The Graph Edit Distance Problem treated by the Local Branching Heuristic

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

The Graph Edit Distance Problem treated by the Local Branching Heuristic

Mostafa Darwiche[1,2], Donatello Conte[1], Romain Raveaux[1], Vincent T'kindt[2]

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

[2] 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) is a well-known problem to match graphs. Solving the GED problem allows computing a dissimilarity measure between graphs that normally represent objects and patterns. It is known to be very flexible and can work on any type of graphs. GED^{EnA} (Edges no Attributes) is a sub-problem of GED that deals with a special type of graphs where edges do not carry attributes. Both are modeled as minimization problems and proven to be NP-Hard, they are exponential in the number of vertices of graphs. A great effort has been made to provide heuristic approaches with a compromise between the execution time and the solution quality. In addition, GED^{EnA} can be expressed efficiently by means of mathematical programming tools to generate Mixed Integer Linear Program (MILP) models. The present work takes advantage of a powerful MILP model 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.

1 Introduction

Graph-based representation is a well-known tool to represent patterns and objects. Graphs are able to depict the components of a pattern by means of vertices, and relational properties between them using edges. Both vertices and edges can carry information and characteristics about the pattern through attributes. Moreover, comparing graphs is considered as an important task since it is very useful in pattern search and classification, and it is also known as the *Graph Matching* (GM) problem. GM problem is very popular in many fields such as computer vision, pattern recognition, biology and chemistry [13, 14]. However, for many years it has been a difficult problem to deal with, due to its computational complexity, especially for large graphs.

Graph Edit Distance (GED) problem belongs to the family of GM problems. It provides a dissimilarity measure between two graphs [6], by computing the cost of editing one graph to transform it into another one. The set of edit operations are substitution, insertion and deletion, and can be applied on both vertices and edges. Solving the GED problem consists in finding the set of edit operations that minimizes the total cost. It has received attention in the past years, because in many work it has been shown that changing the cost metric properties can result in solving other GM problems like maximum common subgraph, graph and subgraph isomorphism [4, 5]. However, GED problem remains a difficult problem and many work has been carried out tackling it with heuristic algorithms, in order to compute good solutions in a reasonable amount of time. The works in [15, 16] present fast algorithms, which mainly solve the linear sum assignment problem for vertices, and then deduce the edges assignment. In these algorithms, the vertices cost matrix includes information about the edges, through estimating the edges assignment cost implied by assigning two vertices from different graphs. However, one drawback in this approach is that, it takes into account only local structures, rather than the global one. Other

algorithms based on beam search are presented in [7, 12]. The first one builds the search tree for all vertices and edges assignment combinations, then only the beam-size nodes are processed. While the second computes an initial solution based on [15] and then tries to improve it by swapping two pairs of assigned vertices. The enumeration of the vertices to be permutated is carried out through a beam search. In the exact solution context, GED problem has been addressed by means of mathematical programming and formulations e.g. linear formulations as in [10] or quadratic formulations as in [2]. A sub-problem of GED is the GED^{EnA} where edges do not carry attributes. The same aforementioned heuristics and exact solution methods can be applied to the GED^{EnA} problem, in addition, a very efficient $MILP^{JH}$ model is found in [9] that works only for the GED^{EnA}. Knowing that GED^{EnA} problem is applied in *Structure-Activity Relationships* domain and considered to be very important [14], there is still a need for having powerful and efficient heuristics for this particular sub-problem.

This work proposes the use of *Local Branching* (LocBra) heuristic to solve the GED^{EnA}. It is presented originally in [8] as a general metaheuristic for *Mixed Integer Linear Program* (MILP). It makes use of a MILP solver in order to explore the solution space, through a defined branching scheme. As well, it involves techniques, such as intensification and diversification during the exploration. To the best of our knowledge, $MILP^{JH}$ is the most efficient model for GED^{EnA} problem, thus it has been chosen in the implementation of LocBra. An adapted version of LocBra is then designed, along with a very efficient diversification mechanism. Henceforth, the heuristic is referred to as $LocBra\_GED^{EnA}$. Subsequently, it is evaluated and compared with existing heuristic algorithms and an exact method.

The remainder is organized as follows: Section 2 presents the definition of GED^{EnA} problem and a review of $MILP^{JH}$ model. Then, Section 3 details the proposed heuristic, and Section 4 shows the results of the computational experiments. Finally, Section 5 highlights some concluding remarks.

