A local branching heuristic for solving a Graph Edit Distance Problem

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

A Local Branching Heuristic for the Graph Edit Distance Problem

Mostafa Darwiche[a,b], Donatello Conte[a], Romain Raveaux[a], Vincent T'Kindt[b]

[a] *Laboratoire d'Informatique (LI), Université François Rabelais 64 avenue Jean Portalis, 37200 Tours, France* {*mostafa.darwiche,donatello.conte,romain.raveaux*}*@univ-tours.fr*

[b] *Laboratoire d'Informatique (LI), ERL-CNRS 6305, Université François Rabelais 64 avenue Jean Portalis, 37200 Tours, France tkindt@univ-tours.fr*

Abstract

The Graph Edit Distance (GED) problem is a very interesting problem that relates to Graph Matching (GM) problems, wherever a graph models a problem-defined pattern. Solving the GED problem leads to compute the matching between two given graphs by minimizing their GED, which is also called a dissimilarity measure between them. It has many application domains such as structural pattern recognition and chemical Structure-Activity Relationships (SAR). GED^{EnA} (Edges no Attributes) is a sub-problem of the GED problem that deals with a special type of graphs where edges do not carry attributes. They are both NP-hard problems and it is difficult to find the optimal solution in reasonable time. Many heuristics exist in the literature that give suboptimal solutions. Some other works have used mathematical programming tools to model and solve these two problems. The present work takes advantage of a powerful Mixed Integer Linear Program (MILP) formulation and proposes a heuristic called Local Branching to solve the GED^{EnA} problem. Mainly, a MILP model is iteratively modified by adding additional constraints to define neighborhoods in the solution space, which are explored using a black-box solver. A problem-dependent exploration is performed to find efficient solutions. Lastly, the proposed heuristic is evaluated considering two factors: its computational time and solution quality against literature heuristics and exact methods. The computational

experiments reported in this paper show that the proposed local branching heuristic strongly outperforms the literature heuristics for the GED^{EnA} problem.

Keywords:
Graph Edit Distance, Graph Matching, Local Branching Heuristic

---

1. Introduction

Graphs have been widely used to model objects and patterns, especially in fields like pattern recognition, biology and cheminformatics. A graph consists of two sets: vertices and edges. The first set represents a certain number of components that form an object, and the set of edges models the relations between those components. Moreover, vertices and edges can carry information and characteristics via sets of nominal or numerical (or both) values, which are called attributes or labels. In pattern recognition field, graphs are used to represent, for instance, objects in images or videos and also to exploit the relations between them. In chemistry field and precisely when considering the chemical molecules, graphs form a natural representation of the atom-bond structure of molecules. Each vertex of the graph then represents an atom, while an edge represents a molecular bond ([1]). Figure 1 shows an example of graphs modeling objects in images and chemical molecules. After modeling the objects using graphs, an important task is to compare the graphs between each other. This is useful to perform pattern search and classification, object tracking and recognition and many other tasks. In the graph space, graph comparison can be achieved by solving the *Graph Matching* (GM) problems. The solution of a GM problem enables comparing two graphs and find similarities between the objects (modeled by the graphs). GM problems are known to be challenging and hard to solve, therefore in the literature there is a growing interest in finding efficient and fast methods for such problems.

In the literature, GM problems are separated into different categories. Conte et al. ([3]) presented a list of matching problems that basically falls into two main categories: exact isomorphism and error tolerant GM problems. Solving exact isomorphism problems leads to decide if two graphs are identical in terms of structure and attributes on the vertices and edges. In the case of error-tolerant problems, it is intended to find the best matching between two graphs even if they differ in their structure and/or their

Figure 1: Example of graphs modeling objects in images on the left ([2]), and a chemical molecule with its associated graph on the right.

attributes. The second category of problems is more important, especially in the task of finding dissimilarities between unknown and known objects. A well-known problem that belongs to the category of error-tolerant GM problems is the Graph Edit Distance (GED) problem. Solving this problem implies minimizing a dissimilarity measure that stands for the cost needed to transform one graph into another through a series of edit operations ([4]). The available edit operations are substitution, insertion and deletion for vertices or edges, and a cost is associated to each operation. The dissimilarity measure is then defined by the sum of the costs of the edit operations realized. Figure 2 shows an example of two graphs edit operations. In the past years, the GED problem has gained more attention, mainly because it has been shown to generalize other GM problems such as maximum common subgraph, graph and subgraph isomorphism ([5, 6]). Due to its application in domains like cheminformatics, a sub-problem of the GED problem, referred to as GED^{EnA} problem, has recently attracted the attention of researchers. This sub-problem differs from the GED problem by the fact that the input graphs do not have attributes on their edges, implying a null cost for edge substitution operations. Zeng et al. ([7]) have shown that this particular case of the GED problem with unitary costs for deletion and insertion edit operations is NP-hard. This implies by the way that GED^{EnA} and GED problems are so. From the literature it is evinced that, despite its interest, there is a lack of efficient heuristic algorithms dedicated to the GED^{EnA} problem.

Many algorithms can be found that deal with the GED problem and subsequently with GED^{EnA}. They can be split into two categories: exact and heuristic methods. Starting with the exact ones, two Mixed Integer Linear Program (MILP) formulations are proposed by Lerouge et al. ([8]) to solve the GED problem. The first is a direct formulation of the problem,

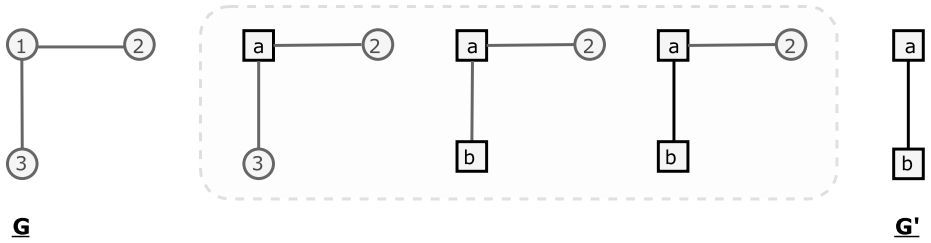

Figure 2: GED process to transform a graph G into graph G'. Edit operations: Substitution of $u_1 \rightarrow v_a$, $u_3 \rightarrow v_b$ and $e_{1,3} \rightarrow f_{a,b}$; deletion of v_2 and $e_{1,2}$

since it consists in minimizing the cost of matching the vertices plus the cost of matching the edges (matching implies either substitution, deletion or insertion). The second MILP is a reformulation of the first MILP, with a reduced number of variables and constraints. A very efficient MILP formulation ($MILP^{JH}$) is introduced by Justice and Hero ([9]), but it is restricted to the GED^{EnA} problem. A quadratic formulation for the GED problem, referred to as Quadratic Assignment Problem (QAP), has been proposed by Bougleux et al. ([10]). The quadratic objective function is the cost of assigning vertices and edges simultaneously. Moving to the heuristic methods, an approach called Bipartite Graph matching (BP) to solve the GED problem is introduced by Riesen et al. ([11]). It consists in solving the Linear Sum Assignment Problem (LSAP) for vertices only, with a cost matrix that includes costs of vertices and edges. Given two graphs G and G', a cost assignment matrix $[C_{ij}]$ is computed: any element C_{ij} corresponds to the cost of the edit operation required to transform vertex u_i from G into v_j from G'. Also, C_{ij} includes an estimation of the cost induced by matching the incident edges. Solving the resulting assignment problem yields a matching of vertices from which a matching of the edges can be deduced. BP is known to be very fast, but it only considers the local structure around vertices, rather than the global one. Later on, many improvements have been proposed as in ([12, 13, 14]), where mainly the cost matrix used is modified and more information about the neighborhood of vertices are considered. Neuhaus et al. ([15]) introduced another heuristic that relies on a beam-search method to solve the GED problem. It mainly builds the search tree for all possible vertex and edge edit operations that can occur, and then only processes a

fixed number (the beam size) of nodes per level of the search tree. An interesting algorithm is presented by Ferrer et al. ([16]) which is basically a combination of BP and beam-search approaches. It solves the LSAP problem to compute an initial solution, and then it tries to improve it by a local search. Two matched vertices are swapped and result in a new solution. The enumeration of the swapped vertices is performed using beam-search. Always in the context of heuristic approaches, Bougleux et al. ([17]) recently have proposed two algorithms called Integer Projected Fixed Point (IPFP) and Graduate Non Convexity and Concavity Procedure (GNCCP). Both use the QAP formulation for the GED problem to compute an initial solution and then iteratively try to improve it by using mathematical transformations and projection methods. All these methods have been designed mainly for the GED problem but can be also applied to the GED^{EnA} problem.

