A Local Branching Heuristic for the Graph Edit Distance Problem

Mostafa Darwiche[1,2], Romain Raveaux[1(✉)], Donatello Conte[1], and Vincent T'Kindt[2]

[1] Laboratoire d'Informatique (LI), Université François Rabelais Tours, Tours, France
{mostafa.darwiche,romain.raveaux,donatello.conte}@univ-tours.fr
[2] Laboratoire d'Informatique (LI), ERL-CNRS 6305, Université François Rabelais Tours, Tours, France
tkindt@univ-tours.fr

Abstract. In graph matching, Graph Edit Distance (GED) is a well-known distance measure for graphs and it is a NP-Hard minimization problem. Many heuristics are defined in the literature to give approximated solutions in a reasonable time. Some other work have used mathematical programming tools to come up with Mixed Integer Linear Program (MILP) models. In this work, a heuristic from Operational Research domain, is proposed and adapted to handle *GED* problem. It is called Local Branching and operates over a MILP model, where it defines neighborhoods in the solution space by adding the local branching constraint. A black-box MILP solver is then used to intensify the search in a neighborhood. This makes the solution search very fast, and allow exploring different sub-regions. Also, it includes a diversification mechanism to escape local solutions and in this work this mechanism is modified and improved. Finally, it is evaluated against other heuristics in order to show its efficiency and precision.

Keywords: Graph Edit Distance · Graph matching
Local branching heuristic

1 Introduction

A powerful and well-known tool to represent patterns and objects is offered by the graph-based representation. Graphs are able to depict the components of a pattern by the mean of vertices, and the relational properties between them using edges. Moreover, through the attributes (labels) that are assigned to vertices and edges, a graph can carry more information and characteristics about the pattern. Finding the (dis)similarities between two graphs requires the computation and the evaluation of the best matching between them. Since exact isomorphism rarely occurs in pattern analysis applications, the matching process must be error-tolerant, i.e., it must tolerates differences in the topology and/or its labeling. Finally, graphs can be used in tasks such as classification and clustering, by studying and comparing them. Error-tolerant graph matching problem is known to be difficult, mainly due to its computational complexity, especially for large graphs. Graphs have become popular in the context of *Structure-Activity Relationships* (SAR), since they provide 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 [9]. SAR is an important application field for graph matching.

Graph Edit Distance (GED) problem is an error-tolerant graph matching problem. It provides a dissimilarity measure between two graphs, by computing the cost of editing one graph to transform it into another. The set of edit operations are substitution, insertion and deletion, and can be applied on both vertices and edges. Solving the GED problem consists of finding the set of edit operations that minimizes the total cost. GED, by concept, is known to be flexible because it has been shown that changing the edit cost properties can result in solving other matching problems like maximum common subgraph, graph and subgraph isomorphism [3]. Also, GED problem has many applications such as Image Analysis, Handwritten Document Analysis, Malware Detection, Bio- and Chemoinformatics and many others [12]. GED is a NP-Hard problem [13], so the optimal solution cannot be obtained in polynomial time. Nevertheless, many heuristics have been proposed to compute good solutions in reasonable amount of time. The works in [10,11] presented fast algorithms that mainly solve the linear sum assignment problem for vertices, and then deduce the edges assignment. The vertices cost matrix includes information about the edges, through estimating the edges assignment cost implied by assigning two vertices. 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 [4,8]. The first builds the search tree for all vertices assignment, then only the beam-size nodes are processed. While the second solves the vertices assignment problem, and then using beam search, it tries to improve the initial assignment by switching vertices, and re-computing the total cost. On the other hand, GED problem is addressed by the means of mathematical programming tools. Two types of mathematical formulations can be found in the literature: linear models as in [6,7] and quadratic as in [2].

