Approximating GED Using a Stochastic Generator and Multistart IPFP

Nicolas Boria(✉), Sébastien Bougleux, and Luc Brun

Normandie Univ, UNICAEN, ENSICAEN, CNRS, GREYC, Caen, France
{boria,luc.brun}@ensicaen.fr, bougleux@unicaen.fr

Abstract. The Graph Edit Distance defines the minimal cost of a sequence of elementary operations transforming a graph into another graph. This versatile concept with an intuitive interpretation is a fundamental tool in structural pattern recognition. However, the exact computation of the Graph Edit Distance is $\mathcal{NP}$-complete. Iterative algorithms such as the ones based on Franck-Wolfe method provide a good approximation of true edit distance with low execution times. However, underlying cost function to optimize being neither concave nor convex, the accuracy of such algorithms highly depends on the initialization. In this paper, we propose a smart random initializer using promising parts of previously computed solutions.

Keywords: Graph edit distance · Parallel gradient descents
Multistart · Stochastic warm start

1 Introduction

Computing a similarity or a dissimilarity measure between graphs is a major challenge in pattern recognition. One of the most well-known and used approach to compute a distance between two graphs is the Graph Edit Distance (GED) [12]. Computing the GED consists in finding a sequence of graph edit operations (insertions, deletions and substitutions of vertices and edges) that transforms a graph into another with a minimal cost. Such a sequence of edit operations is called an edit path, and the edit distance between two graphs G and H is defined by $GED(G,H) = min_{\gamma \in \Gamma(G,H)} \sum_{e \in \gamma} c(e)$, where $\Gamma(G,H)$ denotes the set of edit paths between G and H and $c(e)$ denotes the cost of an elementary operation e belonging to the edit path γ.

If both graphs are simple and if the cost between vertices and edges remains fixed, one can show [3] that the edit distance between two graphs G and H of respective orders n and m may be formulated as the following quadratic problem: $GED(G,H) = \min_{x \in \Pi_{n,m}} \frac{1}{2} x^t \Delta x + c^t x$, where $\Pi_{n,m}$ denotes the set of vectorized assignment matrices between V_G and V_H. Such a matrix x encodes for each element of V_G one and only one operation (either substitution or deletion).

Work supported by Region Normandie under project RIN AGAC.

X. Bai et al. (Eds.): S+SSPR 2018, LNCS 11004, pp. 460–469, 2018.
https://doi.org/10.1007/978-3-319-97785-0_44

In the same way, x encodes for each element of V_H either a substitution or an insertion. The matrix Δ encodes the cost of edge operations while c encodes the cost of operations on vertices. Computing the GED is NP-hard, so several heuristics were proposed to compute approximate solutions in polynomial time.

The design of approximate solutions to the GED problem has been strongly stimulated by the introduction of an approximation of the GED problem into a Linear Sum Assignment Problem with Edition (LSAPE) [9]. This approximation consists in associating to each node of two graphs G and H a substructure and to populate a cost matrix encoding: the costs of matching two substructures, the cost of inserting one substructure in H and the cost of removing one substructure from G. Given such a cost matrix $\tilde{c}$, the assignment matrix x minimizing $\tilde{c}^t x$ provides a set of elementary operations on vertices from which an edit path may be deduced. The cost of this edit path provides an upper bound for the graph edit distance. This minimization step may be solved in polynomial time. This transformation of an $\mathcal{NP}$-complete problem into a minimization problem with a polynomial complexity is the major advantage of the LSAPE approximation. However the computation of the cost matrix $\tilde{c}$ may require non polynomial execution times. Different types of substructures have been defined in [4,8,9,14].

However, LSAPE is based on a linear approximation of a quadratic problem. In order to get a finer approximation of the graph edit distance, several methods use variants of local search such as simulated annealing in order to improve an initial estimation of the edit distance [6,10,11]. A slightly different approach [3], consists in using the Franck Wolfe minimization scheme [7] from an initial guess. This algorithm converges by iterations towards a local minimum of the quadratic function usually close from the initial guess. This heuristic provides close approximations of the graph edit distance on small graphs but is sensitive to the solution used to initialize the method. An heuristic to reduce the influence of the initial guess has been proposed in [5]. This heuristic is based on the use of multiple initial guesses either deduced from a set of solutions of the LSAPE problem or based on the generation of random assignment matrices. The common drawback of these two heuristics is that the generation of initial solutions does not take into account information provided by runs of Franck-Wolfe method which may have converged.

