

Graph Edit Distance: Accuracy of Local Branching from an application point of view

Mostafa Darwiche, Donatello Conte, Romain Raveaux, Vincent t'Kindt

▶ **To cite this version:**

Mostafa Darwiche, Donatello Conte, Romain Raveaux, Vincent t'Kindt. Graph Edit Distance: Accuracy of Local Branching from an application point of view. Pattern Recognition Letters, 2020, 134, pp.20 - 28. 10.1016/j.patrec.2018.03.033 . hal-01761595

HAL Id: hal-01761595

https://espci.hal.science/hal-01761595

Submitted on 18 Apr 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Graph Edit Distance: Accuracy of Local Branching from an application point of view

Mostafa Darwiche Donatello Conte Romain Raveaux
Vincent T'Kindt
Laboratoire d'Informatique Fondamentale et Appliquées de Tours (EA 6300),
64 avenue Jean Portalis, Tours, France
{*firstname.lastname*}@univ-tours.fr

April 18, 2018

Abstract

In the context of graph-based representations, comparing and measuring the dissimilarities between graphs can be done by solving the Graph Edit Distance (GED) problem. It is well known and embedded in many application fields such as Computer Vision and Cheminformatics. GED is a NP-hard minimization problem, therefore the optimal solution cannot be found in reasonable time. The GED problem has been addressed by exact approaches like Mixed Integer Linear Programs (MILP) formulations and heuristic approaches like beam-search, bipartite graph matching among others. Recently, a novel heuristic, called local branching (*LocBra*) for the GED problem, has been proposed and shown to be efficient. In this work, the focus is on evaluating *LocBra* with other competitive heuristics available in the literature from an application point of view. Moreover, it tries to answer the following question: is it important to compute an accurate GED regarding the final applications? Similarity search and graph matching are considered as final applications. Three experiments are conducted to evaluate the accuracy and efficiency of the heuristics. The quality of the obtained solutions and matching w.r.t. optimal solutions and ground-truth matching is studied. The results of those experiments show that LocBra has a high correlation with the optimal solutions and the ground-truth matching.

1 Introduction

A very convenient and efficient way to model objects and patterns is to use graph-based representations. Graphs provide a satisfactory structural representation of an object, by defining the main components that form the object using *vertices*, and drawing the relations between them using *edges*. More information and characteristics can be stored in the graph by assigning labels/attributes to vertices and edges. Attributed graphs have been actively used in many fields, such as *Pattern Recognition* to perform object recognition, image registration, tracking and many other applications [Sanfeliu et al., 2002]. Also, attributed graphs form a natural representation of the atom-bond structure of molecules, therefore they have applications in *Cheminformatics* field [Raymond and Willett, 2002]. Once the graphs representing the objects are at hand, tasks such as graph comparison and finding the (dis)similarities between graphs can be performed, leading to finding the best matching between them. The matching is expressed by determining the correspondences between vertices and edges of graphs based on the attributes carried on them. The matching process must be tolerant to the differences in the topology and attributes of the graphs, since it is unlikely to have isomorphism between graphs in real-life scenarios. This is known as *Error-Tolerant Graph Matching* (ETGM) class of problems, which are difficult to deal with, due to their computational complexity.

One of the most important problems that belongs to ETGM class is the *Graph Edit Distance* (GED). It consists in finding the minimum cost needed to transform one graph into another, through a series of edit operations [Bunke and Allermann, 1983]. The possible edit operations are substitution, insertion and deletion of vertices or edges, with a cost associated to each operation. The GED problem has been shown to be a generalization of other graph matching problems (e.g. maximum common subgraph or graph and subgraph isomorphism), by simply changing

the cost metric properties [Bunke, 1997, Bunke, 1999]. [Stauffer et al., 2017] provide a taxonomy for the application fields: Image Analysis, Handwritten Document Analysis, Biometrics, Bio and Cheminformatics, etc. All these applications require performing graph searches among databases of graphs. For instance, an unknown graph that models an object in an image must be compared with all graphs in a database of known objects in order to find similarities. [Zeng et al., 2009] classify the graph searches into three categories, for a database of graphs $D = \{g_1, g_2, ..., g_n\}$, and a query graph q:

- Full search: find all graphs g_i in D that are the same as q,
- Subgraph search: find all graphs g_i in D that contain or are contained by q,
- Similarity search: find all graphs g_i in D that are similar to q, based on some defined similarity measure.

The GED problem can be involved in the three aforementioned categories, and it has become widely considered as a dissimilarity measure. In the context of similarity search, GED is applied to perform classification of graphs and graph retrieval.

However, and since graphs are flexible and can be large and complex, in the case of modeling complex patterns and objects, solving to optimality the GED problem becomes difficult and intractable in practice. In fact, GED has been proven to be in the class of NP-hard problems [Zeng et al., 2009]. In the literature, both exact and heuristics methods can be found.

In the exact context, the GED problem was addressed by exact methods based on mathematical programming, and more precisely by *Mixed Integer Linear Program* (MILP) formulations. [Lerouge et al., 2017] have proposed two MILP formulations (*F1* and *F2*) to solve the GED problem. *F1* simply minimizes the cost of assigning the vertices plus the cost of assigning the edges. *F2* is a reformulation of *F1* with a reduced number of variables and constraints. Another interesting MILP formulation ($MILP^{JH}$) is introduced by [Justice and Hero, 2006] for the specific case, where the attributes on edges are ignored and the edge edit operations have unitary costs. Nevertheless, $MILP^{JH}$ was shown to be the most efficient in solving the GED problem for this specific case [Lerouge et al., 2017].

Besides the exact methods, researchers have designed fast heuristics to solve the problem and provide good quality solutions. For instance, one of the famous heuristics is the *Bipartite Graph Edit Distance* (*BP*), which was developed by [Riesen et al., 2007]. The main advantage of *BP* is that it is fast and can solve large and hard instances in a short amount of time. It breaks down the GED problem into a *Linear Sum Assignment Problem* (LSAP), by computing a special cost assignment matrix for the vertices sets of the two graphs. The cost matrix covers all possible vertices assignments, e.g. cost of assigning one vertex to another, or cost of deleting/inserting that vertex. In the same cost matrix, it embeds an estimation cost for every vertices assignment, which stands for the cost of assigning the edges connected to the current vertices. Then, the LSAP problem is solved by the *Hungarian* method [Munkres, 1957] in $O\left((n+m)^3\right)$ time, with n and m the number of vertices of the two graphs. *BP* is considered as a fast heuristic, but it only considers local structures around vertices, rather than a global one. Two improved versions of *BP* are proposed afterwards by [Serratosa, 2015], *Fast Bipartite GED* (*FBP*) and *Square Fast Bipartite GED* (*SFBP*). Both suggest modifications (rectangular and square) to the cost matrix in order to boost the solution of *BP*. The same author has conducted a study on the three bipartite methods to compare their accuracy and running time under different settings, in the classification context. Later, an extended version of *BP* is also presented by [Ferrer et al., 2015], referred to as *SBPBeam*. It combines *BP* heuristic with a beam-search heuristic. SBPBeam uses *BP* to compute a solution and then using a beam-search approach, it tries to improve the solution by swapping two matched vertices and recomputing the GED. Beam-search constructs the search tree for all possible vertices swapping, but only processes the first α nodes in the tree (with α the beam size parameter). In a recent work, [Bougleux et al., 2017] have proposed two heuristics *Integer Projected Fixed Point* (*IPFP*) and *Graduate Non Convexity and Concavity Procedure* (*GNCCP*). Both are adapted to operate over a *Quadratic Assignment Problem* (*QAP*) that models the GED problem. The heuristics aim at approximating the quadratic objective function to compute a solution and then improve it by applying projection methods. The computation of a good initial solution is crucial to obtain good solutions in the end. Both heuristics use as an initial solution the one computed by *BP* heuristic. Also recently, an innovative approach to deal with the GED problem has been proposed by [Darwiche et al., 2018]. It introduces a new heuristic called *Local*