This work proposes the use of *Local Branching* (LocBra) heuristic, which is well-known in *Operational Research* domain. It is presented originally in [5] as a general metaheuristic for MILP models. 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 benefit from the efficiency of these techniques, a LocBra version is designed and adapted to handle the GED problem. Also, the original diversification is modified to consider information about the problem that will improve the exploration and return better solutions. Since LocBra depends on a MILP model, $MILP^{JH}$ is chosen in the implementation of LocBra. Because to the best of our knowledge, it is one of the most efficient for GED problem [7]. Henceforth, the heuristic is referred to as *LocBra_GED*. Subsequently, it is evaluated and compared with existing competitive heuristic algorithms. The remainder is organized as follows: Sect. 2 presents the definition of GED problem, followed with a review of $MILP^{JH}$ model. Then, Sect. 3 details the proposed heuristic. And Sect. 4 shows the results of the computational experiments. Finally, Sect. 5 highlights some concluding remarks.

2 *GED* Definition and $MILP^{JH}$ Model

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

Next, given two graphs $G = (V, E, \mu, \xi)$ and $G' = (V', E', \mu', \xi')$, GED is the task of transforming one graph source into another graph target. To accomplish this, GED introduces the vertices and edges edit operations: $(u \to v)$ is the substitution of two nodes, $(u \to \epsilon)$ is the deletion of a node, and $(\epsilon \to v)$ is the insertion of a node, with $u \in V, v \in V'$ and ϵ refers to the empty node. The same logic 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 $\lambda(G, G') = \{e_1, \ldots, e_k\}$ where e_i is an elementary vertex (or edge) edit operation and k is the number of operations. GED is then

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

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 the cost function that assigns the costs to elementary edit operations.

$MILP^{JH}$ is a model proposed in [6], that solves the GED problem. The main idea consists in determining the permutation matrix minimizing the L_1 norm of the difference between adjacency matrix of the input graph and the permuted adjacency matrix of the target one. The details about the construction of the model can be found in [6]. 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

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

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

where A and A' are the adjacency matrices of graphs G and G' respectively, $c : (\mu(u_i), \mu'(v_j)) \to \mathbb{R}^+$ 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

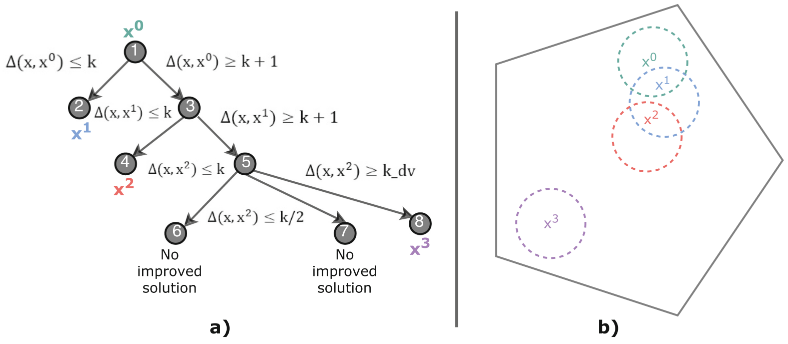

Fig. 1. Local branching flow. (a) Depicts the left and right branching. (b) Shows the neighborhoods in the solution space

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. This model has a limitation that it does not consider the attributes on edges, so edge substitution costs 0 while deletion and insertion have a $const \in \mathbb{R}^+$ cost.

3 Local Branching Heuristic for *GED*

The general MILP formulation is of the form:

$$\min_x \ c^T x \tag{5}$$

$$Ax \geq b \tag{6}$$

$$x_j \in \{0, 1\}, \forall j \in B \neq 0 \tag{7}$$

$$x_j \in \mathbb{N}, \forall j \in I \tag{8}$$

$$x_j \in \mathbb{R}, \forall j \in C \tag{9}$$

where the variable index set is split into three sets (B, I, C), respectively stands for binary, integer and continuous. $MILP^{JH}$ can be written in the same form, where the sets $I = C = \{\phi\}$ since all the variables are binary.

