GEDLIB: A C++ Library for Graph Edit Distance Computation

David B. Blumenthal[1](✉), Sébastien Bougleux[2], Johann Gamper[1], and Luc Brun[2]

[1] Faculty of Computer Science, Free University of Bozen-Bolzano, Bolzano, Italy
{david.blumenthal,gamper}@inf.unibz.it

[2] Normandie Univ, UNICAEN, ENSICAEN, CNRS, GREYC, Caen, France
bougleux@unicaen.fr, luc.brun@ensicaen.fr

Abstract. The graph edit distance (GED) is a flexible graph dissimilarity measure widely used within the structural pattern recognition field. In this paper, we present GEDLIB, a C++ library for exactly or approximately computing GED. Many existing algorithms for GED are already implemented in GEDLIB. Moreover, GEDLIB is designed to be easily extensible: for implementing new edit cost functions and GED algorithms, it suffices to implement abstract classes contained in the library. For implementing these extensions, the user has access to a wide range of utilities, such as deep neural networks, support vector machines, mixed integer linear programming solvers, a blackbox optimizer, and solvers for the linear sum assignment problem with and without error-correction.

Keywords: Graph edit distance · Open source library · C++

1 Introduction

Because of their expressiveness and versatility, labeled graphs are widely used to model various kinds of objects such as molecules, street networks, and images. Many pattern recognition problems defined over these domains presuppose the availability of a (dis-)similarity measure for labeled graphs. Despite the fact that its exact computation is $\mathcal{NP}$-hard [31], one of the most widely used measures is the *graph edit distance* (GED). Given two labeled graphs G and H, it is defined as $\mathrm{GED}(G, H) \coloneqq \min_{P \in \Psi(G,H)} c(P)$, where Ψ is the set of all *edit paths* between G and H and $c(P)$ denotes the cost of an edit path P. An edit path is a sequence of *edit operations* that transforms G into H. There are six edit operations: substituting a node or an edge in G by a node or an edge in H, deleting an edge or an isolated node from G, and inserting an edge or an isolated node into H. Each edit operation comes with an associated non-negative *edit cost* defined in terms of the node or edge labels involved in the operation; and the cost of an edit path is defined as the sum over the costs of its edit operations.

Over the past years, some exact and a lot of approximate algorithms for computing GED have been suggested. As the hardness of GED does not allow for a theoretical evaluation of approximate algorithms (the existence of any α-approximation algorithm for GED would imply that the graph isomorphism problem, a prime candidate for an $\mathcal{NP}$-intermediate problem, is in $\mathcal{P}$), these algorithms are typically evaluated empirically. In order for such a comparison to be fair, it is highly desirable that the compared algorithms be implemented within the same environment. However, to the best of our knowledge, no software is available that can be used for this purpose.

In this paper, we present the C++ template library GEDLIB which is intended to fill this gap. GEDLIB is available on GitHub:

https://github.com/dbblumenthal/gedlib

In its current version, GEDLIB contains implementations of 24 different GED algorithms and 9 different edit cost functions. Further algorithms and edit costs can be implemented easily by implementing abstract classes contained in GEDLIB. For this, the user has access to standard libraries for blackbox optimization, mixed integer linear programming, the linear sum assignment problem with and without error-correction, deep neural networks, and support vector machines. GEDLIB provides a parser to load graphs given in the GXL file format. Alternatively, graphs with user-specified node ID, node, and edge label types can be constructed from within GEDLIB. Internally, GEDLIB uses the Boost Graph Library [22] for representing the graphs and Eigen [19] for matrix operations.

The remainder of this paper is organized as follows: In Sect. 2, the overall architecture of GEDLIB is sketched. In Sect. 3, the user interface is presented. In Sects. 4 and 5, the abstract classes for implementing GED algorithms and edit cost functions are described. Section 6 concludes the paper. Details, examples, and installation instructions can be found in the documentation.

2 Overall Architecture

Figure 1 shows the overall architecture of GEDLIB in a UML diagram. The entire library is contained in the namespace `ged`. The template parameters `UserNodeID`, `UserNodeLabel`, and `UserEdgeLabel` correspond to the types of the node IDs, the node labels, and the edge labels of the graphs provided by the user.

– The class template `ged::GEDEnv` provides the user interface. Via its public member functions, graphs can be constructed or loaded from GXL files, edit costs can be set, the algorithms implemented in GEDLIB can be run, and the results of the runs can be obtained. For users who do not want to provide extensions for GEDLIB, it suffices to get familiar with this class template.
– The abstract class template *`ged::GEDMethod`* provides a generic interface for implementing algorithms that exactly or approximately compute GED.
– The abstract class templates *`ged::LSBasedMethod`*, *`ged::MIPBasedMethod`*, and *`ged::LSAPEBasedMethod`* are derived from the generic interface provided by *`ged::GEDMethod`*. They yield more specialized interfaces for implementing