Branching (*LocBra*), which is an adapted version of the general local branching metaheuristic originally proposed by [Fischetti and Lodi, 2003]. The idea is to solve MILP formulations, within a branching heuristic, to perform local searches in defined regions in the solution space. *LocBra* is reviewed in the following sections.

The present work aims at exploring the efficiency and accuracy of the most promising heuristics that solve the GED problem. It is also a follow up to the paper by [Darwiche et al., 2018] that presented *LocBra* heuristic. *LocBra* was tested against the heuristics available in the literature and was proven to be efficient in minimizing the GED distance. In this work, three application-oriented experiments are proposed to evaluate the heuristics. The target of the first experiment is to show the link between the optimal solutions and the solutions computed by the heuristics: heuristics that return solutions close the optimal ones are considered to be accurate. Such an evaluation is classic in the literature. The second experiment is designed in similarity search context. A query graph is compared to each graph of a database thanks to a given GED heuristic used as a distance. Then, distances are sorted in ascending order to obtain a ranking. The target of this experiment is to compare the ranking given by all the heuristics against the optimal ranking given by optimal methods. Though, this kind of evaluation has not been done when evaluating GED heuristics. Last and third experiment studies the relation between the ground-truth matching given by human experts and the matching computed by the heuristics. Also, this evaluation is not common in the context of heuristics comparison, in spite of its importance. Visualizing the computed matching and comparing it with the ground-truth matching enable assessing the impact of mismatched vertices. The contribution of this paper is then, the study of application-dependent criteria to evaluate GED heuristics in order to decide if a good minimizer is also good in satisfying applications requirements. In addition, showing that second and third experiments are important when evaluating GED heuristics. The results of the experiments will reveal which heuristic is the most accurate in terms of distance, ranking and matching.

The remainder is organized as follows. Section 2 presents the GED problem and a review of the $MILP^{JH}$ formulation. Section 3 discusses the GED applications and challenges. Section 4 is devoted to review the *LocBra* heuristic. Then, section 5 reports the results of the intensive computational experiments. Finally, Section 6 highlights some concluding remarks.

2 GED definition and $MILP^{JH}$ model

An attributed graph is a 4-tuple $G = (V, E, \mu, \xi)$ where, V is the set of vertices, E is the set of edges, such that $E \subseteq V \times V$, $\mu : V \to L_V$ (resp. $\xi : E \to L_E$) is the function that assigns attributes to a vertex (resp. an edge), and L_V (resp. L_E) is the label space for vertices (resp. edges).

Next, given two graphs $G = (V, E, \mu, \xi)$ and $G' = (V', E', \mu', \xi')$, solving the GED problem consists in transforming one graph source into another graph target. To accomplish this, GED introduces the vertices and edges edit operations: $(u \to v)$ is the substitution of two nodes, $(u \to \epsilon)$ is the deletion of a node, and $(\epsilon \to v)$ is the insertion of a node, with $u \in V, v \in V'$ and ϵ refers to the empty node. The same logic holds for the edges. The set of operations that reflects a valid transformation of G into G' is called a complete edit path, defined as $\lambda(G, G') = \{o_1, ..., o_k\}$ where o_i is an elementary vertex (or edge) edit operation and k is the number of operations. GED is then

$$d_{min}(G, G') = \min_{\lambda \in \Gamma(G,G')} \sum_{o_i \in \lambda} \ell(o_i) \tag{1}$$

where $\Gamma(G, G')$ is the set of all complete edit paths, d_{min} represents the minimal cost obtained by a complete edit path $\lambda(G, G')$, and ℓ is the cost function that assigns the costs to elementary edit operations.

$MILP^{JH}$ is a model proposed by [Justice and Hero, 2006] to solve the GED problem. The main idea consists in determining the permutation matrix minimizing the L_1 norm of the difference between adjacency matrix of the input graph and the permuted adjacency matrix of the target one. The details about the construction of the model can be found in [Justice and Hero, 2006]. The model is as follows:

$$\min_{x,s,t \in \{0,1\}^{N \times N}} \left(f(x, s, t) = \sum_{i=1}^{N} \sum_{j=1}^{N} \ell\left(\mu(u_i), \mu'(v_j)\right) x_{ij} + \left(\frac{1}{2} \cdot \kappa \cdot (s_{ij} + t_{ij}) \right) \right) \tag{2}$$

such that

$$\sum_{k=1}^{N} A_{ik}x_{kj} - \sum_{c=1}^{N} x_{ic}A'_{cj} + s_{ij} - t_{ij} = 0 \ \forall i,j \in \{1,2,...,N\} \tag{3}$$

$$\sum_{i=1}^{N} x_{ik} = \sum_{j=1}^{N} x_{kj} = 1 \ \forall k \in \{1,2,...,N\} \tag{4}$$

where A and A' are the adjacency matrices of graphs G and G' respectively, $\ell : (\mu(u_i), \mu'(v_j)) \rightarrow \mathbb{R}^+$ is the cost function that measures the distance between two vertices attributes. As for x, s and t, they are the permutation matrices of size $N \times N$, and of boolean type, with $N = |V| + |V'|$. Besides, x represents the vertices matching i.e. $x_{ij} = 1$ means a vertex $i \in V \cup \{\epsilon\}$ is matched with vertex $j \in V' \cup \{\epsilon\}$. While s and t are for edges matching. Hence, the objective function (Eq. 2) minimizes both, the cost of vertices and edges matching. As for constraint 3, it is to make sure that when matching two couples of vertices, the edges between each couple have to be mapped. Constraint 4 guarantees the integrity of x i.e. one vertex in G can be only matched with one vertex in G'. This model has a limitation that it does not consider the attributes on edges, so edge substitution cost is 0 while deletion and insertion have a $\kappa \in \mathbb{R}^+$ fixed cost.