As presented in [5], 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:

$$\Delta(x, x^0) = \sum_{j \in S^0} (1 - x_j) + \sum_{j \in B \setminus S^0} x_j \leq k \tag{10}$$

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 corresponds to intensifying the search in a neighborhood e.g. node 2 in Fig. 1a. If a new solution x^1 is found, the constraint (Eq. 10) is replaced by $\Delta(x, x^0) \geq k + 1$ (node 3 in Fig. 1a). This constraint makes sure that visited solutions (e.g. x^0) will not be selected twice. Next, a new constraint Eq. 10 is added but now with x^1 to explore its neighborhood. The process is repeated until a stopping criterion is met e.g. a *total time limit* is reached. Moreover, it cannot be generalized that an improved solution could be found, due to reasons such as node time limit is reached or the problem may become infeasible. For instance, assuming that at node 6 (Fig. 1a) 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. 1a), then a diversification phase is applied by adding the constraint $\Delta(x, x^2) \geq k\_dv$ (with k_dv a positive integer), to jump to another point in the solution space (e.g. node 8). The diversification constraint simply forces the algorithm to change the region in the search space by choosing any solution that has k_dv differences from the current solution (the authors in [5] uses another diversification constraint). Figure 1b shows the evolution of the solution search and the neighborhoods.

The key point of this heuristic is the selection of the variables while branching. To adapt the method to GED, $\Delta(x, x^i)$ is replaced by $\Delta'(x, x^i)$ where 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* problem. Another 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. The diversification constraint is then $\Delta''(x, x^i) = \sum_{j \in S^i_{imp}} (1 - x_j) + \sum_{j \in B_{imp} \setminus S^i_{imp}} x_j \geq k\_dv$ with B_{imp} is the index set of binary important variables and $S^i_{imp} = \{j \in B_{imp} : x^i_j = 1\}$. The selection of these variables is based on the assumption that one variable is considered important if changing its value from $1 \to 0$ (or the opposite) highly impacts the objective function's value. This, in turn, helps skipping local solutions and change the matching. Accordingly, B_{imp} is defined 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 vertex operation cost induced by assigning vertex i to vertex j, and θ_{ij} is the 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 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 $[\sigma_i]$. The values in $[\sigma_i]$ are split into two clusters *min* and *max*. Finally, for every σ_i belonging to *max* cluster, the indexes of all binary variables that represent a vertex assignment for i are added to B_{imp}. 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 [5].

4 Computational Experiment

Database: There are many graph databases that represent chemical molecules e.g. MUTA, PAH, MAO [1] in the literature. In the current experiment, MUTA database is chosen because it contains different subsets of small and large graphs and known to be difficult. There are 7 subsets, each of which has 10 graphs of same size (10 to 70 vertices) and a subset of also 10 graphs with mixed graph sizes. Each pair of graphs is considered as an instance. Therefore, a total of 800 instances (100 per subset) are considered in this experiment.

Comparative heuristics and experiment settings: To solve the $MILP^{JH}$, the solver CPLEX 12.6.0 is used in mixed-integer programming mode. *LocBra_GED* algorithm is implemented in C language. The tests were executed on a machine with the following configuration: Windows 7 (64-bit), Intel Xeon E5 4 cores and 8 GB RAM. *LocBra_GED* is applied on all instances with the following parameter values: $k = 20$, $k\_dv = 30$, $total\_time\_limit = 900s$, $node\_time\_limit = 180s$. Since *LocBra_GED* uses CPLEX with time limit, it is interesting to study the performance of the solver itself with the same time limit in order to see whether the proposed heuristic actually improves the solution of the problem, the method is called *CPLEX-900s*. Other competitive heuristics are included in the experiment such as *BeamSearch* and *SBPBeam*. Both are based on beam-search method, so their performance varies based on the choice of the beam size, therefore two versions of each are considered: *BeamSearch 5*, *BeamSearch 15000*, *SBPBeam 5* and *SBPBeam 400*. All parameter values are set based on preliminary experiments that are not shown here. For instance, beam sizes 15000 and 400 increases the execution time of *BeamSearch* and *SBPBeam* to be around $900s$. For each heuristic, the following values are computed for each subset of graphs: t_{avg} is the average CPU time in seconds for all instances. Then, $d_{min}, d_{avg}, d_{max}$ are the deviation percentages between the solutions obtained by one heuristic, and the optimal solutions. Given an instance I and a heuristic H, deviation percentage is equal to $\frac{solution_I^H - optimal_I}{optimal_I} \times 100$, with $optimal_I$ is the optimal solution for I. Lastly, η_I represents the number of solutions obtained by a heuristic that are equal to the optimal ones.