In the present work, a *Local Branching* (LocBra) heuristic is proposed to deal with the GED^{EnA} problem. LocBra was originally introduced by Fischetti et Lodi ([18]), as a general metaheuristic based on MILP formulations. Typically, a local branching heuristic is a local search algorithm, which improves an initial solution by exploring a series of defined neighborhoods via the solution of restricted MILP formulations. The intention of this work is to provide an efficient and dedicated local branching heuristic for the GED^{EnA} problem, which strongly outperforms the available literature heuristics.

The remainder is organized as follows. Section 2 presents the definition of the GED and GED^{EnA} problems and a review of the $MILP^{JH}$ formulation. Section 3 is devoted to a more detailed presentation of the literature's heuristics for the GED problem. Section 4 details the proposed local branching heuristic, while Section 5 reports the results of the intensive computational experiments. Finally, Section 6 highlights some concluding remarks.

2. Definition and modelisation of the GED^{EnA} problem

2.1. The GED^{EnA} problem

To introduce the general GED problem, the definition of the attributed and undirected graph is first given.

Definition 1. *An attributed and undirected 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), with L_V (resp. L_E) the set of all possible attributes for vertices (resp. edges)*

L_V and L_E may contain either numeric or symbolic (or both) attributes.

Next, given two graphs $G = (V, E, \mu, \xi)$ and $G' = (V', E', \mu', \xi')$, solving the GED problem consists in finding the least cost complete edit path that transforms one graph source G into another graph target G'. An edit path is a set of vertices and edges edit operations, which are:

- $u \to v$: substitution of two vertices $u \in V$ and $v \in V'$, i.e. u is matched with v
- $u \to \epsilon$: deletion of vertex $u \in V$
- $\epsilon \to v$: insertion of vertex $v \in V'$
- $e \to f$: substitution of two edges $e \in E$ and $f \in E'$, i.e. e is matched with f
- $e \to \epsilon$: deletion of edge $e \in E$
- $\epsilon \to f$: insertion of edge $f \in E'$

with ϵ refers to the *null* vertex or edge to represent deletion and insertion. Therefore, an edit path is of this form $\lambda(G, G') = \{o_1, ..., o_k\}$, with o_i an elementary vertex or edge edit operation and k is the number of operations. A complete edit path differs from an edit path by some properties and restrictions such as: k is a finite positive number, a vertex/edge can have at most one edit operation applied on it.

Definition 2. *The Graph Edit Distance between two graphs G and G' is defined by:*

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

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 a function that assigns the costs to elementary edit operations e.g. for $u \in V, v \in V'$, $\ell(u \to v) = ||\mu(u) - \mu'(v)||_L$, a defined distance measure between two sets of attributes. Generally, a positive constant is used for deletion and insertion edit operations e.g. $\ell(u \to \epsilon) = \ell(\epsilon \to v) = \tau$, $\forall u \in V, v \in V'$ and $\tau \in \mathbb{R}^+$. Edge operations follow the same logic as vertices for their edit operations costs.

For the GED^{EnA} problem, the graphs are the same as in Definition 1, except that the attribute set for edges is empty ($L_E = \{\phi\}$). Consequently, the costs of edge edit operations are 0 for substitution and a constant for insertion and deletion (i.e. $\ell(e \rightarrow f) = 0$, $\ell(e \rightarrow \epsilon) = \ell(\epsilon \rightarrow f) = \tau$, $\forall e \in E, f \in E'$).

2.2. The $MILP^{JH}$ formulation

$MILP^{JH}$ is a mathematical formulation proposed in ([9]) that deals with the GED^{EnA} 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 ([9]) and only a short description is provided here. The formulation is as follows:

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

such that

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

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

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. Matrices x, s and t are boolean permutation matrices of size $N \times N$, with $N = |V| + |V'|$. x represents the vertices matching i.e. $x_{ij} = 1$ means vertex $u_i \in V \cup \{\epsilon\}$ is matched with vertex $v_j \in V' \cup \{\epsilon\}$. Matrices s and t are for edges matching. Hence, the objective function (Eq. 2) minimizes both, the cost of vertices and edges matching. Constraints 3 guarantee that when matching two couples of vertices, the edges between them have to be matched. Constraints 4 guarantee the integrity of the permutation matrix x.

The $MILP^{JH}$ formulation is the core formulation used in the local branching heuristic proposed in this paper to solve the GED^{EnA} problem. This formulation was shown to be the most efficient for the GED^{EnA} problem among the other formulations found in the literature [19].

3. A review of the literature heuristics

In this section, details about the literature heuristics that solve the GED and GED^{EnA} problems are provided.

BeamSearch is a heuristic for the GED problem presented by Neuhaus et al. ([15]). A beam-search heuristic is an algorithm that explores a truncated search tree to compute a feasible solution to the problem. Given two graphs $G = (V, E, \mu, \xi)$ and $G' = (V', E', \mu', \xi')$, the heuristic first picks a vertex $u \in V$ at the root node, and builds the child nodes corresponding to all possible edit operations on u. For all $v \in V'$, all substitution edit operations are $(u \to v)$, plus one delete operation $(u \to \epsilon)$. This defines the first level of the search tree. Then, only the first α nodes, starting from the left side of the tree, are selected to continue the construction of the search tree, with α the beam size, which is an input parameter to the algorithm. For each of the selected nodes, another vertex $u \in V$ is chosen and the process for creating and selecting child nodes is repeated. Reaching the bottom of the search tree means a complete edit path is built by the way defining a solution. The best solution found is finally returned. This method is known to be very fast because generally the chosen beam size is small.

SBPBeam is a heuristic for the GED problem introduced by Ferrer et al. ([16]). It combines two heuristics: the bipartite (BP) graph matching and the beam search heuristics. The first one has been originally presented in ([11]) and consists in building a special cost matrix for vertices assignment. Given two graphs $G = (V, E, \mu, \xi)$ and $G' = (V', E', \mu', \xi')$, the cost matrix is of size $N \times N$, with $N = |V| + |V'|$, and is divided into four parts. The first one represents the substitution of vertices and is of size $|V| \times |V'|$. The second and third parts are respectively for vertices deletion and insertion and are of size $|V| \times |V|$ and $|V'| \times |V'|$. Only diagonal values represent valid assignments, so the rest is set to a high value (∞) to avoid being selected. Finally, the fourth part is just to complete the matrix and preserve the symmetric form: it is of size $|V'| \times |V|$ and contains only 0 values. The cost matrix is then as follows:

$$\left[\begin{array}{cccc|cccc} c_{11} & c_{12} & \dots & c_{1m} & c_{1\epsilon} & \infty & \dots & \infty \\ c_{21} & c_{22} & \vdots & c_{2m} & \infty & c_{2\epsilon} & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \ddots & \infty \\ c_{n1} & c_{n2} & \dots & c_{nm} & \infty & \dots & \infty & c_{n\epsilon} \\ \hline c_{\epsilon 1} & \infty & \dots & \infty & 0 & 0 & \dots & 0 \\ \infty & c_{\epsilon 2} & \ddots & \vdots & 0 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \infty & \vdots & \ddots & \ddots & 0 \\ \infty & \dots & \infty & c_{\epsilon m} & 0 & \dots & 0 & 0 \end{array}\right] \quad (5)$$

with $n = |V|$ and $m = |V'|$. The cost computation for substitution, deletion and insertion are computed by adding the cost induced by assigning one vertex to another, and the cost of assigning the edges connected directly to both vertices (estimation for edges assignment induced by matching two vertices). The problem becomes a linear sum assignment problem (LSAP), which can be solved in polynomial time. This approach is very fast but it only considers information about the local structure around vertices of the graphs. The SBPBeam heuristic uses a beam-search approach as a local search to improve the initial solution obtained by the BP heuristic. Each level of the search tree corresponds to all possible exchanges of the components (vertices or edges) of two edit operations from the initial solution. At each node the cost of the edit path is computed and the best solution found so far is updated. A selected number of nodes are kept at each level of the search tree, according to the value of the beam size. As the chosen beam size is usually small, in practice, SBPBeam heuristic is very fast.