3 GED applications and challenges

GED has been used in many application fields, as reported by [Stauffer et al., 2017] who provided a taxonomy for these fields: Image Analysis, Handwritten Document Analysis, Biometrics, Bio- and Chem-informatics, Knowledge and Process Management, Malware Detection, and others applications. In Image Analysis field, some of GED applications are: object detection and tracking, image registration and 3D image analysis. GED can be applied to Handwritten Document Analysis to perform for example Keyword Spotting. In Cheminformatics, GED is used to measure the dissimilarities between the graphs modeling chemical molecules. This enables, for instance, grouping molecules that share the same structure (clustering). In all these fields and applications, a common task is graph comparison. So, when a new/unknown graph comes up and the goal is to search in a database of graphs looking for similar or exact graphs. [Zeng et al., 2009] have classified the graph searches in three categories: **i-** Full search, **ii-** Subgraph search and **iii-** Similarity search. Furthermore, in most of the aforementioned applications, the graph search mostly used is the similarity search. The reason is that the graph query may differ from the graph models stored in the database. As an example, graphs that model patterns in images contain extra vertices and edges, because the image quality may be poor or noisy. Therefore, graph and subgraph isomorphism might not be useful as much as similarity search, which is flexible and tolerates the structure and attributes differences in graphs. In addition, similarity search is effective in supervised classification by k-nearest neighbors and unsupervised classification by k-medians clustering. Another important application using the similarity search is graph retrieval. Subsequently, GED problem fits in this context and can be used to perform similarity searches.

The GED problem leads to minimize the distance between two graphs by solving an ETGM problem. The matching consists of the operations applied on the two sets of vertices. It is also called the assignment of the vertices of both graphs. For two sets $V = \{u_1, u_2, u_3\}$ and $V' = \{v_1, v_2\}$, a matching is expressed with the matrix:

$$Matching = \begin{array}{c} \begin{matrix} u_1 & u_2 & u_3 & \epsilon \end{matrix} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \end{array} \begin{matrix} v_1 \\ v_2 \\ \epsilon \end{matrix}$$

Whenever a vertex $u \in V$ (resp. in $v \in V'$) is matched with ϵ, it is said to be deleted (resp. inserted), otherwise u is substituted with v. The same as vertices matching, edges matching can be represented by a matrix. It is of interest to end-users to evaluate and understand the matching, by looking at the matched components for a query graph with similar graphs found in the database. Actually, the matching can help interpreting the results and detecting relevant spot patterns. GED problem enables having both the distance and the matching, which is convenient to end-users. Unlike, other similarity search approaches such as graph embedding into vector space, the matching is lost and cannot be reconstructed.

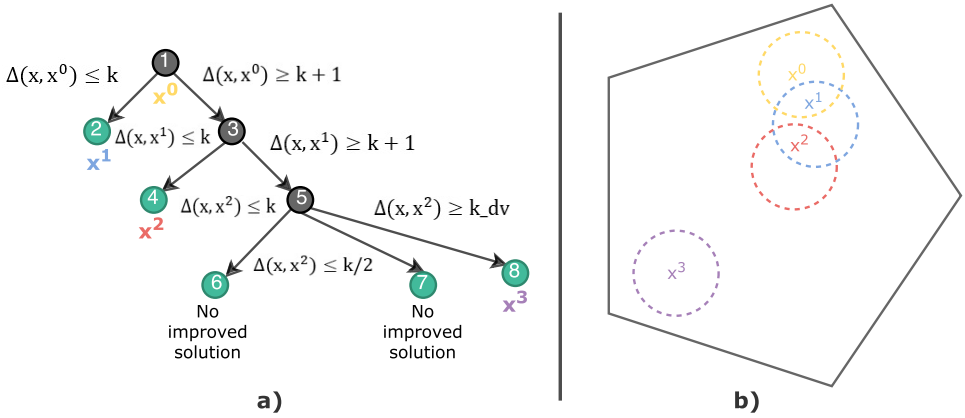


Figure 1: Local branching flow. a) depicts the left and right branching. b) shows the neighborhoods in the solution space

Another aspect to consider when talking about GED is the cost of edit operations. Solving the GED problem implies minimizing the edit operations cost to transform one graph into another. Each edit operation has an associated cost function. Cost functions can take into account vertex or edge attributes. As well, cost functions must reflect the user need, thus they can be learned to fit a specific goal. For instance, the goal is to reduce the gap between the ground-truth matching and the optimal matching. The ground-truth matching is usually given by human experts (aka Oracle) and reflects the true matching between pair of graphs. When the ground-truth is missing, cost functions can also be hand-crafted based on domain-dependent knowledge introduced by an expert of the application.

Finally, solving the GED problem implies a similarity search that is based on a distance function. Many heuristics can be found in the literature that solve the GED problem. Some of them are designed to converge very fast, while others favor the quality of the solutions over the running time. In all cases, there is a compromise between the running time of the heuristic and the accuracy of the solution returned. The only way to overcome this is by checking the application requirements and factors like graph size, density, type of attributes, which are in general responsible in making the problem hard to solve. An application like object detection in images is, in most cases, considered as a real-time application. Therefore, there is a need for having a very fast algorithm, but on the other hand, the graphs being compared should be of small sizes. In other applications, such as finding similar graphs of complex chemical molecules, it is affordable to spend extra running time in order to obtain accurate results. At the end, this compromise is a hard constraint, but there are several promising attempts and heuristics in the literature. Based on these arguments and regarding the final application, is it worth spending time to compute accurate solutions? What is the impact of a better solution on the similarity search or on the matching? These are the main questions that the experimentation conducted will try to answer on.

4 Local Branching Heuristic for the *GED* problem

Local Branching is a general MILP metaheuristic presented by [Fischetti and Lodi, 2003]. MILP formulations are known to be efficient in modeling complex combinatorial optimization problems. These formulations can be solved by using black-box solvers that try to find the optimal solution. Such solvers (e.g. CPLEX) are powerful and equipped with an arsenal of effective algorithms to solve MILP formulations. However, they are not capable all the time of finding the optimal solution, especially in the case of large and complex instances, due to high computational time and memory size needed. To benefit from the power of the solvers, local branching employs a solver to explore the solution space of MILP formulation in a defined scheme. It performs a series of local searches and focuses the search in defined regions looking for good quality solutions. The work presented by [Darwiche et al., 2018] provides an adapted version of local branching for the GED problem. $MILP^{JH}$ is chosen in the implementation of local branching and CPLEX is picked as a solver. LocBra is based on 4 main ingredients:

- Neighborhood definition: giving a solution $x^p \in \{0,1\}^{N \times N}$ to the problem, $N(x^p, k)$ is the *k-opt neighborhood* around x^p, with k a given positive integer. $N(x^p, k)$ is defined by adding the following *local branching constraint* to $MILP^{JH}$:

$$\Delta(x, x^p) = \sum_{(i,j) \in S^p} (1 - x_{ij}) + \sum_{(i,j) \notin S^p} x_{ij} \leq k \tag{5}$$

  with, $S^0 = \{(i,j) : x^p_{ij} = 1\}$. The neighborhood set contains the solutions that are within a distance no more than k from x^p (in the sense of the *Hamming distance*).

