. Adam, Exact graph edit distance computation using a binary linear program, in: Structural, Syntactic, and Statistical Pattern Recognition, 2016, pp. 485–495.

[19] Z.-Y. Liu, H. Qiao, GNCCP–graduated nonconvexity and concavity procedure, IEEE Trans. Pattern Anal. Mach. Intell. 36 (6) (2014) 1258–1267.

[20] J. Munkres, Algorithms for the assignment and transportation problems, J. Soc. Ind. Appl. Math. 5 (1) (1957) 32–38.

[21] M. Neuhaus, K. Riesen, H. Bunke., Fast suboptimal algorithms for the computation of graph edit distance, in: Structural, Syntactic, and Statistical Pattern Recognition, 2006, pp. 163–172.

[22] K. Riesen, Structural Pattern Recognition with Graph Edit Distance - Approximation Algorithms and Applications, Advances in Computer Vision and Pattern Recognition, Springer International Publishing, 2015.

[23] K. Riesen, H. Bunke, IAM graph database repository for graph based pattern recognition and machine learning, in: Structural, Syntactic, and Statistical Pattern Recognition, 2008, pp. 287–297.

[24] K. Riesen, H. Bunke, Approximate graph edit distance computation by means of bipartite graph matching, Image Vision Comput. 27 (7) (2009) 950–959.

[25] K. Riesen, S. Fankhauser, H. Bunke, Speeding up graph edit distance computation with a bipartite heuristic, Mining and Learning with Graphs, MLG, 2007.

[26] F. Serratosa, Fast computation of bipartite graph matching, Pattern Recognit. Lett 45 (2014) 244–250.
[27] F. Serratosa, Speeding up fast bipartite graph matching through a new cost matrix, Int. J. Pattern Recognit. Artif. Intell. 29 (02) (2015).
[28] F. Serratosa, X. Cortés, Graph edit distance: moving from global to local structure to solve the graph-matching problem, Pattern Recognit. Lett. 65 (2015) 204–210.
[29] C.-C. Wang, CMU Dataset, 2003.