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

To introduce the general *Graph Edit Distance* (GED) problem, the definition of attributed and directed graph is given first.

Definition 1. *An attributed and directed 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, some vertices and edges edit operations are available: $(u \to v)$ is the substitution of two vertices, $(u \to \epsilon)$ is the deletion of a vertex, and $(\epsilon \to v)$ is the insertion of a vertex, with $u \in V, v \in V'$ and ϵ refers to the empty vertex. The same logic goes for the edges. The set of operations that reflects a valid transformation of G into G' is called a complete edit path, defined as $\lambda(G, G') = \{e_i, i \in \{1, n\}\}$ where e_i is an elementary vertex (or edge) edit operation and n is the number of operations.

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

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

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 c is a function that assigns the costs to elementary edit operations.

For GED^{EnA} problem, the graphs are the same as in Definition 1, but with $L_E = \{\phi\}$. Consequently, the costs of edge edit operations are 0 for substitution and a constant for insertion and deletion (i.e. $c(e \to f) = 0$, $c(e \to \epsilon) = const$, $c(\epsilon \to f) = const$, $\forall e, f \in E$).

$MILP^{JH}$ is a model proposed in [9] that solves the GED^{EnA} problem. The main idea consists in determining the permutation matrix minimizing the L_1 norm of the difference between adjacency matrix

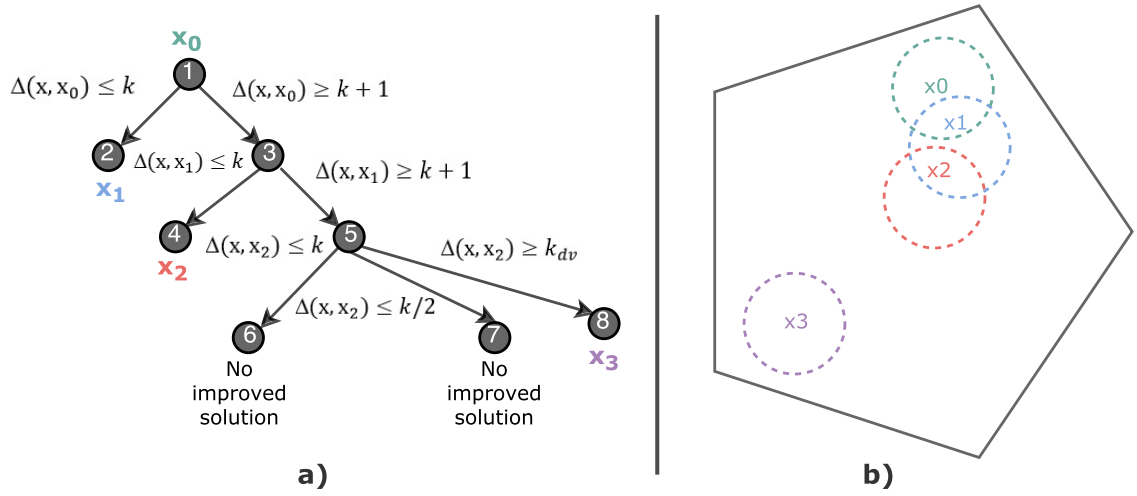

Figure 1: Local branching flow. a) depicts the left and right branching. b) shows the neighborhoods in the solution space

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]. The model is as follows:

$$\min_{P,S,T\in\{0,1\}^{N\times N}} \sum_{i=1}^{N}\sum_{j=1}^{N} c\left(\mu(u_i),\mu'(v_j)\right) P^{ij} + \left(\frac{1}{2}\times const\times (S+T)^{ij}\right) \tag{2}$$

such that

$$\left(AP - PA' + S - T\right)^{ij} = 0\ \forall i,j \in \{1,N\} \tag{3}$$

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

where A and A' are the adjacency matrices of graphs G and G' respectively, $c : (\mu(u_i),\mu'(v_j)) \rightarrow \mathbb{R}^+$ is the cost function that measures the distance between two vertices attributes. As for P, S and T, they are the permutation matrices of size $N \times N$, and of boolean type, with $N = |V| + |V'|$. P represents the vertices matching e.g. $P^{ij} = 1$ 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 P.