- Intensification: after defining the $N(x^p, k)$ by adding the local branching constraint, LocBra solves the corresponding *MILP* formulation by CPLEX. This intensifies the search by exploring the local neighborhood around x^p looking for a new and better solution. This step implies focusing the search into a small region of the solution space instead of exploring the whole space.

- Complementary intensification: it may happen that, when exploring the neighborhood $N(x^p, k)$, CPLEX fails to compute a feasible solution in the imposed CPU time limit. This corresponds to the case where this neighborhood is too large. Then, a complementary intensification phase is performed in the restricted neighborhood $N(x^p, k/2)$.

- Diversification: this step is introduced when the complementary intensification step fails to find an improved solution, which basically interpreted as the current solution x^p being a local optimum. The main goal of this step is to skip local minimum and switch the exploration to new regions in the solution space. To do so, the following constraint is added to $MILP^{JH}$:

$$\Delta'(x, x^p) = \sum_{(i,j) \in S^p_{imp}} (1 - x_{ij}) + \sum_{(i,j) \notin S^p_{imp}} x_{ij} \geq k_dv \tag{6}$$

  with B_{imp} the index set of binary important variables and $S^p_{imp} = \{(i,j) \in B_{imp} : x^p_{ij} = 1\}$. The notion of important variables is explained later.

LocBra procedure is detailed in Algorithm 1 putting all the above 4 ingredients together. The method halts when the total time set is reached and the best solution found is returned. Figure 1 gives an example of LocBra flow. Starting from an initial solution x^0 at node 1, the neighborhood $N(x^0, k)$ is defined by adding the constraint $\Delta(x, x^0) \leq k$. At node 2, an intensification step is performed and a new improved solution x^1 is found. Note that, in order to avoid visiting a solution already seen, the last constraint becomes $\Delta(x, x^0) \geq k + 1$. Again, the neighborhood $N(x^1, k)$ is defined and the intensification is triggered leading to x^2 (node 4). However, the intensification at node 6 seems to have failed at finding an improved solution. Therefore, a complementary intensification is applied at node 7. Since also, it did not find a better solution, a diversification step is performed to skip this region of the solution space. Finally, Figure 1-b shows the evolution of the solutions searches and neighborhoods in the solution space.

The diversification step has an important role in escaping local optima. An improved version of the original one by [Fischetti and Lodi, 2003] is proposed that is specific to the GED problem. The diversification constraint 6 is introduced over the set B_{imp}, which is the index set of important binary variables. The selection of these variables is based on the assumption that one variable is considered important if changing its value from $1 \rightarrow 0$ (or the opposite) highly impacts the objective function value. Forcing the solver to find a new solution with changes to the values of these variables enables changing the search region in the search space and the matching. So, B_{imp} is defined as follows: **i-** it computes a special cost matrix $[M_{ij}]$ for each possible assignment of a vertex $u_i \in V \cup \{\epsilon\}$, to a vertex $v_j \in V' \cup \{\epsilon\}$. Each value $M_{ij} = c_{ij} + \theta_{ij}$, where c_{ij} is the vertex edit operation cost induced by assigning vertex u_i to vertex v_j, and θ_{ij} is the cost of assigning the set of edges $E_i = \{(u_i, w) \in E\}$ to $E_j = \{(v_j, w') \in E'\}$. This assignment problem, of size $max(|E_i|, |E_j|) \times max(|E_i|, |E_j|)$, is solved by the Hungarian algorithm [Munkres, 1957] which requires $O\left(max(|E_i|, |E_j|)^3\right)$ time. **ii-** the standard deviation is computed at each row of $[M_{ij}]$, resulting in a vector $\sigma = [\sigma_1, ..., \sigma_{|V|}]$. Basically, a σ_i with a high value indicates that the variables representing assignments with vertex u_i have impact on the objective function value more than the other variables. **iii-** To select u_i vertices, the values of vector σ are split into two clusters C_{min} and C_{max}, using a simple clustering algorithm. **iv-** Finally, for every σ_i belonging to C_{max}

Algorithm 1: *LocBra* algorithm

```
1  Function LocBra(k, total_time_limit)
       Output: bestSol, opt
2      curr_sol ← ComputeInitialSol();
3      best_sol ← curr_sol ;
4      while tl < total_time_limit do
5          curr_sol ← Intensification(k);
6          if curr_sol is worse than best_sol then
7              curr_sol ← Intensification(k/2)
8          end
9          if curr_sol is still worse than best_sol then
10             curr_sol ← Diversification()
11         end
12         UpdateTime(tl);
13     end
14 End
```

cluster, the indices of all binary variables x_{ij} that correspond to the assignments of vertex u_i are added to B_{imp}. Consequently, the local structure of a vertex is considered to assess its influence on the objective function value. This version of the diversification is more efficient that the original one proposed by [Fischetti and Lodi, 2003], because it includes information about the instance at hand.

5 Experimentation Results

This section presents the experimentation results to the different tests conducted on *LocBra* and the heuristics selected from the literature. The goal of the experiments is to study the accuracy of each heuristic and its influence in GED applications. In the following sub-sections, first the experiments protocol is explained, followed by the heuristics settings and then the results of each experiment. Furthermore, all the experiments results and settings, databases and cost functions, in addition to videos highlighting the matching and more, can be found on this website[1].

5.1 Experiments Protocol

In the following, three experiments are conducted in order to compare the heuristics by examining the quality of the solutions and matching computed with: optimal solutions, ranking (searching for the nearest neighbor graphs) and ground-truth matching.

1st Experiment As in the paper [Darwiche et al., 2018], this first experiment will answer the following question: which heuristic is the best GED minimizer? It is about studying the closeness of the solutions returned by the heuristics to the optimal ones. The metric used in this experiment is the deviation percentage: given an instance I and a heuristic H, deviation percentage is equal to $\frac{solution_I^H - optimal_I}{optimal_I} \times 100$, with $optimal_I$ the optimal solution for I.