IPFP is a heuristic that solves the GED problem and has been proposed by Bougleux et al. ([17]). It is based on a heuristic proposed in ([20]) to find a solution to the quadratic assignment problem (QAP). Bougleux et al. model the GED problem as a QAP problem and then propose to apply IPFP heuristic to compute a solution. The idea of IPFP is to try to linearly approximate the quadratic objective function by its 1st-order Taylor's expansion around an initial solution x^0. The quadratic function is derived to obtain a linear function. From this linear function, a new LSAP problem is solved to obtain a solution b, which gives the direction of the largest possible decrease in the quadratic function. Then, the quadratic function consists in minimizing the QAP in the continuous domain along the direction given by b. This is repeated and after some iterations the method converges to a local

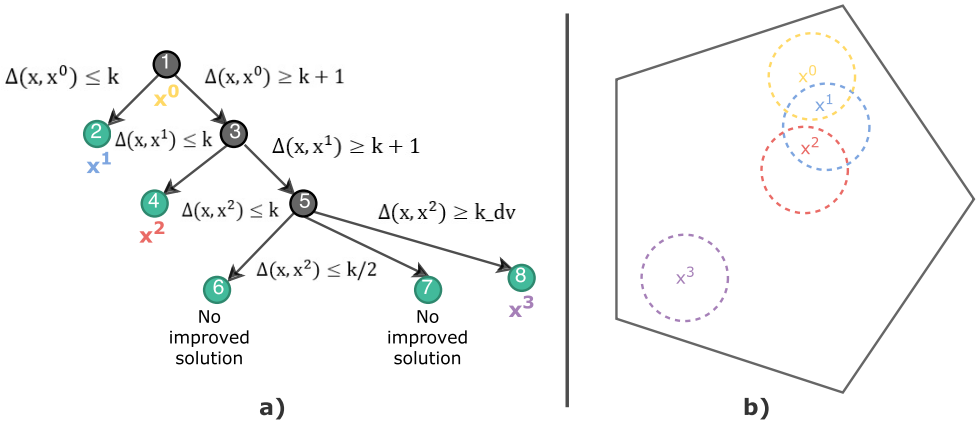

Figure 3: Local branching flow. a) depicts the left and right branching. b) shows the neighborhoods in the solution space

minimum of the relaxed problem. Generally, IPFP converges very fast but the solution quality obtained highly depends on the initial solution x^0, which can be computed using fast heuristics such as BeamSearch or BP heuristics. To limit the computational time, the method has a maximum number of iterations (it) before it stops, in case it has not converged.

GNCCP is introduced by Bougleux et al. in ([17]). It is also a heuristic for the GED problem that works similarly to IPFP by approximating the QAP problem. The quadratic objective function is reformulated as a convex-concave function by introducing a parameter ζ that controls the concavity and convexity of the function. The heuristic decreases iteratively the value of ζ by small positive quantities (e.g. $d = 0.1$) in order to smoothly switch between convex and concave relaxations. Each time, it minimizes the new objective function using IPFP heuristic. GNCCP converges when ζ reaches the value of -1 or when a valid vertices assignment is found at an iteration. More details about IPFP and GNCCP heuristics can be found in ([17]).

4. A Local Branching Heuristic for the GED^{EnA} problem

4.1. Main features of the Local Branching heuristic

This section covers the functionalities and the main features of the local branching heuristic specifically implemented to solve the GED^{EnA} problem. This heuristic version follows the original version introduced by Fischetti and Lodi ([18]), with improvements outlined when appropriate.

Algorithm 1: *LocBra* algorithm

bestUB := UB := ∞; $x^* := \bar{x} := \tilde{x} := undefined$;
tl := elapsed_time := dv := ℓ := dv_cons := 0;
mode_dv := false; opt := false; first_loop := true;
Function `LocBra`($k, k\_dv, total\_time\_limit, node\_time\_limit, dv\_max, \ell\_max, dv\_cons\_max$)
  Output: x^*, opt
  `InitLocBra()`;
  `ImprovedSolution()`;
  elapsed_time := tl;
  while *elapsed_time* < *total_time_limit* **and** *dv* < *dv_max* **and** *dv_cons* < *dv_cons_max* **do**
    tl := min{tl, total_time_limit − elapsed_time};
    status := MIP_SOLVER(tl, UB, $\tilde{x}$);
    tl := node_time_limit;
    if $f(\tilde{x}) = f(\bar{x})$ **and** *mode_dv* = *true* **then** $\ell := \ell + 1$ **else** $\ell := 0$;
    if $\ell \geq \ell\_max$ **then** `Diversification()`; continue;
    if *status* = *"opt_sol_found"* **then**
      if $\tilde{x} \neq \bar{x}$ **then** `ImprovedSolution()` **else** `Diversification()` ;
    end
    if *status* = *"infeasible"* **then** `Diversification()` ;
    if *status* = *"feasible_sol_found"* **then**
      if $f(\tilde{x}) < UB$ **then**
        `ImprovedSolution()`;
      else
        if *mode_dv* = *false* **then** `Intensification()` **else** `Diversification()`;
      end
    end
    elapsed_time := elapsed_time + tl;
  end
End

MILP formulations are very efficient to model hard combinatorial optimization problems. Often, the solution of these formulations is done using a black-box solver, e.g. CPLEX. This kind of exact approach is not capable of solving *real-life instances*, mainly because the size of the instances are big and require a lot of computational time and very large memory size. However, since many years an increasing number of publications have shown the efficiency of heuristics based on the solution of the MILP formulations. This is the case of the local branching method (referred to as LocBra) that is developed here specifically for the GED^{EnA} problem. This method is a local search heuristic, which embeds the truncated solution of $MILP^{JH}$ formulation into a search tree. The basis of LocBra is illustrated in Figure 3. First, LocBra starts with an initial solution x^0, and defines its *k-opt neighborhood* $N(x^0, k)$ (with k a given parameter to the method). The defined neighborhood contains the solutions that are within a distance no more than k from x^0 (in the sense of the *Hamming distance*). This is translated by adding the following *local branching constraint* to the base $MILP^{JH}$ formulation:

$$\Delta(x, x^0) = \sum_{(i,j)\in S^0} (1 - x_{ij}) + \sum_{(i,j)\notin S^0} x_{ij} \leq k \tag{6}$$

with, $S^0 = \{(i,j)/x^0_{ij} = 1\}$. This new model is then solved leading to the search of the best solution in $N(x^0, k)$. This step, which corresponds to node 2 in Figure 3-a, is referred to as the intensification phase. If a new solution x^1 is found, the constraint (Eq. 6) is replaced by $\Delta(x, x^0) \geq k + 1$, and the right branch emanating from node 1 is explored. This guarantees that an already visited solution space will not be visited again. Next, a left branch is created but now using the solution x^1: the neighborhood $N(x^1, k)$ is explored by solving the $MILP^{JH}$ formulation with the constraint $\Delta(x, x^1) \leq k$ added (node 4 in Figure 3-a). Then, the process is repeated until a stopping criterion is met, e.g. a *total time limit* is reached. Note that solving the sub-problems (with local branching constraints) may not be possible in a reasonable amount of time, because they can be also hard to solve. For this reason, a *node time limit* is imposed when solving the sub-problems. Therefore, during the exploration of the neighborhood $N(x^i, k)$ of the solution x^i, it may occur that no improving solution is found or even the solver is not able to find a feasible solution during the node time limit. For instance, assuming at node 6 in Figure 3-a, the solution of the $MILP^{JH}$ plus equation $\Delta(x, x^2) \leq k$ does not lead to a feasible solution in the given time limit. Then,

in LocBra a complementary intensification step is applied, by replacing the last constraint on x^2 by $\Delta(x, x^2) \leq k/2$ and solving the new sub-problem: this results in the exploration of a reduced neighborhood around x^2. If again, no feasible solution is found (node 7 in Figure 3-a), then a diversification step is applied to jump to another point in the solution space. Since it is a more complex and problem-dependent mechanism than the original one proposed by Fischetti and Lodi, its description is then given in section 4.2. Figure 3-b shows the evolution of the solution search and the neighborhoods.