In this paper we propose a new heuristic based on alternative runs of the generation of initial solutions and the determination of their associated local minima using Franck-Wolfe method. This method is described in Sect. 3. The proposed method is evaluated in Sect. 4 through several experiments.

2 Preliminaries

Throughout the paper, we will use the same concepts and notations as those introduced in [3]. Vertices of graphs G and H are numbered respectively from 1 to n and from 1 to m, and two virtual vertices indexed as $n+1$ in G and as $m+1$ in H are added. These virtual vertices, denoted by ϵ_G and ϵ_H, correspond respectively to insertions and deletions. An assignment $i \rightarrow \epsilon_H$ (resp. $\epsilon_G \rightarrow j$) corresponds to the deletion of vertex i of G (resp. insertion of vertex j of H).

begin
  Minimize a linear approximation of Q around the current solution $\mathbf{x}$ in the discrete domain by solving an LSAP:$\mathbf{b}^\star \leftarrow \underset{\mathbf{b}\in\pi_{n,m,\epsilon}}{\operatorname{argmin}} (\mathbf{x}^\top \Delta)\mathbf{b}$
  Perform the descent by minimizing Q along the segment $[\mathbf{x}, \mathbf{b}]$ in the continuous domain: $\alpha^\star \leftarrow \underset{\alpha\in[0,1]}{\operatorname{argmin}} Q(\mathbf{x} + \alpha(\mathbf{b}^\star - \mathbf{x}))$
  Update $x : \mathbf{x} \leftarrow \mathbf{x} + \alpha^\star(\mathbf{b}^\star - \mathbf{x})$
  Repeat steps 2 to 4 until $\mathbf{x}^T\Delta(\mathbf{x} - \mathbf{b}^\star) < \beta\left|Q(\mathbf{x}) + \mathbf{x}^T\Delta(\mathbf{b}^\star - \mathbf{x})\right|$ holds for a given scalar $\beta \in (0, 1)$, or if a given number of iterations is reached

Algorithm 1. FW$(\Delta, \mathbf{x})$

In this context, a solution for GED will be described as an *error-correcting assignment matrix* x, where all vertices of G are assigned to a single element of $H \cup \{\epsilon_H\}$, and all vertices of H are assigned to a single element of $G \cup \{\epsilon_G\}$. The polytope $\Pi^\epsilon_{n,m}$ of *error-correcting assignment matrices* thus contains all matrices of dimensions $(n+1)\times(m+1)$, with binary values, and with a single 1 in each line and in each column, except for the last line and the last column. We naturally extend the concept to matrices with fractional values: we call *error-correcting bistochastic matrix* any matrix where the sum of all cells for each line except the last one and each column except the last one amounts exactly to 1.

Given a cost matrix Δ and an initial continuous or discrete candidate solution x, Algorithm 1 describes Frank-Wolfe algorithm FW$(\Delta, \mathbf{x})$. In the following, we denote by IPFP the method that consists in running FW$(\Delta, \mathbf{x})$ and projecting the returned solution in the discrete space. See [3] for more details.

3 RANDPOST Algorithm

The conception of algorithm RANDPOST finds its origin in the following intuition regarding the - often conflicting - criteria that a smart initial solution generator should fulfill. On the one hand, a smart generator should propose solutions that are well distributed inside the $\Pi^\epsilon_{n,m}$ polytope, in order to increase the diversity among local minima that are ultimately returned. We call this criterium *exploration criterium*. On the other hand, we obviously wish that one of the initial solutions ultimately led to a global minimum, so that a good generator should somehow already generate solutions that includes "smart" assignments. We call this criterium *quality criterium*.

Building up on the progresses of the multistart IPFP (mIPFP) algorithm [5], we propose a new algorithm that explores the polytope and generates new solutions by taking advantage of the whole statistical information contained in the set of solutions returned by mIPFP. This algorithm is presented in Sect. 3.2. We also devise a new parameterized random generator that is described in Sect. 3.1.

3.1 Generating Initial Solutions with Parameterized Number of Insertions and Deletions

With respect to the initial random generator used in [5], where vertices were randomly assigned by the use of the `std::random_shuffle` procedure of the C++ standard library, resulting in a random proportion of insertions and deletions, we decided to use a random generator with a parameterized proportion of insertions and deletions.