2nd Experiment An important question is brought up: what is the impact of the GED heuristics on the (dis)similarity search? The second experiment intends to answer this question by evaluating the ranking of the graphs, which is considered as an important task in graph retrieval. For a target graph, the GED is computed against the rest of the graphs in the database. Then ordering the obtained solutions by ascending order will show first the graphs with high resemblance w.r.t. the target one. Consequently, the experiment will compare the orders provided by the heuristics and the order provided by an exact method. It proceeds as follows: assuming a graph database with 5 graphs $D_G = [g_1, g_2, ..., g_5]$. Starting with graph g_1, the optimal and heuristics solutions (distances) are computed for all possible pairs of graphs e.g. $(g_1, g_1); (g_1, g_2); ...(g_1, g_5)$. Then, graphs are ordered by ascending order based on the GED solutions. For instance, assuming that

[1] https://sites.google.com/site/gedlocbra/

Table 1: LocBra vs. heuristics from the literature on PAH instances

	LocBra	*SBPBeam*	*IPFP*	*GNCCP*
t_{avg}	3.03	4.38	**1.45**	2.63
d_{avg}	**0.35**	371.14	117.37	81.04
η_I	**8702**	100	491	1143

the optimal order for g_1 is $g_1^{opt} = [g_1, g_2, g_4, g_3, g_5]$ and a heuristic H order is $g_1^H = [g_1, g_5, g_4, g_3, g_2]$. Then, the metric used is Kendall rank correlation coefficient τ_b. Kendall is a statistic used to study the correlation between two ranked/sorted ordinal variables [Agresti, 2010]. Computing τ_b consists in measuring the degree of concordance between the two ranked variables. An ordinal variable is a categorical variable for which the possible values are ordered. The correlation τ_b is computed between g_1^{opt} and g_1^H. Then, to analyze the τ_b values, a *p-value* test is applied, which is a statistical test based on the *null hypothesis* that assumes two variables (vectors) are uncorrelated and $\tau_b = 0$. The *alternative hypothesis* is that the variables are correlated, and τ_b is non-zero. If p-value is smaller than a threshold (referred to as significance level), then the null hypothesis is rejected, and the alternative is accepted. τ_b and p-value are computed for the rest of the heuristics after ordering the solutions for all pairs of graphs. So far, this is done for g_1, the same process is repeated for the rest of the graphs in the database. Finally, the p-values obtained for each graph and for each heuristic are grouped by heuristics, and the average of the values smaller than the threshold are calculated for each heuristic. The heuristic with the highest percentage, means that the null hypothesis is rejected and eventually the solutions returned by it, are strongly correlated with the optimal ones.

3rd Experiment The main question to be answered by this experiment is: does the best GED minimizer guarantees finding the ground-truth matching? For many graph databases, there is a ground-truth matching that is given by human experts (aka Oracle). They represent the true matching expected to be obtained between each pair of graphs. Therefore, this third experiment consists in studying the closeness of the matching computed by the heuristics to the ground-truth matching. The metric used here is the *Hamming Distance* between the ground-truth and heuristics matching. A matching is represented through a binary matrix, each value refers to an assignment of two vertices, e.g. $[x_{ij}]$ is a matching matrix, where i and j are the indices of two vertices $u_i \in V \cup \{\epsilon\}$ and $v_j \in V' \cup \{\epsilon\}$. The Hamming distance simply counts the number of positions at which the corresponding values are different. For two matching matrices $[x_{ij}]$ and $[y_{ij}]$, with $i \in \{1, 2, ..., N\}$, $j \in \{1, 2, ..., M\}$ and $N = |V| + 1$, $M = |V'| + 1$,

$$\mathcal{HD}(x, y) = \sum_{i=1}^{N} \sum_{j=1}^{M} (1 - \delta(x_{ij}, y_{ij})) \tag{7}$$

where $\delta : \mathbb{R}^2 \to \{0, 1\}$ is the Kronecker delta function, it returns 1 when $x_{ij} = y_{ij}$ and 0 otherwise. Note that the value obtained is then normalized by dividing with $N + M$, in order to return representative quantities between $[0, 1]$. The normalized distance is denoted by $\widehat{\mathcal{HD}}$. Therefore, what is of interest is when $\widehat{\mathcal{HD}} = 0$, because it means that the heuristic matching is equivalent to the ground-truth matching.

In summary the experiments are categorized as follows.

Index	Evaluation Metric	Application
Distance	Deviation	Near-optimal quality
Ranking	Kendall correlation	Similarity search/graph retrieval
Matching	Hamming Distance	Result interpretation

5.2 Comparative heuristics and experimentation settings

Very recent and efficient heuristics are selected in the experiments: *SBPBeam* [Ferrer et al., 2015], *IPFP* and *GNCCP* [Bougleux et al., 2017]. *SBPBeam* is picked over *FBP* and *SFBP*, since they were mainly introduced to improve the running time of *BP*. While *SBPBeam* tries to improve the solution of *BP*, which is in line with the purpose of these experiments. [Darwiche et al., 2018] conducted extensive experiments on these heuristics, considering two versions of each: original and extended. The original versions are the heuristics with their default parameter values as they were published by their authors. The extended versions are the heuristics with bigger parameter

Table 2: LocBra vs. heuristics from the literature on MUTA instances.

	S	10	20	30	40	50	60	70	Mixed
	opt_I	**100**	**100**	**100**	**99**	**92**	**71**	**35**	**91**
LocBra	t_{avg}	**0.17**	**1.12**	212.36	364.86	573.51	727.58	474.97	276.79
	d_{avg}	**0.00**	**0.00**	**0.00**	**0.06**	**0.04**	**0.47**	**0.76**	**0.22**
	η_I	**100**	**100**	**100**	**98**	**97**	**70**	**28**	**84**
SBPBeam	t_{avg}	0.84	10.02	47.65	139.75	322.06	590.47	1144.21	279.17
	d_{avg}	20.43	44.90	76.45	82.54	99.55	99.68	107.02	23.73
	η_I	14	10	10	10	10	10	10	11
IPFP	t_{avg}	1.20	9.62	48.90	115.14	237.91	469.15	**474.12**	240.32
	d_{avg}	3.44	10.18	16.45	17.21	19.15	20.18	28.19	5.37
	η_I	69	29	14	11	10	10	10	21
GNCCP	t_{avg}	0.55	6.41	**29.80**	**81.24**	**193.61**	**375.88**	951.65	**140.51**
	d_{avg}	3.23	10.67	24.20	21.79	24.69	18.67	36.57	10.14
	η_I	81	34	4	6	5	9	5	17

values in order to extend their running time, and so they can reach the time given to *LocBra*. This is to make the comparison consistent and fair by allowing all heuristics to run approximately the same amount of time. Based on the results obtained in [Darwiche et al., 2018], *LocBra* was more accurate in comparison to both original and extended versions of the heuristics. In addition, the extended versions of the heuristics performed better than the original versions. Hence, the experiments in this paper considers only the extended versions. For each experiment the appropriate parameter values are calculated first and then, the heuristics are launched. The parameters of each heuristic are: *LocBra* with k the neighborhood size, k_dv distance from current solution when diversifying, $total_time_limit$ is the total running time before the algorithm halts, $node_time_limit$ the maximum running time for intensification, UB the time limit to compute a first solution; *SBPBeam-*α with α the beam size; *IPFP-it* with it the number of iterations before the algorithm stops; *GNCCP-d* with d the quantity to be reduced from the variable that controls the concavity and convexity of the objective function.