4.2. Problem-dependent features of the Local Branching heuristic

The efficiency of the LocBra heuristic for solving the GED^{EnA} problem has been improved by adapting certain mechanisms of the method.

The first particularization relates to the choice of the variables for defining the search space. Traditionally, in a local branching heuristic all boolean variables are considered to define the local branching constraint $\Delta(x, x^i) \leq k$. However, for the GED^{EnA} problem it turns out that the crucial variables are the x_{ij}'s, which model the vertices matching. Other sets of variables (s_{ij} and t_{ij}) in $MILP^{JH}$, which correspond to edge matching, can be easily fixed by the solver as soon as the vertices are matched. Letting LocBra explores the solution space only on the basis of the x_{ij} variables, leads to the consideration of a smaller number of variables in the local branching constraint. By the way, this strengthens the local search by avoiding fast convergence towards local optima. Consequently, the local branching constraint is:

$$\Delta(x, x^p) = \sum_{(i,j) \in S^p} (1 - x_{ij}) + \sum_{(i,j) \notin S^p} x_{ij} \leq k \tag{7}$$

with, $S^p = \{(i,j)/x^p_{ij} = 1\}$.

Another important improvement is proposed for the diversification mechanism, where also not all binary variables are included but a smaller set of *important* variables is used instead. Note that, again only x_{ij} variables that represent vertices matching are considered. The diversification constraint is then

$$\Delta'(x, x^p) = \sum_{(i,j) \in S^p_{imp}} (1 - x_{ij}) + \sum_{(i,j) \in B_{imp} \setminus S^p_{imp}} x_{ij} \geq k\_dv \tag{8}$$

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 variable is based on the idea that when changing its value from $1 \rightarrow 0$ (or the opposite), it highly impacts

the objective function value. Forcing the solver to modify such variables enables escaping from local optima and changes the macthing. Accordingly, B_{imp} is obtained by computing a special cost matrix $[M_{ij}]$ for each possible assignment of a vertex $u_i \in V \cup \{\epsilon\}$, to a vertex $v_j \in V' \cup \{\epsilon\}$.

$$M = \begin{bmatrix} c_{11}+\theta_{11} & c_{12}+\theta_{12} & \dots & c_{1|V'|}+\theta_{1|V'|} & c_{1\epsilon}+\theta 1\epsilon \\ c_{21}+\theta_{21} & c_{22}+\theta_{22} & \vdots & c_{2|V'|}+\theta_{2|V'|} & c_{2\epsilon}+\theta 2\epsilon \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ c_{|V|1}+\theta_{|V|1} & c_{|V|2}+\theta_{|V|2} & \dots & c_{|V||V'|}+\theta_{|V||V'|} & c_{|V|\epsilon}+\theta|V|\epsilon \\ c_{\epsilon 1}+\theta\epsilon 1 & c_{\epsilon 2}+\theta\epsilon 2 & \dots & c_{\epsilon|V'|}+\theta\epsilon|V'| & 0 \end{bmatrix} \quad (9)$$

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 ([21]) which requires $O(max(|E_i|, |E_j|)^3)$ time. Next, the standard deviation is computed at each row of the matrix $[M_{ij}]$, resulting in a vector $\sigma = [\sigma_1, ..., \sigma_{|V|}]$. Typically, a high value of σ_i means that the contribution to the objective function of the matching of vertex $u_i \in V \cup \{\epsilon\}$ with a vertex $v_j \in V'$ strongly varies depending on v_j. Such variables are considered as important. To isolate the ones with the highest σ_i values, a simple clustering algorithm is applied. Two clusters C_{min} and C_{max} are built by starting with the minimum σ_{min} and maximum σ_{max} values as the centers of the clusters. $\forall u_i \in V \cup \{\epsilon\}$ if $|\sigma_i - avg_{C_{min}}| < |\sigma_i - avg_{C_{max}}|$ then $\sigma_i \rightarrow C_{min}$, otherwise $\sigma_i \rightarrow C_{max}$, with $avg_{C_{min}}$ and $avg_{C_{max}}$ the averages of the selected values in the clusters. Every time a value σ_i is added to C_{min} or C_{max}, its average value $avg_{C_{min}}$ or $avg_{C_{max}}$ is updated. Finally, for every σ_i belonging to C_{max} cluster, the indexes of all the binary variables x_{ij} that correspond to the assignment of vertex u_i are added to B_{imp}. Finally, note that k_{dv} used in the diversification constraint is chosen to be greater than k (in constraint 7), in order to guarantee better diversification.

Preliminary experiments, not reported here, have shown that such a diversification significantly improves the local branching heuristic, better than the original one introduced by Fischetti and Lodi ([18]), which was quite inefficient for escaping local optima. Regardless of the quality of the new solution (whether better or worse than the current best solution), what is important is that the diversification mechanism succeeds in diversifying the search. Then, the intensification steps afterwards lead to a deep exploration

of the solution space around the new solution.

4.3. The Local Branching algorithm

A detailed algorithmic presentation of LocBra heuristic is provided in Algorithm 1, and the details of the functions used are given in Appendix A. The core function of the heuristic takes the following parameters as input:

1. k, is the neighborhood size.

2. k_dv, is for diversification to guarantee that the next solution to be found is far from the current one by at least k_dv changes of binary variables.

3. *total_time_limit*, is the total running time allowed for LocBra before stopping.

4. *node_time_limit*, is the maximum running time given to the solver at any node to solve the $MILP^{JH}$ formulation.

5. ℓ_max, is used to force a diversification step after a sequence of ℓ_max intensification steps returning solutions with the same objective function value. This parameter avoids spending a lot of time searching in a region where no improving solutions are found.

6. dv_max, is the maximum number of diversification steps allowed during the execution of LocBra. The rational for such a parameter comes from preliminary experiments, which have shown that first diversification steps are useful to reach very good solutions. Then, this parameter enables to decrease the global execution time without losing in the quality of the returned solution by LocBra.

7. dv_cons_max, serves as a stopping criterion, when consecutive diversification steps have returned solutions with the same value of the objective function, then the heuristic stops. When this situation occurs, then the diversification mechanisms is inefficient in allowing the heuristic to escape from the current local optima.

From the above list of parameters, the stopping criterion of the proposed LocBra heuristic does not only rely on the total time spent: the algorithm stops whenever one of these three conditions is met:

(i) the total execution time exceeds the *total_time_limit*,

(ii) the number of diversification steps done during the search exceeds *dv_max*,

(iii) the number of consecutive diversification steps done exceeds *dv_cons_max*.

The output of the algorithm is the best solution found (x^*) and a flag to indicate whether it has been proved to be optimal or not (*opt*). The initial solution x^0 used by LocBra is obtained by solving $MILP^{JH}$ formulation within a time limitation of *node_time_limit* seconds. First, it calls *InitLocBra* function that initializes the heuristic by computing an initial solution $\tilde{x}$ (it is the first solution x^0 as introduced in Section 4.1). If at this point, $MILP^{JH}$ is solved to optimality or no feasible solution has been found, the heuristic halts and returns the available solution and/or the status. Otherwise, the current solution $\bar{x}$ is set to the solution found and the exploration begins. Lines 2 to 23 present the core of the heuristic as previously described. At each iteration and after a left/right branching constraint is added, the solver is called through *MIP_Solver(tl,UB,$\tilde{x}$)* function, and the returned status is considered to make the next decision. Note that, *tl* variable corresponds to the time limit imposed when solving $MILP^{JH}$, $\tilde{x}$ and UB are, respectively, the solution computed by the solver (new solution) and its objective function value. Three possible statuses may occur:

(i) **Optimal solution** is found at a branch, and then two cases must be distinguished (line 11). If the new solution $\tilde{x}$ is better than the current solution $\bar{x}$, then *ImprovedSolution* is called to update the current and best solutions (if needed), and to define a new neighborhood by adding the constraint Eq. 7 using the new solution $\tilde{x}$. If the new solution $\tilde{x}$ and the last solution $\bar{x}$ are equal, i.e. $\tilde{x} = \bar{x}$, then *Diversification* is called to skip the current neighborhood and search in a different region in the search space. *Diversification* function ensures that the current solution is skipped with a distance *k_dv*, and the upper bound UB is reset to ∞ to allow finding a new solution even if it is worse than the best known one.