3 Local Branching Heuristic for GED^{EnA}

As presented in [8], LocBra heuristic is a local search approach that makes use of MILP solver to explore the neighborhoods of solutions through a branching scheme. In addition, it involves mechanisms such as intensification and diversification. Starting from an initial solution x_0, it defines the *k-opt neighborhood* $N(x_0,k)$, with k a given integer. In other words, the neighborhood set contains the solutions that are within a distance no more than k from x_0 (in the sense of *Hamming distance*). This implies adding the following *local branching constraint* to the base $MILP^{JH}$ model:

$$\Delta(x,x_0) = \sum_{j\in S_0}(1-x_j) + \sum_{j\in B\setminus S_0} x_j \leq k \tag{5}$$

such that, B is the index set of binary variables defined in the model, and $S_0 = \{j \in B : \{x_0\}_j = 1\}$. This new model is then solved leading to the search of the best solution in $N(x_0,k)$. This phase

Algorithm 1: $LocBra\_GED^{EnA}$ algorithm

```
1  bestUB := UB := ∞; x* := x̄ := x̃ := undefined;
2  tl := elapsed_time := dv := l := dv_cons := 0;
3  mode_dv := false; opt := false; first_loop := true;
1  Function LocBraGED(k, k_dv, total_time_limit, node_time_limit, dv_max, l_max, dv_cons_max)
       Output: x*, opt
2      InitLocBraGED();
3      ImprovedSolution();
4      elapsed_time := tl;
5      while elapsed_time < total_time_limit and dv < dv_max and dv_cons < dv_cons_max do
6          tl := min{tl, total_time_limit − elapsed_time};
7          status := MIP_SOLVER(tl, UB, x̃);
8          tl := node_time_limit;
9          if ValueOf(x̃) = ValueOf(x̄) and mode_dv = true then l := l + 1 else l := 0;
10         if l ≥ l_max then Diversification(); continue;
11         if status = "opt_sol_found" then
12             if x̃ ≠ x̄ then ImprovedSolution() else Diversification();
13         end
14         if status = "proven_infeasible" then Diversification();
15         if status = "feasible_sol_found" then
16             if ValueOf(x̃) < UB then
17                 ImprovedSolution();
18             else
19                 if mode_dv = false then Intensification() else Diversification();
20             end
21         end
22         elapsed_time := elapsed_time + tl;
23     end
24 End
1  Function InitLocBraGED()
2      status := MIP_SOLVER(tl, UB, x̃);
3      if status = "opt_sol_found" then opt := true; x* := x̃; exit;
4      if status = "proven_infeasible" then opt := false; exit;
5  End
1  Function ImprovedSolution()
2      if mode_dv = false and x̄ ≠ undefined then
3          replace last constraint Δ(x, x̄) ≤ k with Δ(x, x̄) ≥ k + 1;
4      end
5      x̄ := x̃; UB := ValueOf(x̃); mode_dv := false; dv_cons := 0;
6      add new constraint Δ(x, x̄) ≤ k;
7      if UB < bestUB then x* := x̃; bestUB := ValueOf(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 constraint Δ(x, x̄) ≤ k with Δ(x, x̄) ≤ k − k/2;
3      mode_dv := false; dv_cons := 0;
4  End
```

corresponds to intensifying the search in a neighborhood e.g. node 2 in Fig 1-a. If a new solution x_1 is found, the constraint (Eq. 5) is replaced by $\Delta(x, x_0) \geq k + 1$, at the right branch (node 3 in Fig. 1-a). Next, a left branch is recreated but now using x_1, and the process is repeated until a stopping criterion is met e.g. a *total time limit* is reached. However, and since solving sub-problems (with local branching constraints) may not be possible in a reasonable time, a *node time limit* is imposed at each branch. Therefore, it cannot be generalized that an improved solution could be found at a branch, due to reasons such as *node time limit* is reached, or the problem has become infeasible. For instance, assuming that at node 6 (Fig. 1-a) the solution of model $MILP^{JH}$ plus equation $\Delta(x, x_2) \leq k$ does not lead to a feasible solution in the given time limit. It might be interesting to apply a complementary intensification phase, by adding constraint $\Delta(x, x_2) \leq k/2$ and solving the new model. If again, no feasible solution is found (e.g. node 7 of Fig.1-a), then a diversification phases is applied to jump to another point in the solution space (e.g. node 8). Fig. 1-b shows the evolution of the solution search and the neighborhoods.