Based on *LocBra* definition, it uses a MILP formulation and a MILP solver. $MILP^{JH}$ is the formulation used in the implementation of *LocBra* and CPLEX 12.6.0 is the solver. The algorithm is implemented in C language. All tests were executed on a machine with the following configuration: Windows 7 x64, Intel Xeon E5 4 cores and 8 GB of RAM. The optimal solutions were obtained by running CPLEX without time and memory limitations. However, not all optimal solutions were found, because for very large instances CPLEX exploits all the available resources (8 GB RAM) on the machine and halts returning the best solution found. The number of optimal solutions is indicated in the results.

5.3 Accuracy of the heuristics w.r.t. optimal solutions

This experiment focuses on studying the closeness of the heuristics solutions to the optimal solutions. To this end, two databases MUTA [Abu-Aisheh et al., 2015] and PAH [Brun, 2016], out of many, are selected from the public datasets [Abu-Aisheh et al., 2015, Brun, 2016, Moreno-García et al., 2016]. Both databases consist of graphs representing chemical molecules. In MUTA, 8 subsets can be found, the first 7 subsets contain 10 graphs, each of the same size (same number of vertices) starting from 10 until 70 vertices. The last subset has 10 graphs of mixed sizes. This database is interesting because it has large graphs, and they are known to be difficult for matching algorithms. PAH database consists of 94 graphs of different and small sizes (the largest graph has 28 vertices). Each pair of graphs is considered as an instance. Therefore, MUTA database holds a total of 800 instances (100 per subset) and 8836 instances for PAH database. Regarding the cost values assigned to edit operations, they depend on the database and are already defined in their references. For instance, vertices (resp. edges) edit operations are substitution, deletion and insertion and referred to as cvs, cvd, cvi (resp. ces, ced, cei). Based on [Abu-Aisheh et al., 2015], the cost values for MUTA are: $cvs = 5500,\ cvd = cvi = 5500,\ ces = 0,\ ced = cei = 825$. Note that cvs becomes 0 when two vertices have the same attribute values. Moreover, for PAH based on [Brun, 2016]: $cvs = 0,\ cvd = cvi = 3,\ ces = 0,\ ced = cei = 3$. Finally, the heuristics parameter values are set as follows.

For MUTA graphs:

LocBra	$k = 20$, $k_dv = 30$, $total_time_limit = 900s$, $node_time_limit = 180s$, $UB = 180s$
SBPBeam-α	$\alpha = 400$
IPFP-it	$it = 20000$
GNCCP-d	$d = 0.03$

For PAH graphs:

LocBra	$k = 20$, $k_dv = 30$, $total_time_limit = 12.5s$, $node_time_limit = 1.75s$, $UB = 1.75s$
SBPBeam-α	$\alpha = 140$
IPFP-it	$it = 2000$
GNCCP-d	$d = 0.09$

The deviation percentages d_{avg} are computed as explained in sub-section 5.1, 1st experiment paragraph, for each heuristic. The average running time t_{avg} is recorded as well. The number of solutions equal to the optimal ones obtained by one heuristic is represented by η_I. All the optimal solutions are known for PAH database. However, for MUTA and especially for subsets with big graphs, not all optimal solutions are found. The number of optimal solutions known are indicated in the results for each subset. The results are shown in Tables 1 and 2.

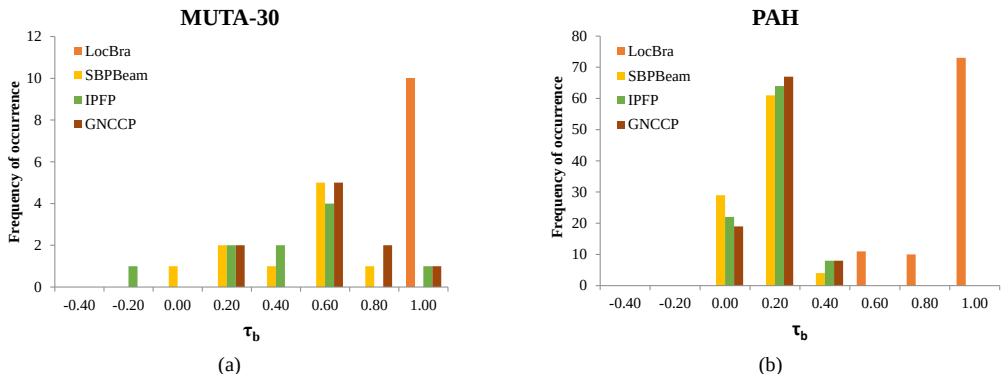


Figure 2: Histograms showing τ_b distribution for each heuristic for MUTA-30 (a) and PAH (b)

Results and analysis The results of PAH instances are reported in Table 1, *LocBra* has the lowest deviation percentage at 0.35%, followed by *GNCCP* with 81%. Clearly, the gap is big between the first and the second heuristic. This is also reflected by the number of solutions obtained by *LocBra* that match the optimal solutions with highest $\eta_I = 8702$, over 8836 instances in total. The fastest is *IPFP* with $1.45s$, but the deviation percentage is very high. Regarding the results of MUTA, which are presented in Table 2, the same conclusion as before on PAH instances can be drawn. *LocBra* has scored the lowest deviation percentages and highest number η_I. For easy instances (subsets 10 to 30), *LocBra* is at 0% deviation. On hard instances (subsets 40 to 70), *LocBra* deviation is always less than 1%. Also, on mixed subset it has the highest $\eta_I = 84$ out of 91 optimal solutions. *IPFP* comes in the second place when considering the deviation percentages for all subsets but subsets 60 and 70, where *GNCCP* appears to have better deviations. Contrary to PAH results regarding the running time, *GNCCP* is the fastest with smaller t_{avg}, however not for all subsets. *LocBra* is faster on subsets 10 and 20, while *IPFP* is faster on subset 70.

These results show that *LocBra* have yielded near-optimal solutions. The rest of the heuristics are outperformed by *LocBra* in terms of solution quality and closeness to the optimal. However, it is still not clear which heuristic is the fastest, because of inconsistent results obtained on the two databases in terms of running time. For small instances, the heuristics have close running time but the difference starts to grow with bigger instances, in favor of *GNCCP* and *IPFP*.

5.4 Accuracy of the heuristics in similarity search

The purpose of this experiment is to examine the correlation between the ranking obtained by the heuristics and the optimal ranking. The MUTA subset with graph size 30 is picked in this experiment, because all optimal solutions are known for these instances (100 instance) and 30 is the average graph sizes. All PAH instances (*8836*) are selected to be part of this experiment as well. The same settings and parameters are used as in the first experiment.