(ii) The model is **infeasible** (line 14). Therefore *Diversification* is triggered to switch the last local branching constraint and look into a new neighborhood in the search space.

	CPLEX-∞	*LocBra*
t_{min}	0.09	0.06
t_{avg}	**2.08**	3.03
t_{max}	278.20	12.25
d_{min}	-	0.00
d_{avg}	-	**0.35**
d_{max}	-	100.00
η_I	8836	6715
η'_I	-	**8702**
η''_I	-	0

Table 1: LocBra vs. Exact solution on PAH instances

	CPLEX-∞ (4 threads)				*LocBra (1 thread)*									*LocBra (4 threads)*								
S	t_{min}	t_{avg}	t_{max}	η_I	t_{min}	t_{avg}	t_{max}	d_{min}	d_{avg}	d_{max}	η_I	η'_I	η''_I	t_{min}	t_{avg}	t_{max}	d_{min}	d_{avg}	d_{max}	η_I	η'_I	η''_I
10	0.07	**0.12**	0.32	100	0.06	0.17	2.92	0.00	**0.00**	0.00	100	**100**	0	0.07	0.16	0.48	0.00	**0.00**	0.00	**100**	**100**	0
20	0.15	**0.95**	19.74	100	0.13	1.12	3.63	0.00	**0.00**	0.00	100	**100**	0	0.14	1.00	21.80	0.00	**0.00**	0.00	**100**	**100**	0
30	0.31	101.53	2865.24	100	0.28	212.36	900.13	0.00	**0.00**	0.00	78	**100**	0	0.32	**101.33**	900.10	0.00	**0.00**	0.00	**91**	**100**	0
40	0.52	266.00	9243.72	99	0.45	364.86	900.12	0.00	0.06	3.90	63	98	0	0.49	**179.45**	900.13	0.00	**0.00**	0.00	**84**	**100**	0
50	0.83	682.71	4212.68	92	0.69	580.04	900.17	-1.79	0.04	4.14	37	97	**1**	0.73	**435.16**	900.32	-1.79	**0.00**	2.07	**54**	**98**	**1**
60	1.24	2419.33	14732.35	71	0.95	753.48	900.27	-2.68	0.36	3.57	16	82	2	1.09	**718.25**	901.66	-3.31	**-0.03**	3.21	**21**	**90**	**6**
70	1.80	3740.34	24185.25	35	1.36	751.44	900.36	-2.67	0.78	8.85	17	52	14	1.48	**741.52**	901.35	-3.90	**0.22**	3.65	**18**	**60**	**16**
Mixed	0.09	1613.41	17084.43	91	0.14	332.92	902.25	-2.67	**0.03**	3.43	64	88	**5**	0.09	**324.17**	900.27	-1.35	0.05	1.87	**66**	**92**	2

Table 2: LocBra vs. Exact solution on MUTA instances

(iii) A **feasible solution** is returned (line 15). This is very close to the first case, except when a worse solution is found, i.e. $f(\tilde{x}) > UB$. An additional *Intensification* step is done but within a neighborhood limited to $k/2$ variable changes from $\bar{x}$ in order to try to improve it. However, if the solver fails again, then a *Diversification* step is performed.

In addition, there is the condition (at line 10) that forces the diversification step, in the case where ℓ_max consecutive intensification iterations have returned solutions with the same objective function value. This in turn guarantees the exploration of many neighborhoods in different regions of the solution space.

	LocBra	*CPLEX-12.48*	*CPLEX_LocBra-3.5*	*BeamSearch-5*	*SBPBeam-5*	*IPFP-10*	*GNCCP-0.1*
t_{min}	0.06	0.05	0.05	0.00	0.01	0.00	0.17
t_{avg}	3.03	1.97	1.79	**0.01**	0.14	0.03	2.08
t_{max}	12.25	12.48	6.41	0.03	0.37	0.08	6.02
d_{min}	0.00	0.00	0.00	0.00	0.00	0.00	0.00
d_{avg}	0.31	**0.05**	0.91	122.65	379.90	127.20	84.43
d_{max}	75.00	190.91	200.00	2400.00	4200.00	2400.00	1000.00
η_I	8716	**8830**	8553	433	100	450	1042

Table 3: LocBra vs. literature heuristics on PAH instances

	S	10	20	30	40	50	60	70	Mixed
LocBra	t_{min}	0.06	0.13	0.28	0.45	0.69	0.95	1.36	0.14
	t_{avg}	0.17	1.12	212.36	364.86	580.04	753.48	751.44	332.32
	t_{max}	2.92	3.63	900.13	900.12	900.17	900.27	900.36	902.25
	d_{min}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	d_{avg}	**0.00**	**0.00**	**0.00**	0.06	**0.02**	**0.17**	**0.59**	**0.04**
	d_{max}	0.00	0.00	0.00	3.90	2.03	3.35	5.57	1.77
	η_I	**100**	**100**	**100**	98	**99**	**93**	**79**	**95**
CPLEX-900	t_{min}	0.06	0.14	0.28	0.49	0.77	1.18	1.70	0.09
	t_{avg}	0.13	1.02	141.07	247.80	451.40	723.68	745.91	305.72
	t_{max}	0.49	3.52	900.20	900.42	900.46	900.71	900.92	900.70
	d_{min}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	d_{avg}	**0.00**	**0.00**	**0.00**	**0.00**	0.30	0.55	1.05	0.12
	d_{max}	0.00	0.00	0.00	0.00	6.42	5.04	8.57	5.49
	η_I	**100**	**100**	**100**	**100**	90	81	68	**95**
CPLEX_LocBra-180	t_{min}	0.09	0.22	0.41	0.73	1.03	1.45	1.98	0.14
	t_{avg}	0.21	1.51	60.36	104.19	141.43	167.59	181.18	86.32
	t_{max}	0.74	5.77	182.86	194.08	195.43	217.38	263.60	223.53
	d_{min}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	d_{avg}	**0.00**	**0.00**	**0.00**	0.16	1.16	1.41	4.24	0.36
	d_{max}	0.00	0.00	0.00	3.90	7.19	6.70	27.20	6.86
	η_I	**100**	**100**	**100**	94	72	57	41	82
BeamSearch-5	t_{min}	0.00	0.00	0.01	0.01	0.02	0.04	0.06	0.01
	t_{avg}	**0.00**	**0.00**	**0.01**	**0.03**	**0.07**	**0.11**	**0.18**	**0.09**
	t_{max}	0.07	0.02	0.04	0.11	0.09	0.13	0.22	0.21
	d_{min}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	d_{avg}	15.17	36.60	47.21	58.69	72.13	62.96	68.71	21.20
	d_{max}	110.00	124.59	147.37	186.67	200.00	146.37	210.71	112.71
	η_I	35	10	10	10	10	10	10	12
SBPBeam-5	t_{min}	0.01	0.08	0.31	1.11	2.69	4.87	9.02	0.05
	t_{avg}	0.01	0.10	0.45	1.37	3.19	5.56	10.72	3.38
	t_{max}	0.05	0.14	0.54	1.60	3.71	6.85	12.79	12.05
	d_{min}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	d_{avg}	20.43	44.90	76.45	82.54	98.90	95.02	94.62	27.16
	d_{max}	90.00	127.87	206.90	204.71	314.29	198.50	280.36	135.91
	η_I	15	10	10	10	10	10	10	10
IPFP-10	t_{min}	0.00	0.01	0.02	0.03	0.06	0.10	0.15	0.01
	t_{avg}	0.01	0.06	0.20	0.30	0.39	0.66	1.05	0.46
	t_{max}	0.08	0.20	0.35	0.59	0.56	1.01	1.49	1.39
	d_{min}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	d_{avg}	3.44	10.84	18.31	21.34	22.59	25.9	27.63	7.45
	d_{max}	30.00	80.77	90.41	93.33	66.67	66.67	99.08	49.72
	η_I	69	28	14	11	10	10	10	19
GNCCP-0.1	t_{min}	0.02	0.12	0.38	0.89	1.68	2.88	4.59	0.15
	t_{avg}	0.16	1.30	4.77	11.78	22.08	72.29	111.30	28.53
	t_{max}	0.29	2.52	10.86	31.58	73.46	145.53	255.88	218.99
	d_{min}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	d_{avg}	13.29	22.00	26.93	21.69	31.46	21.99	27.61	10.66
	d_{max}	411.43	400.00	188.79	119.12	205.36	205.77	101.79	125.16
	η_I	57	35	6	6	5	9	4	17