We focus our analysis on the number of deletions, as - for given $n = |G|$ and $m = |H|$ - the number of deletions is completely determined by it, namely, $\#\text{insertions} \equiv \#\text{deletions} - n + m$.

Namely, given a parameter $\alpha \in (0, 1)$, we used a new initial random generator `RANDGEN`(α) which generates solutions with an expected number of deletions equal to $\alpha n + (1 - \alpha) \max\{(n - m), 0\}$. In other words, α denotes the expected proportion of "unnecessary" deletions of vertices in the set of initial solutions, reminding that if $n > m$ any feasible assignment will have to assign at least $n - m$ vertices to the virtual node ε_H which thus correspond "necessary" to deletions.

3.2 Stochastic Generation of New Initial Solutions Based on Several Refined Solutions

We describe here the functioning of `RANDPOST` (Algorithm 2). Given a pair of graphs G and H, the algorithm starts by running mIPFP, which outputs a set S^* of r solutions for the problem. Each of these r solutions is represented as an error-correcting assignment matrix x. A matrix Ψ is then created by simply summing all of these r matrices and dividing all values by r. Hence, $\forall i \in [1, n + 1], j \in [1, m + 1]$, $\Psi_{i,j}$ represents the proportion of solutions where i is assigned to j among the r solutions of S^*. Matrix Ψ (which is an error-correcting bistochastic matrix) is then used as a probability distribution to compute a new set of k solutions which are subsequently refined using IPFP, and the first r that have converged among these k parallel processes are added to the set S^*. The whole

```
1 begin
2     Generate an initial set S ⊂ D_{n,m,ε} of k solutions using RANDGEN(α)
3     Start refining all solutions in S using IPFP, and stop the refinement process
        when r of them have converged, which results in a set of r improved
        solutions S*
4     for i=1 to l do
5         Update matrix Ψ in the following way: Ψ ← Σ_{x∈S*} x/|S*|.
6         Generate a new set S of k solutions using a random generator that uses
            the probability distribution described by Eq. (1).
7         Start refining all solutions in S using IPFP, and add the r first to have
            converged to S*
```

Algorithm 2. `RANDPOST`(k, r, l)

sequence that updates Ψ, generates new solutions and finally refines them is repeated l times.

Parameters k and r enables to speed up the algorithm when $k \gg r$: some of the k initial solutions might require many iterations of IPFP in order to converge to a local minimum, so that by launching k parallel IPFPs and stopping all them when r of them have converged, the whole process is likely to be faster than launching r parallel IPFPs and waiting for all of them to converge.

The intuitive idea behind algorithm `RANDPOST`is the following: if an assignment $i \rightarrow j$ appears with a high frequency within solutions of the set S^*, then this assignment is likely to be part of many good solutions for the problem at hand. Hence, the algorithm generates k new solutions in a stochastic way, where the probability for a given assignment to be part of a solution is higher for assignments with high $\Psi_{i,j}$ value, and thus high frequency.

To be even more precise, the random generator assigns each vertex of G to a vertex of $H \cup \{\epsilon_H\}$ following a greedy procedure. The matrix x of dimensions $(n+1) \times (m+1)$ (which will eventually contain the solution returned by the algorithm) is initialized with zeros, and whenever an assignment $i \rightarrow j$ is made by the algorithm, this translates to $x_{i,j} \leftarrow 1$. At the end of the algorithm x should be an error-correcting permutation. The random generator assigns iteratively vertices from 1 to n based on the following probabilistic distribution P_i, where $P_i(j)$ denotes the probability for vertex i of G to be assigned to vertex j of H by the generator, given a partial assignment of vertices from 1 to $i-1$:

$$\begin{array}{l} \forall j = 1, ..., m+1, \quad P_1(j) = \Psi(j) \\ \forall i = 2, \ldots, n, \forall j = 1, ..., m+1 \\ P_i(j) = \begin{cases} 0 & \text{if } j \neq m+1 \text{ and } \exists h < i, \text{s.t } x_{h,j} = 1 \\ \dfrac{\Psi_{i,j}}{1 - \sum_{h=1}^{m} \left(\sum_{l=1}^{i-1} x_{h,l} \Psi_{i,l} \right)} & \text{otherwise} \end{cases} \end{array} \tag{1}$$

Finally, all vertices of H that have been left unassigned at the end of the procedure are all assigned to ϵ_G. First let us prove that matrix x produced by the proposed random generator is always an error-correcting permutation matrix. This is done by putting together the two following facts: (1) by construction, each line of x but the last one has exactly one value set to 1 and all the others set to 0 (a single assignment is made for a single line at each step of the generator), (2) the probability distribution described by (1) ensures that no vertex j of H is assigned twice (once assigned, its probability of being assigned again is zero), and the very last step ensures that each one is assigned at least once.