For each database, the Kendall correlation τ_b and p-values are computed for every graph. The p-values are collected and averaged as explained in sub-section 5.1, 2nd experiment paragraph. The

Table 3: Average p-value for each heuristic on MUTA-30 instances

	Average p-value
LocBra	100
SBPBeam	50
IPFP	50
GNCCP	70

Table 4: Average p-value for each heuristic on PAH instances

	Average p-value
LocBra	100
SBPBeam	14
IPFP	21
GNCCP	23

results obtained are reported in Tables 3 and 4. Furthermore, Figure 2 shows the τ_b correlation distribution for both MUTA-30 and PAH.

Results and analysis For MUTA-30 instances in Table 3, the average p-value is 100% for *LocBra*, which means that the null hypothesis is always rejected for all instances. Hence, there is a strong correlation between the ranking of *LocBra* and the optimal ranking. Moreover, *GNCCP* has scored 70%, higher than *SBPBeam* and *IPFP* (both 50%). *GNCCP* should reject the null hypothesis in 70% of the cases. Similar results are noted on PAH instances in Table 4, where *LocBra* is still at 100% and the rest of the heuristics got lower percentages and are far from *LocBra*. Regarding the correlation distribution shown in the histograms in Figure 2; chart (a) shows the distribution for MUTA-30, where all the values of *LocBra* are uniformly in bin 1. This means that *LocBra* ranking is perfectly correlated with the optimal ranking. *GNCCP* comes in the second place but the correlation values are distributed in a wide range between $[0.2, 1]$. *IPFP* shows poor correlation with the optimal and has negative value (-0.2) for one instance. On PAH instances, again the order is conserved: *LocBra*, *GNCCP*, *SBPBeam*, *IPFP*. *LocBra* has correlation values between $[0.6, 1]$, which proves that the heuristic ranking are very similar to the optimal ranking.

The experiment has proved that *LocBra* ranking is strongly correlated with the optimal ranking and therefore, *LocBra* is suitable for GED applications, especially in the contexts of similarity search and graph retrieval. In other words, the use of inaccurate heuristics may lead to really wrong results in terms of nearest neighbors of a graph query.

5.5 Accuracy of the heuristics w.r.t. ground-truth matching

The goal of this experiment is to study the matching accuracy of the heuristics and the relation between the obtained matching and the ground-truth matching, given by the Oracle. The graph database CMU-HOUSE published by [Moreno-García et al., 2016] is considered to this end. It contains 111 graphs corresponding to 3-D images of houses, each graph consists of 30 vertices with attributes described using Shape Context feature vector and the edges are unattributed. The graphs are extracted from 3-D house images, where the houses are rotated with different angles. This is interesting because it enables testing and comparing graphs that represent the same house but positioned differently inside the images. The database also comes with the ground-truth matching for all pairs of graphs. Two versions of the database are considered in the experiment: one without attributes (referred to as HOUSE-NA) on the vertices; another with attributes (referred to as HOUSE-A). It turns out that the first one is harder to solve that the second, because the second considers the attributes, which helps assigning the vertices and edges. Regarding the cost values as defined by [Abu-Aisheh et al., 2015], for HOUSE-NA: $cvs = 0$, $cvd = cvi = \infty$, $ces = 0$, $ced = cei = 0.5$; HOUSE-A: $cvs = 0.5 \times |\mu(u) - \mu'(v)|$, $cvd = cvi = \infty$, $ces = 0$, $ced = cei = 0.5$, with $u \in V$ and $v \in V'$. Finally, the heuristics parameters are set as follows.

For HOUSE-NA graphs:

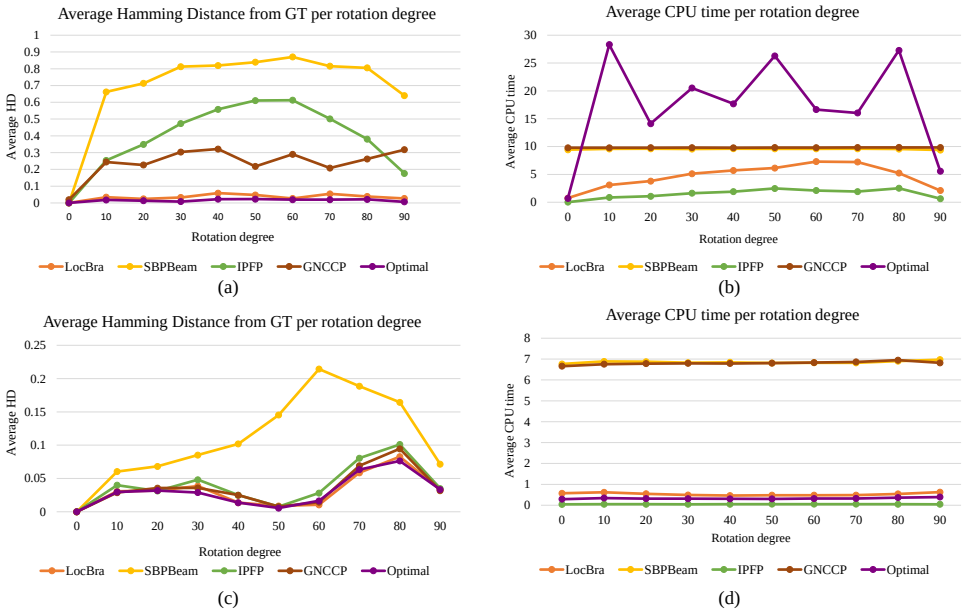


Figure 3: Hamming distance and time averages for heuristics matching vs. ground-truth matching. (a) and (b) are the results for graphs without attributes HOUSE-NA, (c) and (d) with attributes HOUSE-A

LocBra	$k = 20$, $k_dv = 30$, $total_time_limit = 10s$, $node_time_limit = 2s$, $UB = 4s$
SBPBeam-α	$\alpha = 8$
IPFP-it	$it = 400$
GNCCP-d	$d = 0.09$

For HOUSE-A graphs:

LocBra	$k = 20$, $k_dv = 30$, $total_time_limit = 2s$, $node_time_limit = 0.5s$, $UB = 1s$
SBPBeam-α	$\alpha = 5$
IPFP-it	$it = 10$
GNCCP-d	$d = 0.01$

This experiment includes as well the optimal matching computed using the solver CPLEX. The reason behind this, is to evaluate the optimal against the ground-truth matching and detect if they are conformed. This proves the relevance of the edit operations cost values defined for the database. The hamming distances ($\widehat{\mathcal{HD}}$) are computed as explained in sub-section 5.1, 3rd experiment paragraph. Then, the values obtained are grouped by rotation degrees between the graphs being compared. The average values of hamming distance $\widehat{\mathcal{HD}}_{avg}$ obtained are depicted in Figure 3 (a) for HOUSE-NA and (c) for HOUSE-A. In the same figure, (b) and (d) respectively show the average running time variation for each heuristic grouped by rotation degrees as well.