$LocBra\_GED^{EnA}$ is a modified and adapted version to deal with the GED^{EnA} problem, and is detailed in Algo. 1. The input parameters are: **i-** k is the neighborhood size, **ii-** k_dv is for diversification to skip current solution, **iii-** $total\_time\_limit$ stopping criterion, represents the total running time, **iv-** $node\_time\_limit$ forces the solver to exit and return the found solution (if any), **v-** dv_max stopping criterion, is the number of diversification allowed, **vi-** l_max is to force a diversification after a sequence of branching returning the same solutions, **vii-** dv_cons_max serves as a stopping criterion, in case consecutive diversifications have returned the same solutions, then the heuristic will stop. As for the output, the algorithm returns the best solution found x^*, and the optimality opt status. In detail, function *LocBraGED* describes the flow of the heuristic, it starts by calling the *InitLocBraGED* function, which initializes the heuristic by getting a first solution $\bar{x}$. It calls function *MIP_SOLVER* to solve the model as it is, with a time limit. If at this point, the model is solved to optimality or proven infeasible, the heuristic halts and returns the available solution and status. Else, the initial solution is set and the exploration begins. A loop takes place until at least one of the stopping criterion is violated. At each iteration and after a left/right branching constraint is added, the solver is called again and the returned status is considered to make the next decision. Three main cases may occur: **i-** The Optimal solution is found (line 11), and two cases must be distinguished. Either $\tilde{x}$ (new solution) is better than $\bar{x}$ (current solution), then *ImprovedSolution* is called to switch the current and best (if needed) solutions, also to add the local branching constraints and define a new neighborhood. Or it has found the same solution $\bar{x}$, thus *Diversification* is called to skip the current neighborhood. *Diversification* function ensures that the current solution is skipped with a distance k_dv, and the upper bound UB is reset to ∞. **ii-** The model is infeasible (line 14), therefore *Diversification* is triggered to switch the last local branching constraint and look into a new neighborhood. **iii-** A feasible solution is returned (line 15). This is very close to case **i-**, except when a worse solution is found ($ValueOf(\tilde{x}) < UB$), an *Intensification* step is introduced. It shrinks the neighborhood by $k/2$ to boost the exploration. However, a failed *Intensification* is then followed by a *Diversification*. In addition, there is the condition (at line 10) that forces the diversification, in the case where l_max iterations have returned the same solution. This in turn guarantees the exploration of many neighborhoods, regardless of the new solutions' quality (whether better or worse).

The key point of this heuristic is the selection of the variables while branching. For instance, the x vector in $\Delta(x, \bar{x})$ contains only the set of binary variables that represent the vertices assignment (edges assignment are excluded). The reason behind this relies on the fact that edges assignment are driven by the vertices assignment, i.e. deleting one vertex implies deleting all edges that are connected to it, this is based on the definition of the GED^{EnA} problem. For diversification, it is slightly different, a vector x_{imp} is defined such that, instead of forcing k_dv flips over the whole set of vertices assignment variables, it is done over a subset of **important** variables. The selection of these variables is based on the assumption that one variable is important if changing its value from $1 \rightarrow 0$ (or the opposite) highly impacts the objective function's value. This, in turn, helps skipping local solutions and change the matching. Accordingly, the selection of variables in x_{imp} is done by computing a special cost matrix $[C_{ij}]$ for each possible assignment of a vertex $i \in V \cup \{\epsilon\}$, to a vertex $j \in V' \cup \{\epsilon\}$. Each value $C_{ij} = c_{ij} + \theta_{ij}$, where c_{ij} is the node operation cost induced by assigning vertex i to vertex j, and θ_{ij} is the

	LocBra_GED^{EnA}	*CPLEX-12.48*	*CPLEX_LocBra-3.5*	*BeamSearch-5*	*SBPBeam-5*
t_{min}	0.06	0.05	0.05	0.00	0.01
t_{avg}	3.03	1.97	1.79	**0.01**	0.14
t_{max}	12.25	12.48	6.41	0.03	0.37
d_{min}	0.00	0.00	0.00	0.00	0.00
d_{avg}	0.31	**0.05**	0.91	122.65	379.90
d_{max}	75.00	190.91	200.00	2400.00	4200.00
η_I	8716	**8830**	8553	433	100