Let us briefly prove that (1) defines a proper probability distribution. It is easy to verify that $\sum_{j=1}^{m+1} P_1(j) = 1$ by simply reminding that Ψ is an error-correcting bistochastic matrix. For $i = 2, \ldots, n$, consider a matrix $\tilde{\Psi}$ which values are as follows:

$$\tilde{\Psi}_{i,j} = \begin{cases} 0 & \text{if } j \neq m+1 \text{ and } \exists h < i, \text{s.t } x_{h,j} = 1 \\ \Psi_{i,j} & \text{otherwise} \end{cases}$$

It is easy to verify that:

$$\forall i = 2, \ldots, n \quad \sum_{j=1}^{m+1} \tilde{\Psi}_{i,j} = 1 - \sum_{h=1}^{m} \left(\sum_{l=1}^{i-1} x_{h,l} \Psi_{i,l} \right) \tag{2}$$

We finally prove that (1) defines a proper probability distribution:

$$\forall i = 2, \ldots, n \quad \sum_{j=1}^{m+1} P(i \to j) = \frac{\sum_{j=1}^{m+1} \tilde{\Psi}_{i,j}}{1 - \sum_{h=1}^{m} \left(\sum_{l=1}^{i-1} x_{h,l} \Psi_{i,l} \right)} \overset{(2)}{=} 1$$

Finally, whenever the stochastic generator produces a candidate solution that has already been generated earlier, the solution is discarded and a new solution is produced using a slightly flatter (and thus more explorative) distribution.

4 Experimental Results

In this section, we evaluate the proposed method through several experiments, in order to determine as clearly as possible the relevance and importance of the *exploration* and *quality* criteria described in Sect. 3.

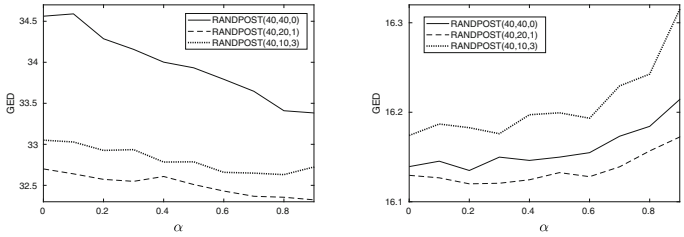
4.1 Datasets and Protocol

Table 1 presents the chemical datasets that were used in our experiments. MAO, PAH, MUTA10-70 and MUTAmix were considered in ICPR 2016 – Graph Distance Contest [2]. We also extracted 25 graphs from ClinTox [13], and 10 graphs with more than 100 vertices from MUTA.

Table 1. Characteristics of datasets

Dataset	#graphs	Avg order	Labels on nodes/edges
MAO	68	18.4	Labeled
PAH	94	20.7	Unlabeled
MUTA10–70	10	10–70	Labeled
MUTAmix	10	45	Labeled
MUTA100+	10	131.6	Labeled
ClinTox	25	115.7	Labeled

We evaluated the four following versions of `RANDPOST`(k, r, l): `RANDPOST`$(40, 40, 0)$, `RANDPOST`$(40, 20, 1)$, `RANDPOST`$(40, 10, 3)$ and `RANDPOST`$(40, 5, 7)$. This choice of parameters is determined by the following idea: considering two algorithms `RANDPOST`(k, r_1, l_1) and `RANDPOST`(k, r_2, l_2) such that $r_1(l_1+1) = r_2(l_2+1)$ and $r_1 > r_2$, their relative performances can be compared without bias as the



(a) MAO, metric costs c_4 (b) MAO, anti-metric costs c_3

Fig. 1. Behavior of `RANDPOST` w.r.t. parameter α, metric vs. anti-metric costs

overall number of candidate solutions is the same (in our case, all four algorithms generate exactly 40 candidates), while the latter algorithm performs better on the quality criteria, and the former on the exploration one. We thus consider that r represents the exploration parameter, while l represents the quality one.