Table 4: LocBra vs. literature heuristics on MUTA instances

	S	10	20	30	40	50	60	70	Mixed
LocBra	t_{min}	0.06	0.13	0.28	0.45	0.69	0.95	1.36	0.14
	t_{avg}	**0.17**	**1.12**	212.36	364.86	580.04	753.48	751.44	332.32
	t_{max}	2.92	3.63	900.13	900.12	900.17	900.27	900.36	902.25
	d_{min}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	d_{avg}	**0.00**	**0.00**	**0.00**	0.02	**0.02**	**0.13**	**0.54**	**0.04**
	d_{max}	0.00	0.00	0.00	1.69	2.03	2.94	5.57	1.77
	η_I	**100**	**100**	**100**	99	**99**	**94**	**80**	**97**
CPLEX_LocBra-800	t_{min}	0.08	0.21	0.38	0.67	1.01	1.40	1.94	0.20
	t_{avg}	0.20	1.34	130.26	230.68	424.70	662.58	**688.13**	291.64
	t_{max}	0.71	3.90	802.16	806.16	821.39	839.69	869.65	849.58
	d_{min}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	d_{avg}	**0.00**	**0.00**	**0.00**	**0.00**	0.38	0.57	1.02	0.13
	d_{max}	0.00	0.00	0.00	0.00	6.42	5.04	11.27	5.49
	η_I	**100**	**100**	**100**	**100**	89	81	71	95
BeamSearch-15000	t_{min}	0.00	0.00	0.03	0.10	0.55	0.24	2.28	0.03
	t_{avg}	8.57	80.65	167.48	279.11	439.68	640.29	938.66	828.52
	t_{max}	31.52	118.71	230.63	419.73	771.90	878.89	1385.11	1800.00
	d_{min}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-
	d_{avg}	1.35	26.66	47.45	52.29	63.98	62.51	63.71	-
	d_{max}	30.00	142.31	165.52	180.00	150.00	157.63	226.79	-
	η_I	88	12	10	10	10	10	10	-
SBPBeam-400	t_{min}	0.76	9.02	39.85	116.11	288.38	548.04	1019	1.98
	t_{avg}	0.84	10.02	47.65	139.75	322.43	590.86	1155	326.64
	t_{max}	0.96	11.27	54.11	152.34	360.47	657.26	1310	1225.92
	d_{min}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	d_{avg}	20.43	44.90	76.45	82.45	98.90	94.94	94.54	26.95
	d_{max}	90.00	127.87	206.90	204.71	314.29	198.50	280.36	135.91
	η_I	15	10	10	10	10	10	10	10
IPFP-20000	t_{min}	0.00	0.01	0.02	0.3	0.11	0.10	0.18	0.01
	t_{avg}	1.20	9.62	48.90	115.14	240.54	528.82	903	303.21
	t_{max}	8.52	53.83	165.44	456.93	771.64	1620	2839	1827
	d_{min}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	d_{avg}	3.44	10.18	16.45	17.17	19.00	18.99	20.70	6.03
	d_{max}	30.00	80.77	90.41	56.47	47.62	50.53	85.71	38.03
	η_I	69	29	14	11	10	10	10	21
GNCCP-0.03	t_{min}	0.03	0.18	0.58	1.26	2.44	4.33	6.65	0.25
	t_{avg}	0.55	6.41	**29.80**	**81.24**	**195.89**	**396.37**	946.25	**185.55**
	t_{max}	1.13	16.81	71.94	167.06	450.41	797.39	2330	1398.72
	d_{min}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	d_{avg}	3.23	6.67	17.20	15.74	18.38	16.12	18.17	5.13
	d_{max}	90.00	30.7 7	82.76	57.35	95.24	52.29	77.06	25.35
	η_I	81	34	4	6	5	9	5	17

Table 5: LocBra vs. literature heuristics with extended running time on MUTA instances

5. Computational Experiments

This section provides the computational experiment done to evaluate the efficiency of the proposed LocBra heuristic, w.r.t. both exact and heuristic methods from the literature.

5.1. Instances and experimentation settings

As stated when introducing the GED^{EnA} problem, it has numerous applications in Pattern Recognition. This has led along the years to the creation of relevant databases of instances related to particular applications. They also serve as recognized basis for comparing matching algorithms. To conduct the experiments, instances from MUTA ([22]) and PAH ([23]) databases are selected. These two databases are chosen after reviewing the public datasets ([22, 23]), which are usually used to evaluate the GED methods. MUTA and PAH databases contain graphs that represent chemical molecules. MUTA is divided into 8 subsets, 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 (sizes $50, 60, 70$), and they are known to be difficult for matching algorithms. PAH database consists of 94 graphs of various and small sizes (the largest graph has 28 vertices). Each pair of graphs is considered as an instance for the GED^{EnA} problem. Therefore, MUTA database holds a total of 800 instances (100 per subset) and 8836 instances for PAH database.

LocBra algorithm is implemented in C language. The solver CPLEX 12.6.0 is used to solve the MILP formulations. Experiments are ran on a machine Intel Core $i4$ with 8 GB RAM. When solving a MILP formulation within LocBra heuristic, CPLEX solver is parametrized to use a single thread even if 4 *cores* are available. The aim of this in the experiments is to evaluate the efficiency of the inner mechanism of LocBra. It can be then expected that its efficiency is going to be improved by enabling the use of more threads. Two experiments are conducted for each database. The first consists in evaluating the quality of the solutions obtained by LocBra w.r.t. to optimal solutions. The second consists in studying the efficiency of LocBra in comparison to the literature heuristics.

5.2. Comparison of LocBra with the exact approach

In this experiment, LocBra is compared with an exact approach in order to evaluate the quality of its solutions against the optimal ones. The

exact approach consists in solving the $MILP^{JH}$ formulation using CPLEX without time and resources limitations, in order to get the optimal solution. Note that, here the default settings of the solver are used, this means the solver uses 4 threads. However, even without imposing restrictions and limitations on the resources of CPLEX, it can occur that CPLEX may not be able, on the large instances of MUTA database, to compute the optimal solution after several hours. In this case and when CPLEX is stopped, the best solution found is returned. The exact approach is referred to as *CPLEX*-∞. The following metrics are computed for each database: $t_{min}, t_{avg}, t_{max}$, which are respectively the minimum, average and maximum CPU times in seconds for all instances. Correspondingly, $d_{min}, d_{avg}, d_{max}$ are the deviations (in percentage) of the solutions obtained by LocBra, from the optimal or best solutions found. Given an instance I, the deviation is equal to $\frac{solution_I^H - optimalOrBest_I}{optimalOrBest_I} \times 100$, with $solution_I^H$ the solution value returned by LocBra and $optimalOrBest_I$ is the solution value obtained by *CPLEX*-∞ for I. In addition, η_I is the number of optimal solutions found, and η'_I is the number of solutions found by LocBra that are equal to the optimal or best known ones. At last, η''_I is the number of solutions computed by LocBra that are better than the best known solutions, where *CPLEX*-∞ was not able to prove their optimality.