Results and analysis Starting with the $\widehat{\mathcal{HD}}_{avg}$ for HOUSE-NA (Figure 3-(a)), and before analyzing the heuristics behaviors, the *optimal* matching is considered in order to confirm the relation and closeness between ground-truth and *optimal* matching. A strong relation is noted when looking at the *optimal* line, since it is close to 0 for all rotation degrees. Next, the heuristic with the smallest values is *LocBra* for all the rotation degrees. The line is constantly close to 0 and almost linear, which means that *LocBra* has computed matching very close to the ground-truth matching. *GNCCP* and *IPFP* are at the second and third positions after *LocBra*, except for rotated images at 90° *IPFP* outperforms *GNCCP* and $\widehat{\mathcal{HD}}_{avg}$ drops to 0.2. *SBPBeam* comes last with poor $\widehat{\mathcal{HD}}_{avg}$ values. The same conclusion can be seen when looking at the chart (c) (in Figure 3) for HOUSE-A graphs. The heuristics positions are maintained, however an important remark is that the gap between *LocBra*, *IPFP* and *GNCCP* is reduced. Their lines are very close to each other and below 0.1 for all rotation degrees. This is due to the fact that the instances of graphs with attributes are easier to solve, and therefore all the heuristics (except *SBPBeam*)

are able to compute accurate matching and close to the ground-truth matching. Remarkably, the Shape Context features are meaningful and the objective function guides well the exploration of the solution space. Another important point is that the *optimal* has scored always the smallest values except for 60 and 70 degrees, where it is slightly outperformed by *LocBra*. For the average time charts, for HOUSE-NA graphs shown in (b), *IPFP* and *LocBra* are the fastest, while *SBPBeam* and *GNCCP* are very slow and close to each other. This means that *IPFP* and *LocBra* are finding solutions and converging faster than the others. For the *optimal* method, as expected, it is not the fastest because CPLEX spends more time proving the optimality of the solution found. The same is concluded from chart (d) for HOUSE-A graphs.

To sum up, *LocBra* has shown to be very competitive and gave the best accuracy and closeness to the ground-truth matching. Nevertheless, the slightest difference between *LocBra* and ground-truth matching could occur on important vertices of the graphs. It is good then to visualize the matching to see the impact. Such visualization is proposed on the website[1].

6 Conclusion

The present work aims at evaluating the recent and efficient heuristics that deal with the GED problem. It extends the work of [Darwiche et al., 2018] by studying the accuracy and solutions quality of the heuristics. The experiments conducted by [Darwiche et al., 2018] has proved the superiority of LocBra over the heuristics available in the literature as a minimizer to the GED problem. However, these results did not approve the accuracy of the heuristics from an application point of view. This work focuses on executing application-oriented experiments. It points out the need of having an accurate (maybe slower) GED method in graph matching or graph retrieval applications. The conclusions of the experiments are that the quality of the solutions returned by *LocBra* are the closest to the optimal ones, the gap between its ranking and the optimal ranking is negligible as well, leading to a strong correlation with the optimal solutions. The conducted experiments show that optimal solutions of the GED problem are very close to ground truth solutions. This, in turn, proves that *LocBra* is very suitable to be applied in GED applications to perform full (sub-)graph and similarity searches. LocBra is efficient when dealing with complex graphs where neighborhoods and attributes do not allow to easily differentiate between vertices. Therefore, it is suitable for chemical graphs and graphs extracted from images. However, it cannot be generalized unconditionally since some exceptions or untested scenarios may be encountered in cases where graphs are a bit different (very sparse, unconnected vertices, ...). Also, LocBra has a limitation due to $MILP^{JH}$, which can only be applied when edge edit operations have unitary costs. Next, more techniques will be investigated in order to boost the solution and convergence of the method, without degrading the quality and accuracy.

References

[Abu-Aisheh et al., 2015] Abu-Aisheh, Z., Raveaux, R., and Ramel, J. (2015). A graph database repository and performance evaluation metrics for graph edit distance. In *Graph-Based Representations in Pattern Recognition - 10th IAPR-TC-15.Proceedings*, pages 138–147.

[Agresti, 2010] Agresti, A. (2010). *Analysis of ordinal categorical data*, volume 656. John Wiley & Sons.

[Bougleux et al., 2017] Bougleux, S., Brun, L., Carletti, V., Foggia, P., Gaüzère, B., and Vento, M. (2017). Graph edit distance as a quadratic assignment problem. *Pattern Recognition Letters*, 87:38–46.

[Brun, 2016] Brun, L. (2016). Greyc's chemistry dataset.

[Bunke, 1997] Bunke, H. (1997). On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters*, 18(8):689–694.

[Bunke, 1999] Bunke, H. (1999). Error correcting graph matching: On the influence of the underlying cost function. *IEEE transactions on pattern analysis and machine intelligence*, 21(9):917–922.

[Bunke and Allermann, 1983] Bunke, H. and Allermann, G. (1983). Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters*, 1(4):245–253.

[Darwiche et al., 2018] Darwiche, M., Conte, D., Raveaux, R., and T'Kindt, V. (2018). A local branching heuristic for solving a graph edit distance problem. *Computers & Operations Research*.

[Ferrer et al., 2015] Ferrer, M., Serratosa, F., and Riesen, K. (2015). Improving bipartite graph matching by assessing the assignment confidence. *Pattern Recognition Letters*, 65:29–36.

[Fischetti and Lodi, 2003] Fischetti, M. and Lodi, A. (2003). Local branching. *Mathematical programming*, 98(1-3):23–47.

[Justice and Hero, 2006] Justice, D. and Hero, A. (2006). A binary linear programming formulation of the graph edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(8):1200–1214.

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

[Moreno-García et al., 2016] Moreno-García, C. F., Cortés, X., and Serratosa, F. (2016). A graph repository for learning error-tolerant graph matching. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 519–529. Springer.

[Munkres, 1957] Munkres, J. (1957). Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 5(1):32–38.

[Raymond and Willett, 2002] Raymond, J. W. and Willett, P. (2002). Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of computer-aided molecular design*, 16(7):521–533.

[Riesen et al., 2007] Riesen, K., Neuhaus, M., and Bunke, H. (2007). Bipartite graph matching for computing the edit distance of graphs. In *International Workshop on Graph-Based Representations in Pattern Recognition*, pages 1–12. Springer.

[Sanfeliu et al., 2002] Sanfeliu, A., Alquézar, R., Andrade, J., Climent, J., Serratosa, F., and Vergés, J. (2002). Graph-based representations and techniques for image processing and image analysis. *Pattern recognition*, 35(3):639–650.

[Serratosa, 2015] Serratosa, F. (2015). Computation of graph edit distance: reasoning about optimality and speed-up. *Image and Vision Computing*, 40:38–48.

[Stauffer et al., 2017] Stauffer, M., Tschachtli, T., Fischer, A., and Riesen, K. (2017). A survey on applications of bipartite graph edit distance. In *International Workshop on Graph-Based Representations in Pattern Recognition*, pages 242–252. Springer.

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