Regarding cost functions, we tested all algorithms with four different sets of costs: c_1, c_2 and c_3 correspond to the costs used in [2], while c_4 is the cost function used in [5] and references therein. Note that c_1, c_2 and c_4 correspond to metric costs where a substitution cost of two elements is lower or equal to the cost of the removal of the first element together with the insertion of the second one. Conversely, c_3 is an anti-metric cost violating this last inequality. The main idea between these two classes of cost functions is that metric cost functions favor substitutions while the anti-metric ones favor deletions and insertions.

All tests were performed using 4 AMD Opteron processors at 2.6 Ghz with 512G of RAM. The number of parallel threads was limited to 40 (which corresponds to parameter k). The code for the algorithm is written in C++.

4.2 Behavior of the Algorithm w.r.t. Parameter α

We tested three versions of `RANDPOST` with several values for parameter α of `RANDGEN`, and several cost functions (see the previous section). The most significant results are presented in Fig. 1 for MAO, a dataset with enough and relatively simple instances so that interesting statistical tendencies can emerge. Contrasting tendencies can be observed with the metric cost function c_4 and the anti-metric one c_3. Interestingly, the algorithm performs better as the initial proportion of "unfavored" choice rises. We believe that this is due to the design of the IPFP gradient descent, which is likely to find a better local minimum when starting from a solution including a greater number of "neutral" (in the sense of easily improvable) assignments.

Unfortunately, the behavior that we observe on MAO does not emerge with the same clarity on more complex datasets containing bigger or unlabeled graphs. However, it seems that IPFP requires a medium value for α (around 0.4) to perform best when dealing with unlabeled graphs. For bigger graphs (more than 40 vertices) high values of α seem to produce better starting points for IPFP, independently of cost functions.

Table 2. Experimental results of RANDPOST(k, r, l), cost c1

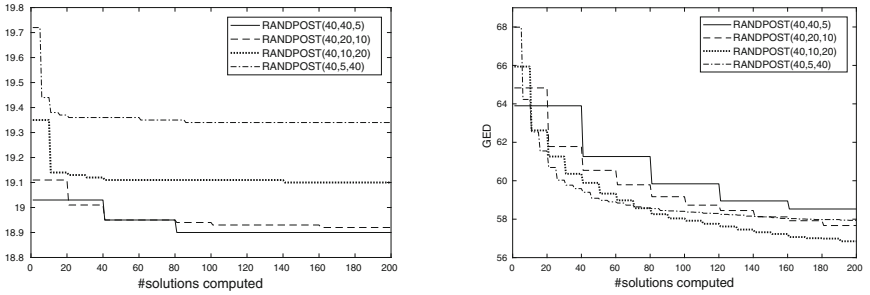
Algorithms	MAO $\alpha = 0$				PAH $\alpha = 0.3$				ClinTox $\alpha = 0.9$			
	Time	GED	err.	%best	Time	GED	err.	%best	Time	GED	err.	%best
RANDPOST(40, 1, 0)	0.013	34.43	10.30	25	0.013	36.94	24.82	1	3.542	209.42	52.12	0
RANDPOST(40, 40, 0)	0.074	24.16	0.03	98	0.099	21.23	9.11	19	17.205	167.76	10.46	2
RANDPOST(40, 20, 1)	0.029	**24.14**	**0.01**	**100**	0.038	20.71	8.59	27	13.330	163.18	5.88	10
RANDPOST(40, 10, 3)	0.051	24.19	0.06	98	0.063	**20.42**	**8.30**	**33**	19.514	160.24	2.94	30
RANDPOST(40, 5, 7)	0.144	24.48	0.35	89	0.116	20.90	8.78	26	29.278	**157.98**	**0.69**	**76**
Algorithms	MUTA 10 $\alpha = 0$				MUTA 20 $\alpha = 0.2$				MUTA 30 $\alpha = 0.8$			
	Time	GED	err.	%best	Time	GED	err.	%best	Time	GED	err.	%best
RANDPOST(40, 1, 0)	0.013	13.19	1.21	60	0.012	33.35	14.49	23	0.027	73.80	49.51	5
RANDPOST(40, 40, 0)	0.020	**11.98**	**0.00**	**100**	0.080	19.00	0.14	86	0.235	25.68	1.39	42
RANDPOST(40, 20, 1)	0.028	**11.98**	**0.00**	**100**	0.041	**18.96**	**0.10**	**91**	0.128	25.28	0.99	51
RANDPOST(40, 10, 3)	0.062	**11.98**	**0.00**	**100**	0.062	19.03	0.17	89	0.181	**25.07**	**0.78**	**61**
RANDPOST(40, 5, 7)	0.148	12.01	0.03	97	0.153	19.33	0.47	73	0.452	25.51	1.22	51
Algorithms	MUTA 40 $\alpha = 0.6$				MUTA 50 $\alpha = 0.9$				MUTA 60 $\alpha = 0.9$			
	Time	GED	err.	%best	Time	GED	err.	%best	Time	GED	err.	%best
RANDPOST(40, 1, 0)	0.063	83.94	50.23	2	0.123	81.67	44.83	5	0.246	98.55	51.97	5
RANDPOST(40, 40, 0)	0.575	36.07	2.36	26	1.141	40.10	3.26	20	2.120	50.64	4.06	11
RANDPOST(40, 20, 1)	0.302	35.00	1.29	46	0.565	38.56	1.72	31	1.158	48.95	2.37	24
RANDPOST(40, 10, 3)	0.391	**34.31**	**0.60**	**67**	0.886	**37.57**	**0.73**	**61**	1.862	47.69	1.11	54
RANDPOST(40, 5, 7)	0.516	34.85	1.14	53	1.465	37.84	1.00	55	3.133	**47.33**	**0.75**	**56**
Algorithms	MUTA 70 $\alpha = 0.9$				MUTA 100+ $\alpha = 0.9$				MUTAmix $\alpha = 0.9$			
	Time	GED	err.	%best	Time	GED	err.	%best	Time	GED	err.	%best
RANDPOST(40, 1, 0)	0.528	84.18	25.80	6	3.181	259.28	37.88	0	0.111	155.71	21.68	6
RANDPOST(40, 40, 0)	3.641	63.90	5.52	12	19.67	234.24	12.84	1	0.848	136.08	2.05	42
RANDPOST(40, 20, 1)	2.559	61.45	3.07	25	12.39	227.49	6.09	9	0.455	135.16	1.13	57
RANDPOST(40, 10, 3)	3.573	59.93	1.55	49	18.51	224.50	3.10	34	0.634	**134.75**	**0.72**	**68**
RANDPOST(40, 5, 7)	8.181	**59.44**	**1.06**	**56**	28.70	**222.15**	**0.75**	**78**	1.117	135.06	1.03	55