PAH database: LocBra parameters are set to the following values: $k = 20$, $k\_dv = 30$, $total\_time\_limit = 12.25s$, $node\_time\_limit = 1.75s$, $dv\_max = 5$, $l\_max = 3$, $dv\_cons\_max = 2$. Table 1 shows the obtained results. The optimal solutions are computed for all instances by *CPLEX*-∞ ($\eta_I = 8836$). LocBra has found the optimal solutions for 8702 instances. The average deviation of LocBra to the optimal solution is of 0.35%, which is small. When looking at the 134 instances (out of 8836), for which LocBra failed to find the optimal solution, the average deviation restricted to these instances is about 23%. Nonetheless, it can be concluded that LocBra provides very good solutions on PAH instances. For the running time, *CPLEX*-∞ is on the average faster than LocBra but in the worst case CPLEX becomes computationally expensive (up to $278.20s$), while the heuristic remains at maximum below than $13s$. This experiment clearly shows that PAH instances are easy ones for the $MILP^{JH}$ formulation. However, they remain standard instances for algorithms comparison in the Pattern Recognition community.

MUTA database: These instances are harder than PAH instances. To this end, two versions of LocBra are included in this experiment: the first

one is with one thread used by CPLEX to solve the MILP formulation, and the second one with 4 threads. The aim is to evaluate the gain for LocBra heuristic when increasing the calculation capacity of CPLEX solver. LocBra parameters are set to the following values: $k = 20$, $k\_dv = 30$, $total\_time\_limit = 900s$, $node\_time\_limit = 180s$, $dv\_max = 5$, $l\_max = 3$, $dv\_cons\_max = 2$. The results on MUTA instances are reported in Table 2. All optimal solutions are found by *CPLEX-∞ (4 threads)* for easy instances (subsets 10 to 40), except one, and both LocBra with 1 and 4 threads have $d_{avg} = 0\%$ (except for 2 instances in subset 40). Clearly both versions have returned the same optimal or best solutions. For hard instances (subsets 50 to 70), d_{avg} is always less than 1% and even less than 0% (-0.03%) for the version with 4 threads on subset 60. Note that, a negative deviation implies that for some instances *CPLEX-∞* was not able to find the optimal solution and that LocBra provided a better solution. The values of η_I reveal that this case occurred. η''_I for hard instances reveals that the heuristics have outperformed *CPLEX-∞* and found improved solutions (better than the best ones obtained) for 17 instances with 1 thread and 23 instances with 4 threads. Considering the running time, it drastically increases for *CPLEX-∞* and reaches thousands ($t_{avg} = 3740s$), while the proposed heuristic has a maximum of $t_{avg} = 751s$. It is the same conclusion for subset Mixed, both LocBra versions are faster in terms of average CPU time, and also they have very low average deviations. The first conclusion from the experiments on MUTA instances is that the local branching mechanism as implemented is very efficient in reaching optimal or near-optimal solutions. The second conclusion relies on the number of threads used within LocBra heuristic: even if, as expected, increasing the number of used threads in CPLEX leads to an improvement of the heuristic, this one is reduced enough to render this improvement marginal.

In the case of easy instances (small graphs) as in PAH database and subsets 10 to 40 in MUTA database, CPLEX is very efficient in solving $MILP^{JH}$ and obtaining the optimal solutions. On the other hand, LocBra results prove also its efficiency in obtaining the near-optimal solutions. One advantage for LocBra is that it is faster in the worst case because of the time limit imposed. Regarding the hard instances of MUTA (subsets $50, 60, 70$), where *CPLEX-∞* is not able to compute all the optimal solutions, LocBra performs better than CPLEX and yield better solutions for some instances in the limited time of $900s$.

5.3. Comparison of LocBra with the literature heuristics

LocBra heuristic is tested against the following heuristics: **i-** *CPLEX-t* is the solver CPLEX ran on $MILP^{JH}$ with t seconds as time limit. **ii-** *CPLEX-LocBra-t* refers to enabling local branching heuristic implemented in CPLEX solver. Note, that in the default settings, this heuristic is disabled. So, *CPLEX-LocBra-t* is the heuristic implemented under CPLEX with a time limit of t seconds. The time limit is imposed in order to compute an initial solution, which will be given to CPLEX after to run local branching with that solution. **iii-** *BeamSearch-α* ([15]), with α the beam size. **iv-** *SBPBeam-α* ([16]), with α the beam size. **v-** *IPFP-it* ([17]), with it the maximum number of iterations. **vi-** *GNCCP-d* ([17]), with d the quantity to be deducted from the ζ variable at each iteration. ζ is the variable that controls the concavity and convexity of the objective function of the QAP model. All these methods are executed on PAH and MUTA instances and the following metrics are computed: $t_{min}, t_{avg}, t_{max}$ are the minimum, average and maximum CPU times in seconds for all instances. Correspondingly, $d_{min}, d_{avg}, d_{max}$ are the deviation for the solutions obtained by one heuristic, from the best solutions found. The deviations are expressed in percentage. Given an instance I and a heuristic H, the deviation is equal to $\frac{solution_I^H - bestSolution_I}{bestSolution_I} \times 100$, with $bestSolution_I$ the smallest solution value found by all heuristics for I. Lastly, η_I is the number of instances for which a given heuristic has found the best solutions.

PAH database: First the parameters values for each method are discussed. For LocBra, *CPLEX-t* and *CPLEX-LocBra-t* heuristics, the parameters are set following to preliminary experiments, not reported here. In this experiment, the aim is to find the parameters leading to the most efficient heuristic in terms of quality of the computed solution within a short CPU time. For *BeamSearch-α* and *SBPBeam-α*, the heuristics values α are taken from the paper that originally presented the methods ([15, 16]). The same holds for *IPFP-it* and *GNCCP-d* heuristics, where the parameters values are extracted from in [17].

LocBra	$k = 20$, $k\_dv = 30$, $total\_time\_limit = 12.25s$, $node\_time\_limit = 1.75s$, $dv\_max = 5$, $l\_max = 3$, $dv\_cons\_max = 2$
CPLEX-t	$t = 12.48$
CPLEX_LocBra-t	$t = 3.5$
BeamSearch-α	$\alpha = 5$
SBPBeam-α	$\alpha = 5$
IPFP-it	$it = 10$
GNCCP-d	$d = 0.1$

The results are shown in Table 3. *CPLEX-12.48* has an average deviation of 0.05% which is the smallest among all the heuristics. Next *LocBra* comes with 0.31%. Consequently, *CPLEX-12.48* has performed better than the proposed heuristic. However, an important note is that d_{max} for *LocBra* is 75% against 190.91% for *CPLEX-12.48*, which means that the former provides the closest solutions to the best ones in the worst case. *CPLEX_LocBra-3.5* comes at the third position, with an average deviation less than 1%. The beam-search based heuristics, *IPFP-10* and *GNCCP-0.1* are strongly outperformed by the other MILP-based heuristics with a high average deviations. In terms of CPU time, the beam-search based heuristics seems to be very fast ($t_{avg} < 1s$), while the proposed heuristic is the slowest with $t_{avg} = 3.03s$.

MUTA database: The following are the values of the parameters set for each method. As in PAH experiment, the values are set for LocBra, *CPLEX-t* and *CPLEX-LocBra-t* based on preliminary tests. *CPLEX-LocBra-t* is giving t seconds equal to the time given to CPLEX in an intensification step in LocBra heuristic. The literature methods have the same values of parameters as they were presented and set originally by the authors.