Results and analysis: Table 1 shows the results of the experiment. Looking at η_I, *LocBra_GED* and *CPLEX 900s* heuristics seem to have the highest values among the others. Moreover, *LocBra_GED* has higher values than *CPLEX*

Table 1. *LocBra_GED* vs. literature heuristics on MUTA instances.

	S	10	20	30	40	50	60	70	Mixed
	opt_I	100	100	100	99	92	71	35	91
LocBra_GED	t_{avg}	0.17	1.12	212.36	359.45	552.21	693.54	474.97	276.79
	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	**0.00**	**0.33**	**0.76**	**0.22**
	d_{max}	0.00	0.00	0.00	3.90	0.00	3.57	8.57	3.43
	η_I	**100**	**100**	**100**	97	**92**	**63**	**28**	84
CPLEX 900s	t_{avg}	0.13	1.02	141.07	241.21	412.36	651.43	458.20	246.90
	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**	0.29	0.57	1.80	0.29
	d_{max}	0.00	0.00	0.00	0.00	6.42	6.86	8.61	6.86
	η_I	**100**	**100**	**100**	**99**	84	58	22	**87**
BeamSearch 5	t_{avg}	**0.00**	**0.00**	**0.01**	**0.03**	**0.07**	**0.11**	**0.16**	**0.08**
	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	58.73	73.73	66.47	57.76	27.02
	d_{max}	110.00	124.59	147.37	186.67	200.00	146.37	210.71	120.00
	η_I	35	10	10	10	10	10	10	11
BeamSearch 15000	t_{avg}	8.57	80.65	167.48	278.70	436.20	610.82	777.43	782.78
	d_{min}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-
	d_{avg}	1.35	26.66	47.45	52.53	65.77	66.34	78.36	-
	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_{avg}	0.01	0.10	0.45	1.37	3.18	5.54	10.91	2.90
	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	82.74	101.63	101.39	107.02	30.31
	d_{max}	90.00	127.87	206.90	204.71	314.29	198.50	280.36	133.71
	η_I	15	10	10	10	10	10	10	10
SBPBeam 400	t_{avg}	0.84	10.02	47.65	139.78	321.21	587.96	1144	279.17
	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	82.74	101.63	101.39	107.02	30.00
	d_{max}	90.00	127.87	206.90	204.71	314.29	198.50	280.36	133.71
	η_I	15	10	10	10	10	10	10	10

900s for all subsets except Mixed with 87 against 84 solutions equal to the optimal. Now considering the d_{avg}, again *LocBra_GED* and *CPLEX 900s* has the smallest deviations w.r.t. the optimal solutions (always less than 2%). For easy instances (subsets 10 to 40), both heuristics have scored 0%, except for subset 40 where *LocBra_GED* has 0.06% average deviation, but still very close to *CPLEX 900s*. For hard instances (subsets 50 to 70), *LocBra_GED* has the lowest deviations (always less than 1%). The same conclusion can be seen when checking the deviations for Mixed subset. The d_{avg} values for beam-search based methods are very high comparing to the first two heuristics, where on hard instances they reach over 100% (e.g. *SBPBeam* for subset 70 has 107%). Regarding the execution time of the heuristics, *BeamSearch 5* is the fastest with t_{avg} always less than $0.2s$, but in terms of solution quality it is very far from the optimal with d_{avg} up to 73%. And even after increasing the beam size for both *BeamSearch* and *SBPBeam* both are not able to provide better solutions and the average deviations remain very high. *BeamSearch 15000* was not applicable on subset Mixed because it did not return feasible solutions for all the instances, so the deviations and η_I are not computed. In summary, *LocBra_GED* is the most accurate and has provided near optimal solutions.