4.3 Performance of RANDPOST

Table 2 presents the performance of the four versions of RANDPOST(k, l) that we mentioned earlier, plus RANDPOST$(1, 0)$ which corresponds to a single run of IPFP starting from a random candidate solution. For each pair of graphs in each dataset, we extracted the best known GED among those returned by a set of 14 algorithms (9 algorithms of [2] + 5 versions of RANDPOST), except for ClinTox and MUTA100+ that weren't part of the benchmark in [1]. For these two datasets, the best GED was extracted from our 5 algorithms alone. The "err." column presents the mean error w.r.t. best known solutions, while the "%best" column presents the proportion of pairs of graphs for which the best known GED was found. For each dataset, we selected the value of α leading to a minimal mean GED over all 5 tested algorithms. The selected value is indicated in the table. Due to space restrictions, we present results regarding the metric cost $c1$ only. The same tendencies can be observed with all the other cost functions.

The tendencies that emerge from Table 2 are quite clear: the more qualitative versions of RANDPOST(k, r, l) perform better than all algorithms presented in [2] on datasets with labeled graphs containing at least 60 vertices.

Under this threshold, the balance between exploration and quality criteria that performs better GED favors more exploratory methods as the size of the graphs decreases. Further analysis shows that the phenomenon is deeply linked to the speed and quality of convergence of the algorithms: a more exploratory version of RANDPOSTwill ultimately converge to better GED estimations, but it will also converge at a slower rate. On the other hand, bigger graphs lead to slower overall convergence rates. These two phenomenons are visible in Fig. 2. Both plots represent the improvement in GED estimations over the successive loops of RANDPOST. Each stairstep measures the best GED computed in a loop, and as the x-axis represents the number of computed solution, the length of the steps equals r for each algorithm RANDPOST(k, r, l).