LocBra	$k = 20$, $k\_dv = 30$, $total\_time\_limit = 900s$, $node\_time\_limit = 180s$, $dv\_max = 5$, $l\_max = 3$, $dv\_cons\_max = 2$
CPLEX-t	$t = 900$
CPLEX_LocBra-t	$t = 180$
BeamSearch-α	$\alpha = 5$
SBPBeam-α	$\alpha = 5$
IPFP-it	$it = 10$
GNCCP-d	$d = 0.1$

Based on the results shown in Table 4, the heuristics *LocBra*, *CPLEX-900*,

CPLEX_LocBra-180, which are MILP-based, have the highest η_I for all the subsets, and they strongly outperform the two beam search-based heuristics, *IPFP-10* and *GNCCP-0.1*. On easy instances (graphs' subsets between 10 and 40 vertices), the MILP-based heuristics have yielded the best solutions for almost all instances (except 2 instances for subset 40). However, a major difference starts to appear on hard instances (subsets $50, 60, 70$), where $LocBra$ scores the highest values, with 99 for subset 50, 93 for subset 60 and 79 for subset 70 of best solutions. *CPLEX-900* comes in the second place, followed by *CPLEX_LocBra-180*, regarding the number of best solutions obtained. Considering the average deviations, $LocBra$, on hard instances, has the smallest value (d_{avg} less than 0.6%), and again *CPLEX-900* scores the second lowest deviations $0\% \leq d_{avg} \leq 1.05\%$. *IPFP-10* and *GNCCP-0.1* have a maximum deviation d_{avg} of 28%, which means they perform better than the beam-search based heuristics, which are very poor in terms of solutions quality with a very high d_{avg} (about at most 98.9%). With respect to the solution quality, the results show that LocBra strongly outperforms all the literature heuristics, in the case where the default parameters, as in their original references, are used in when executing them. Besides, LocBra also outperforms *CPLEX-900* and *CPLEX-LocBra-180*. To this end, the proposed local branching heuristic is more efficient than the solver and its generic implementation of local branching. Looking at the solution time, *BeamSearch-5* is the fastest with a running time between 0 and 0.18 seconds. Heuristics *IPFP-10*, *SBPBeam-5* and *GNCCP-0.1* come after *BeamSearch-5* in terms of CPU time. Note that, for the instances of mixed sizes, the above conclusions regarding the quality and time still hold. Since the CPU time given to LocBra is high with respect to the literature heuristics with their default parameters, a final experiment is applied. In this one, the parameters of the heuristics are empirically set to as their CPU time is approximately of the same order of $900s$, given to LocBra heuristic.

LocBra	$k = 20$, $k\_dv = 30$, $total\_time\_limit = 900s$, $node\_time\_limit = 180s$, $dv\_max = 5$, $l\_max = 3$, $dv\_cons\_max = 2$
CPLEX-t	$t = 900$
CPLEX_LocBra-t	$t = 800$
BeamSearch-α	$\alpha = 15000$
SBPBeam-α	$\alpha = 400$
IPFP-it	$it = 20000$
GNCCP-d	$d = 0.03$

Table 5 shows the results of the heuristics with extended running time. *LocBra* and *CPLEX_LocBra-800* seem to have d_{avg} very close for small instances. The difference starts to grow on hard and mixed instances where *LocBra* scores the lowest values (less than 0.6%). The average deviation d_{avg} remains very high for beam-search based heuristics: increasing the beam size did not actually improve the results obtained by BeamSearch and SBP-Beam. *BeamSearch-15000* did not return feasible solutions for the set of mixed graphs, therefore the deviations and η_I are not computed. *IPFP-20000* and *GNCCP-0.03* perform better and get smaller deviation comparing to the original versions (Table 4). However, they remain far from LocBra heuristic. Regarding the running time, LocBra is the fastest for subsets 10 and 20, then *GNCCP-0.03* becomes the fastest method for the rest of the subsets. Note that, GNCCP and IPFP are heuristics that have converging conditions, which means that they stop if the condition is satisfied, regardless of the number of iterations left. For such reason, GNCCP is the fastest because it does not reach always its maximum number of iterations.

Based on all the experiments reported in this section, the proposed local branching heuristic significantly improves the literature heuristics and provides near optimal solutions. This is due, to the analysis and the branching scheme combined with the efficiency reached by *CPLEX* when solving $MILP^{JH}$ model. A second important element is the diversification procedure which is problem dependent and really helps the algorithm to escape local optima.

6. Conclusion

This work presents a local branching heuristic for the GED^{EnA} problem based on the $MILP^{JH}$ formulation of Justice and Hero ([9]). Starting from

an initial solution, the heuristic mainly focuses on searching locally in a specific neighborhood for an improved solution. This is done by an intensive use of mathematical programming. In addition, to avoid getting stuck in local minima, it uses a specific diversification mechanism based on problem-dependent information. Next, the heuristic is evaluated on two databases called MUTA ([22]) and PAH ([23]). Two factors are considered: **i-** the solution time and the solutions quality in comparison to other heuristics, **ii-** the solutions closeness to the optimal or best known solutions. The results on easy instances (PAH database) show that the proposed heuristic is capable of finding very good solutions in a short period of time and compete with the exact solution of the $MILP^{JH}$ formulation. The results obtained on MUTA database confirm the large superiority of the proposed local branching heuristic over the literature heuristics. It is important to note that the proposed LocBra heuristic is a significant contribution to the Pattern Recognition research field: this heuristic solves very efficiently some classes of Graph Matching problems, by the way making it possible to improve the solution of other application domains that use such problems like graph classification and clustering, object detection in images or image registration. Remarkably, the local branching heuristic is general enough to be tested on the GED problem at the cost of replacing $MILP^{JH}$ model by a model valid for this problem. Tackling the general problem is planned as an interesting short-term research direction.

References

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

[2] C. F. Moreno-García, X. Cortés, F. Serratosa, 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), Springer, 2016, pp. 519–529.

[3] D. Conte, P. Foggia, C. Sansone, M. Vento, Thirty years of graph matching in pattern recognition, International journal of pattern recognition and artificial intelligence 18 (03) (2004) 265–298.

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

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

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

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

[8] J. Lerouge, Z. Abu-Aisheh, R. Raveaux, P. Héroux, S. Adam, Graph edit distance: a new binary linear programming formulation, arXiv preprint arXiv:1505.05740.

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

[10] S. Bougleux, L. Brun, V. Carletti, P. Foggia, B. Gaüzère, M. Vento, A quadratic assignment formulation of the graph edit distance, arXiv preprint arXiv:1512.07494.

[11] K. Riesen, M. Neuhaus, H. Bunke, Bipartite graph matching for computing the edit distance of graphs, in: International Workshop on Graph-Based Representations in Pattern Recognition, Springer, 2007, pp. 1–12.

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

[13] F. Serratosa, Fast computation of bipartite graph matching, Pattern Recognition Letters 45 (2014) 244–250.

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

[15] M. Neuhaus, K. Riesen, H. Bunke, Fast suboptimal algorithms for the computation of graph edit distance, in: Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR), Springer, 2006, pp. 163–172.

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

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

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

[19] A. Robles-Kelly, M. Loog, B. Biggio, F. Escolano, R. C. Wilson (Eds.), Structural, Syntactic, and Statistical Pattern Recognition - Joint IAPR International Workshop, S+SSPR 2016, Mérida, Mexico, November 29 - December 2, 2016, Proceedings, Vol. 10029 of Lecture Notes in Computer Science, 2016. doi:10.1007/978-3-319-49055-7.
URL https://doi.org/10.1007/978-3-319-49055-7

[20] 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, 2009, pp. 1114–1122.

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

[22] 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 - 10th IAPR-TC-15.Proceedings, 2015, pp. 138–147.

[23] L. Brun, Greyc's chemistry dataset, https://brunl01.users.greyc.fr/CHEMISTRY/.

Appendix A. Extra materials

The details of the functions used in the algorithm of *LocBra* for the GED^{EnA} problem:

Algorithm 2: *LocBra* helper functions

```
1 Function InitLocBra()
2 |  status := MIP_SOLVER(tl, UB, x̃);
3 |  if status = "opt_sol_found" then opt := true; x* := x̃; exit;
4 |  if status = "infeasible" then opt := false; exit;
5 End
1 Function ImprovedSolution()
2 |  if mode_dv = false and x̄ ≠ undefined then
3 |  |  replace last added constraint Δ(x, x̄) ≤ k by Δ(x, x̄) ≥ k + 1;
4 |  end
5 |  x̄ := x̃; UB := f(x̃); mode_dv := false; dv_cons := 0;
6 |  add new constraint Δ(x, x̄) ≤ k;
7 |  if UB < bestUB then  x* := x̃; bestUB := f(x̃) ;
8 End
1 Function Diversification()
2 |  replace last constraint Δ(x, x̄) ≤ k with
  |    Δ(x_important, x̄) ≥ k_div;
3 |  UB := ∞; dv := dv + 1; mode_dv := true; dv_cons := dv_cons + 1;
4 End
1 Function Intensification()
2 |  replace last added constraint Δ(x, x̄) ≤ k by Δ(x, x̄) ≤ k/2;
3 |  mode_dv := false; dv_cons := 0;
4 End
```