Table 1: $LocBra\_GED^{EnA}$ vs. literature heuristics on PAH instances

	CPLEX-∞	$LocBra\_GED^{EnA}$
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 2: $LocBra\_GED^{EnA}$ vs. Exact solution on PAH instances

cost of assigning the set of edges $E_i = \{(i, v) \in E\}$ to $E_j = \{(j, v') \in E'\}$. This assignment problem, of size $max(|E_i|, |E_j|) \times max(|E_i|, |E_j|)$, is solved by the Hungarian algorithm [11] which requires $(O(max(|E_i|, |E_j|)^3))$ time. Next, the standard deviation is computed at each row of the matrix $[C_{ij}]$, resulting in a vector $[\sigma_i]$. Then, they are split into two clusters *min* and *max*, by starting with the minimum σ_{min} and maximum σ_{max} values as the centers of the clusters. $\forall i \in V \cup \{\epsilon\}$ if $|\sigma_i - avg_{min}| < |\sigma_i - avg_{max}|$ then $\sigma_i \rightarrow min$, otherwise $\sigma_i \rightarrow max$, with avg_{min} and avg_{max} are the averages of the existing values in the clusters. Finally, for every σ_i belonging to *max* cluster, all $\{x_{ij}, \forall j \in |V'| \cup \{\epsilon\}\}$ variables are added to x_{imp}. Henceforth, the diversification constraint is $\Delta(x_{imp}, \bar{x}) \geq k_{dv}$. Consequently, the local structure of a vertex is considered to assess its influence on the objective function value. Preliminary experiments, not reported here, have shown that such diversification helps improving the local branching heuristic better than the original diversification defined in [8].

4 Computational Experiments

This section shows the computational experiments conducted to evaluate the efficiency of $LocBra\_GED^{EnA}$ heuristic, with respect to the literature algorithms. These experiments have been done on reference databases from Pattern Recognition community, where researches have introduced the GM problems. Therefore, two databases of chemical molecules graphs are chosen, MUTA [1] and PAH [3]. The first one contains different subsets of small and large graphs and is known to be difficult to solve. It has 7 subsets, each of which has 10 graphs of same size (10 to 70 vertices). The second database contains 94 graphs, with at most 28 vertices. Each pair of graphs is considered as an instance. Therefore, MUTA has a total of 700 instances (100 per subset) and PAH has 8836 instances.

Experiment settings and evaluation metrics: $LocBra\_GED^{EnA}$ algorithm is implemented in C. The solver CPLEX 12.6.0 is used to solve the MILP formulation. Experiments are ran on a machine Intel Core $i4$ with 8 GB RAM. For each database, two experiments are conducted. The first one is to compare $LocBra\_GED^{EnA}$ against existing heuristics, while the second one studies the quality of the solutions obtained by comparing them to the optimal or best known ones found by CPLEX without time or resource (e.g. RAM) limits. In the first experiment, the following metrics are computed for each heuristic: $t_{min}, t_{avg}, t_{max}$ are the minimum, average and maximum CPU time in seconds for all instances. Correspondingly, $d_{min}, d_{avg}, d_{max}$ are the deviation percentages for the solutions obtained by one heuristic, from the best solutions found. Given an instance I and an heuristic H, deviation percentage is equal to $\frac{solution_I^H - bestSolution_I}{bestSolution_I} \times 100$, with $bestSolution_I$ is the smallest 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. In