(a) MUTA-20 (b) MUTA-70

Fig. 2. Convergence of RANDPOSTon datasets MUTA-20 and MUTA-70

When dealing with smaller graphs, qualitative methods converge very rapidly to suboptimal solutions, while the exploratory ones converge more slowly to better GED estimations. On the other hand, when dealing with bigger graphs, the fast rate of convergence of qualitative methods becomes a strength rather than a flaw, Fig. 2b shows that when the number of computed solutions is limited to 40 (which corresponds to the results in Table 2), none of the algorithms has yet converged, so that the faster converging algorithm yields better results. This phenomenon eventually reverses on the long run: as an example, Fig. 2b suggests that the limit on the number of computed solutions must be brought as high as 90 for RANDPOST$(40, 10, 20)$ to outperform RANDPOST$(40, 5, 40)$ on MUTA70.

5 Conclusion

Using a new iterative IPFP-based algorithm relying on stochastically generated solutions, we investigated the relative importance of exploration and quality criteria when generating candidate solutions for a multistart version of IPFP. Our results suggest that the balance leading to better GED estimations depends mostly on some ratio between the dimension of the problem at hand and the overall number of generated solutions.

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. Abu-Aisheh, Z., et al.: Graph edit distance contest: results and future challenges. Pattern Recogn. Lett. **100**, 96–103 (2017). https://doi.org/10.1016/j.patrec.2017.10.007
3. Bougleux, S., Brun, L., Carletti, V., Foggia, P., Gaüzère, B., Vento, M.: Graph edit distance as a quadratic assignment problem. Patt. Recogn. Lett. **87**, 38–46 (2017). https://doi.org/10.1016/j.patrec.2016.10.001
4. Carletti, V., Gaüzère, B., Brun, L., Vento, M.: Approximate graph edit distance computation combining bipartite matching and exact neighborhood substructure distance. In: Liu, C.-L., Luo, B., Kropatsch, W.G., Cheng, J. (eds.) GbRPR 2015. LNCS, vol. 9069, pp. 188–197. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-18224-7_19
5. Daller, E., Bougleux, S., Gaüzère, B., Brun, L.: Approximate graph edit distance from several assignments and multiple IPFP. In: International Conference on Pattern Recognition Applications and Methods (2018). https://doi.org/10.5220/0006599901490158
6. Ferrer, M., Serratosa, F., Riesen, K.: A first step towards exact graph edit distance using bipartite graph matching. In: Liu, C.-L., Luo, B., Kropatsch, W.G., Cheng, J. (eds.) GbRPR 2015. LNCS, vol. 9069, pp. 77–86. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-18224-7_8
7. Frank, M., Wolfe, P.: An algorithm for quadratic programming. Nav. Res. Logist. Q. **3**(1–2), 95–110 (1956)
8. Gaüzère, B., Bougleux, S., Brun, L.: Approximating graph edit distance using GNCCP. In: Robles-Kelly, A., Loog, M., Biggio, B., Escolano, F., Wilson, R. (eds.) S+SSPR 2016. LNCS, vol. 10029, pp. 496–506. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49055-7_44
9. Riesen, K., Bunke, H.: Approximate graph edit distance computation by means of bipartite graph matching. Image Vis. Comput. **27**, 950–959 (2009). https://doi.org/10.1016/j.imavis.2008.04.004
10. Riesen, K., Bunke, H.: Improving bipartite graph edit distance approximation using various search strategies. Pattern Recogn. **48**(4), 1349–1363 (2015). https://doi.org/10.1016/j.patcog.2014.11.002
11. Riesen, K., Fischer, A., Bunke, H.: Improved graph edit distance approximation with simulated annealing. In: Foggia, P., Liu, C.-L., Vento, M. (eds.) GbRPR 2017. LNCS, vol. 10310, pp. 222–231. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-58961-9_20
12. Sanfeliu, A., Fu, K.S.: A distance measure between attributed relational graphs for pattern recognition. IEEE Trans. Syst. Man Cybern. **13**(3), 353–362 (1983). https://doi.org/10.1109/TSMC.1983.6313167
13. Wu, Z., et al.: MoleculeNet: a benchmark for molecular machine learning. Chem. Sci. **9**, 513–530 (2018). https://doi.org/10.1039/C7SC02664A
14. Zeng, Z., Tung, A.K.H., Wang, J., Feng, J., Zhou, L.: Comparing stars: on approximating graph edit distance. Proc. VLDB Endow. **2**(1), 25–36 (2009). https://doi.org/10.14778/1687627.1687631