5 Conclusion

In this work, a local branching heuristic is proposed for the *GED* problem. As a conclusion, *LocBra_GED* significantly improves the literature heuristics and provides near optimal solutions. This is basically due, to the analysis and the branching scheme combined with the efficiency of CPLEX when solving $MILP^{JH}$ model. A second important factor is the diversification procedure that is problem dependent and really helps escaping local optima solutions. Next, more heuristic algorithms will be tested against *LocBra_GED*, and more techniques will be investigated in order to boost the solution of the method and also to allow dealing with graphs that have attributes on their edges.

References

1. Abu-Aisheh, Z., Raveaux, R., Ramel, J.-Y.: A graph database repository and performance evaluation metrics for graph edit distance. In: Liu, C.-L., Luo, B., Kropatsch, W.G., Cheng, J. (eds.) GbRPR 2015. LNCS, vol. 9069, pp. 138–147. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-18224-7_14
2. Bougleux, S., Brun, L., Carletti, V., Foggia, P., Gaüzère, B., Vento, M.: Graph edit distance as a quadratic assignment problem. Pattern Recogn. Lett. **87**, 38–46 (2016)
3. Bunke, H.: On a relation between graph edit distance and maximum common subgraph. Pattern Recogn. Lett. **18**(8), 689–694 (1997)
4. Ferrer, M., Serratosa, F., Riesen, K.: Improving bipartite graph matching by assessing the assignment confidence. Pattern Recogn. Lett. **65**, 29–36 (2015)
5. Fischetti, M., Lodi, A.: Local branching. Math. Program. **98**(1–3), 23–47 (2003)
6. Justice, D., Hero, A.: A binary linear programming formulation of the graph edit distance. IEEE Trans. Pattern Anal. Mach. Intell. **28**(8), 1200–1214 (2006)
7. Lerouge, J., Abu-Aisheh, Z., Raveaux, R., Héroux, P., Adam, S.: Exact graph edit distance computation using a binary linear program. In: Robles-Kelly, A., Loog, M., Biggio, B., Escolano, F., Wilson, R. (eds.) S+SSPR 2016. LNCS, vol. 10029, pp. 485–495. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49055-7_43
8. Neuhaus, M., Riesen, K., Bunke, H.: Fast suboptimal algorithms for the computation of graph edit distance. In: Yeung, D.-Y., Kwok, J.T., Fred, A., Roli, F., de Ridder, D. (eds.) SSPR /SPR 2006. LNCS, vol. 4109, pp. 163–172. Springer, Heidelberg (2006). https://doi.org/10.1007/11815921_17

9. Raymond, J.W., Willett, P.: Maximum common subgraph isomorphism algorithms for the matching of chemical structures. J. Comput. Aided Mol. Des. **16**(7), 521–533 (2002)
10. Riesen, K., Neuhaus, M., Bunke, H.: Bipartite graph matching for computing the edit distance of graphs. In: Escolano, F., Vento, M. (eds.) GbRPR 2007. LNCS, vol. 4538, pp. 1–12. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72903-7_1
11. Serratosa, F.: Fast computation of bipartite graph matching. Pattern Recogn. Lett. **45**, 244–250 (2014)
12. Stauffer, M., Tschachtli, T., Fischer, A., Riesen, K.: A survey on applications of bipartite graph edit distance. In: Foggia, P., Liu, C.-L., Vento, M. (eds.) GbRPR 2017. LNCS, vol. 10310, pp. 242–252. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-58961-9_22
13. Zeng, Z., Tung, A.K., Wang, J., Feng, J., Zhou, L.: Comparing stars: on approximating graph edit distance. Proc. VLDB Endow. **2**(1), 25–36 (2009)