	S	10	20	30	40	50	60	70
$LocBra\_GED^{EnA}$	t_{min}	0.06	0.13	0.28	0.45	0.69	0.95	1.36
	t_{avg}	0.17	1.12	212.36	364.86	580.04	753.48	751.44
	t_{max}	2.92	3.63	900.13	900.12	900.17	900.27	900.36
	d_{min}	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**
	d_{max}	0.00	0.00	0.00	3.90	2.03	3.35	5.57
	η_I	**100**	**100**	**100**	98	**99**	**93**	**79**
CPLEX-900	t_{min}	0.06	0.14	0.28	0.49	0.77	1.18	1.70
	t_{avg}	0.13	1.02	141.07	247.80	451.40	723.68	745.91
	t_{max}	0.49	3.52	900.20	900.42	900.46	900.71	900.92
	d_{min}	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
	d_{max}	0.00	0.00	0.00	0.00	6.42	5.04	8.57
	η_I	**100**	**100**	**100**	**100**	90	81	68
CPLEX_LocBra-180	t_{min}	0.09	0.22	0.41	0.73	1.03	1.45	1.98
	t_{avg}	0.21	1.51	60.36	104.19	141.43	167.59	181.18
	t_{max}	0.74	5.77	182.86	194.08	195.43	217.38	263.60
	d_{min}	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
	d_{max}	0.00	0.00	0.00	3.90	7.19	6.70	27.20
	η_I	**100**	**100**	**100**	94	72	57	41
CPLEX_LocBra-800	t_{min}	0.08	0.21	0.38	0.67	1.01	1.40	1.94
	t_{avg}	0.20	1.34	130.26	230.68	424.70	662.58	688.13
	t_{max}	0.71	3.90	802.16	806.16	821.39	839.69	869.65
	d_{min}	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	d_{avg}	**0.00**	**0.00**	**0.00**	0.04	0.38	0.60	1.06
	d_{max}	0.00	0.00	0.00	3.90	6.42	5.04	11.27
	η_I	**100**	**100**	**100**	99	89	80	69
BeamSearch-5	t_{min}	0.00	0.00	0.01	0.01	0.02	0.04	0.06
	t_{avg}	**0.00**	**0.00**	**0.01**	**0.03**	**0.07**	**0.11**	**0.18**
	t_{max}	0.07	0.02	0.04	0.11	0.09	0.13	0.22
	d_{min}	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
	d_{max}	110.00	124.59	147.37	186.67	200.00	146.37	210.71
	η_I	35	10	10	10	10	10	10
BeamSearch-15000	t_{min}	0.00	0.00	0.03	0.10	0.55	0.24	2.28
	t_{avg}	8.57	80.65	167.48	279.11	439.68	640.29	938.66
	t_{max}	31.52	118.71	230.63	419.73	771.90	878.89	1385.11
	d_{min}	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	d_{avg}	1.35	26.66	47.45	52.35	63.98	62.57	63.76
	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-5	t_{min}	0.01	0.08	0.31	1.11	2.69	4.87	9.02
	t_{avg}	0.01	0.10	0.45	1.37	3.19	5.56	10.72
	t_{max}	0.05	0.14	0.54	1.60	3.71	6.85	12.79
	d_{min}	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
	d_{max}	90.00	127.87	206.90	204.71	314.29	198.50	280.36
	η_I	15	10	10	10	10	10	10
SBPBeam-500	t_{min}	0.76	9.02	39.85	116.11	288.38	548.04	1019
	t_{avg}	0.84	10.02	47.65	139.75	322.43	590.86	1155
	t_{max}	0.96	11.27	54.11	152.34	360.47	657.26	1310
	d_{min}	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
	d_{max}	90.00	127.87	206.90	204.71	314.29	198.50	280.36
	η_I	15	10	10	10	10	10	10

Table 3: $LocBra\_GED^{EnA}$ vs. literature heuristics on MUTA instances

	CPLEX-∞ (4 threads)				*$LocBra\_GED^{EnA}$ (1 thread)*									*$LocBra\_GED^{EnA}$ (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	21.8	0.00	**0.00**	0.00	**100**	**100**	0
30	0.31	**101**	2865	100	0.28	212	900	0.00	**0.00**	0.00	78	**100**	0	0.32	**101**	900	0.00	**0.00**	0.00	**91**	**100**	0
40	0.52	266	9244	99	0.45	365	900	0.00	0.06	3.90	63	98	0	0.49	**179**	900	0.00	**0.00**	0.00	**84**	**100**	0
50	0.83	683	4213	92	0.69	580	900	-1.79	0.04	4.14	37	97	**1**	0.73	**435**	900	-1.79	**0.00**	2.07	**54**	**98**	**1**
60	1.24	2419	14732	71	0.95	753	900	-2.68	0.36	3.57	16	82	2	1.09	**718**	902	-3.31	**-0.03**	3.21	**21**	**90**	**6**
70	1.80	3740	24185	35	1.36	751	900	-2.67	0.78	8.85	17	52	14	1.48	**741**	901	-3.90	**0.22**	3.65	**18**	**60**	**16**

Table 4: $LocBra\_GED^{EnA}$ vs. Exact solution on MUTA instances

the second experiment, time and deviation metrics are calculated for each method as stated previously. In addition, η_I, η'_I, η''_I are, respectively, the number of optimal solutions found, the number of solutions found that are equal to the optimal or best known ones, and the number solutions that are better than the best known ones.

Comparative methods: *CPLEX-t* heuristic is ran on $MILP^{JH}$, where t represents the time limit for CPLEX to try finding the optimal solution. The method becomes exact when t is set to ∞, therefore CPLEX will not stop until the optimal solution is found, or there are no more resources available on the machine, and in this case the solution obtained is referred to as the best known solution. Also, CPLEX has its own local branching implementation, which gets applied only on nodes of the B&B's tree where a new incumbent solution is found. To include it in the evaluation, a feasible solution is computed first (as an incumbent solution) and then CPLEX local branching is called on the first node of the tree. This is called *CPLEX_LocBra-t*, with t is the time limit allowed to compute the feasible solution. From the literature, to the best of our knowledge, the heuristics *BeamSearch-*α [12] and *SBPBeam-*α [7] are known to be the best, so they are chosen in the evaluation (α is the beam size).

Results and analysis:

- **PAH tests**: In order to set the parameters of the heuristics, preliminary tests were done but are not shown here. PAH instances are small, which means that CPLEX is capable of finding the optimal solutions in few seconds. The parameters of $LocBra\_GED^{EnA}$ are set to: $k = 20$, $k\_dv = 30$, $total\_time\_limit = 12.25s$, $node\_time\_limit = 1.75s$, $dv\_max = 5$, $l\_max = 3$, $dv\_cons\_max = 2$. For CPLEX with time limit, $t = 12.48s$. *CPLEX_LocBra-t* is launched with $t = 3.5s$. Lastly, α is set to 5 for *BeamSearch* and *SBPBeam* as in the experiments done in [7].

  The results of the heuristics comparison is shown in Table 1. *CPLEX-12.48* has an average deviation of 0.05% which is the smallest among all the heuristics. Next $LocBra\_GED^{EnA}$ comes with 0.31%. Clearly, *CPLEX-12.48* has performed better than the proposed heuristic. However, an important note is the d_{max}: $LocBra\_GED^{EnA}$ has 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 are strongly outperformed by the other MILP-based heuristics with a high average deviation. On the other hand, the beam-search based heuristics seems to be very fast ($t_{avg} < 1s$), while the proposed heuristics is the slowest with $t_{avg} = 3.03s$. Moreover, Table 2 shows the results of comparing the heuristic against the exact solution. *CPLEX-*∞ is on the average faster than $LocBra\_GED^{EnA}$ but in the worst case CPLEX becomes computationally expensive (up to $278.20s$), while $LocBra\_GED^{EnA}$ remains at $12.25s$ max. Further, the average deviation is 0.35% with 8702 instances having the same solutions as the optimal ones, shows that $LocBra\_GED^{EnA}$ is able to find the optimal solutions or stay very close to them.

- **MUTA tests**: The instances of MUTA database are much more difficult to solve than PAH instances, therefore the time limits are increased. $LocBra\_GED^{EnA}$ parameters are set to: $k = 20$, $k\_dv = 30$, $total\_time\_limit = 900s$, $node\_time\_limit = 180s$, $dv\_max = 5$, $l\_max = 3$, $dv\_cons\_max = 2$. t is set to $900s$ in CPLEX heuristic. Then, two cases are considered for CPLEX local branching: *CPLEX_LocBra-180* where $180s$ is spent to compute a feasible solution, which is the same time as one iteration of $LocBra\_GED^{EnA}$. And *CPLEX_LocBra-800* spends $800s$ to find a feasible solution, this will give a time limit $\simeq 900s$ as the total time limit of $LocBra\_GED^{EnA}$. For the beam-search based heuristics, and since their performances depend on the beam size, two versions of each are considered: *BeamSearch-5*, *Beam Search-15000*, *SBPBeam-5* and *SBPBeam-400*. The reason for which α is set to 15000 and 400 is to give the heuristics an average time close to the total time limit ($900s$) of $LocBra\_GED^{EnA}$. As well in the exact solution, *CPLEX-*∞ is set to work with 4 threads to make use of maximum resources of the machine and the power of CPLEX to find the optimal or best solutions. Then, two versions of $LocBra\_GED^{EnA}$ are considered with the same parameters as stated previously, but the first version has only 1 thread while the second has 4 threads.

Based on the results shown in Table 3, the heuristics $LocBra\_GED^{EnA}$, *CPLEX-900*, *CPLEX_LocBra-180* and *CPLEX_LocBra-800*, which are MILP-based, have the highest η_I for all the subsets, and they strongly outperform the four beam search-based heuristics. On easy instances (graphs' subsets between 10 and 40), they have yielded the best solutions for almost all instances (except few instances for subset 40). However, a major difference starts to appear on hard instances (subsets $50, 60, 70$), where $LocBra\_GED^{EnA}$ scores the highest values, with $99, 93, 79$ (over 100) best solutions. Next, *CPLEX-900* and *CPLEX_LocBra-800* seem to be very close in the number of best solutions obtained, however the former is slightly better. CPLEX_LocBra-180 comes at fourth place, with less number of best solutions. Remarkably *CPLEX-900* achieves better values than both *CPLEX_LocBra* 180 and 800, which means that the default behavior of the solver with the default embedded heuristics is more efficient. Considering the average deviations, $LocBra\_GED^{EnA}$ on hard instances has the smallest value (d_{avg} less than 0.6%), and again *CPLEX-900* and *CPLEX_LocBra-800* are close with $0\% \leq d_{avg} \leq 1.06\%$. The beam-search based heuristics are very poor in terms of solutions quality, their d_{avg} are very high (reaches 98.9%) and the numbers of best solutions are very small. Considering the solution time, *BeamSearch-5* and *SBPBeam-5* are the fastest with time between 0 and 10 seconds. Moreover, even after increasing the beam size, which increases their solution time, both are not able to provide better solutions and the average deviations remain high. The Results of comparing the proposed heuristic with the exact solution of $MILP^{JH}$ are reported in Table 4. For easy instances (graphs' subsets between 10 and 40), all optimal solutions, but one, are found by *CPLEX-∞ (4 threads)*, and both $LocBra\_GED^{EnA}$ with 1 and 4 threads have 0% as d_{avg} (except for 2 instances in subset 40). This clearly means that the heuristic is able to find the same best solutions. However, in terms of CPU time, *CPLEX-∞ (4 threads)* spent more time ($> 900s$) proving optimality, while the heuristic has reached the same solutions with less time (max of $900sec$). For hard instances, d_{avg} is always less than 1%, and even less than 0% (-0.03%) with $LocBra\_GED^{EnA}$ *(4 threads)* for subset 60. It is important to note as well, that the CPU time drastically increases for *CPLEX-∞* and reaches thousands ($t_{avg} = 3740s$), while the heuristic has a max $t_{avg} = 751s$. 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. As a conclusion, $LocBra\_GED^{EnA}$ is capable of finding the optimal or best solutions as *CPLEX* in exact solution mode, and even better in some cases.

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.

5 Conclusion

This work presents a local branching heuristic for GED^{EnA} problem based on the $MILP^{JH}$ formulation presented in [9]. Starting from an initial solution, the heuristic mainly focuses on searching locally in a specific neighborhood for an improved solution. In addition, to avoid getting stuck in local minima, it uses a specific diversification mechanism, that ensures defining and visiting important neighborhoods. Next, the heuristic is evaluated on two databases of chemical graphs MUTA and PAH. Two factors are considered, the solution time and the solutions quality in comparison to other heuristics, and the solutions closeness to the optimal or best known ones. The results on easy instances (PAH database) show that the heuristic is capable of finding very good solutions in a short period of time and compete with CPLEX in the exact mode. The results obtained on MUTA database confirm the large superiority of the proposed local branching heuristic over the literature heuristics. 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 next step in the near future.

References

[1] Zeina Abu-Aisheh, Romain Raveaux, and Jean-Yves 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*, pages 138–147, 2015.

[2] Sébastien Bougleux, Luc Brun, Vincenzo Carletti, Pasquale Foggia, Benoit Gaüzère, and Mario Vento. Graph edit distance as a quadratic assignment problem. *Pattern Recognition Letters*, 2016.

[3] Luc Brun. Greyc's chemistry dataset. https://brunl01.users.greyc.fr/CHEMISTRY/.

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

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

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

[7] Miquel Ferrer, Francesc Serratosa, and Kaspar Riesen. Improving bipartite graph matching by assessing the assignment confidence. *Pattern Recognition Letters*, 65:29–36, 2015.

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

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

[10] Julien Lerouge, Zeina Abu-Aisheh, Romain Raveaux, Pierre Héroux, and Sébastien Adam. Exact graph edit distance computation using a binary linear program. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 485–495. Springer, 2016.

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

[12] Michel Neuhaus, Kaspar Riesen, and Horst 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)*, pages 163–172. Springer, 2006.

[13] Romain Raveaux, Jean-Christophe Burie, and Jean-Marc Ogier. Structured representations in a content based image retrieval context. *J. Visual Communication and Image Representation*, 24(8):1252–1268, 2013.

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

[15] Kaspar Riesen, Michel Neuhaus, and Horst Bunke. Bipartite graph matching for computing the edit distance of graphs. In *International Workshop on Graph-Based Representations in Pattern Recognition*, pages 1–12. Springer, 2007.

[16] Francesc Serratosa. Fast computation of bipartite graph matching. *Pattern Recognition Letters*, 45:244–250, 2014.