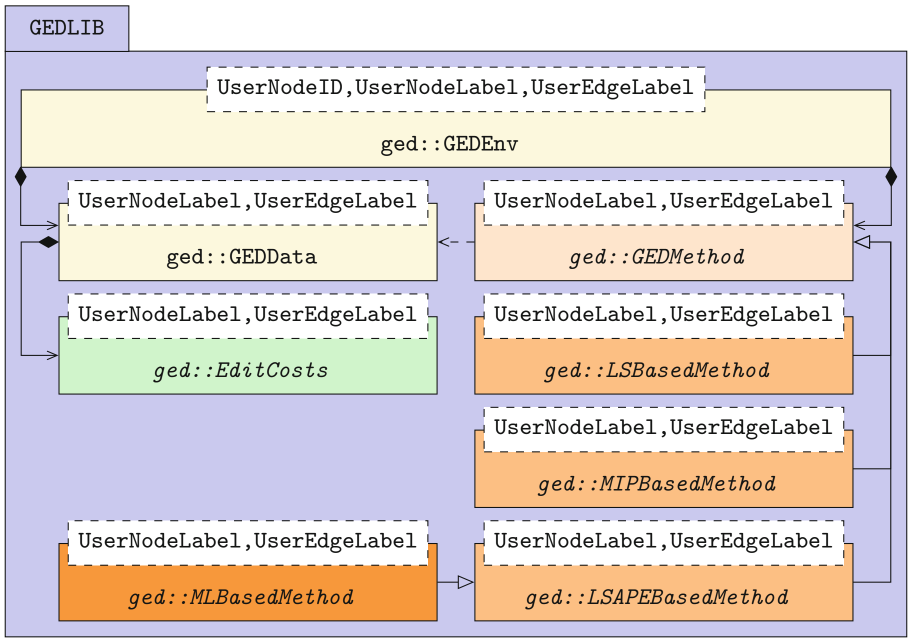

GEDLIB
UserNodeID,UserNodeLabel,UserEdgeLabel
ged::GEDEnv
UserNodeLabel,UserEdgeLabel
ged::GEDData
UserNodeLabel,UserEdgeLabel
ged::GEDMethod
UserNodeLabel,UserEdgeLabel
ged::EditCosts
UserNodeLabel,UserEdgeLabel
ged::LSBasedMethod
UserNodeLabel,UserEdgeLabel
ged::MIPBasedMethod
UserNodeLabel,UserEdgeLabel
ged::MLBasedMethod
UserNodeLabel,UserEdgeLabel
ged::LSAPEBasedMethod

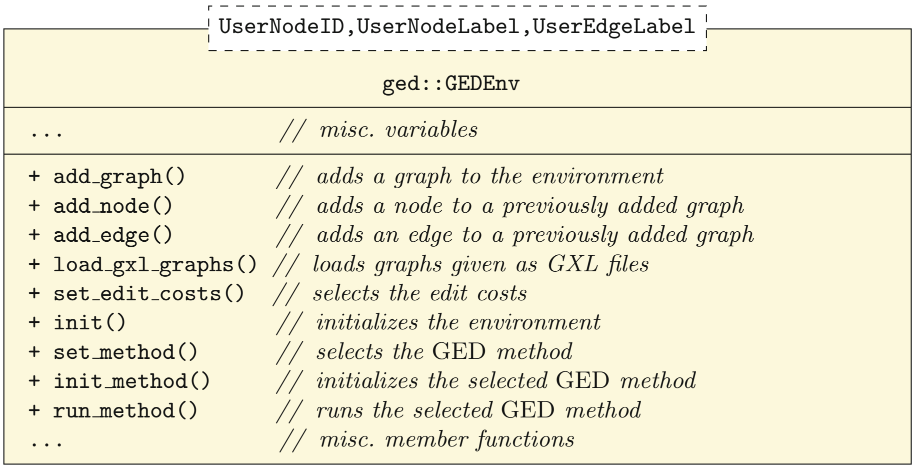
Fig. 1. The overall architecture of GEDLIB shown in a UML class diagram.

methods using local search, mixed integer linear programming, and transformations to the linear sum assignment problem with error-correction.
- The abstract class template *`ged::MLBasedMethod`* is derived from the interface *`ged::LSAPEBasedMethod`*. It can be used to implement algorithms that use deep neural networks or support vector machines for transforming GED to the linear sum assignment problem with error-correction.
- The class template `ged::GEDData` contains the normalized input data on which all GED algorithms contained in GEDLIB operate. Via the public member functions of `ged::GEDData`, derived classes of *`ged::GEDMethod`* have access to the graphs that have been added to the environment and to the edit cost functions selected by the user.
- The abstract class template *`ged::EditCosts`* provides a generic interface for implementing edit cost functions.

3 User Interface

In Fig. 2, the class template `ged::GEDEnv`, which constitutes the user interface of GEDLIB, is displayed in detail. By calling `add_graph()`, `add_node()`, and `add_edge()`, the user can add labeled graphs to the environment. Alternatively, `load_gxl_graphs()` can be used to load graphs given in the GXL file format. For this, the template parameter `UserNodeID` must be set to `ged::GXLNodeID` a. k. a. `std::string`, and the template parameters `UserNodeLabel` and `UserEdgeLabel` must be set to `ged::GXLLabel` a. k. a. `std::map<std::string,std::string>`.

Calls to `set_edit_costs()` add edit cost functions to the environment. The user can either select one of the predefined edit cost functions or use her own implementation of *`ged::EditCosts`*. Calls to `init()` initialize the environment eagerly or lazily. If eager initialization is chosen, all edit costs between graphs contained in the environment are precomputed. Otherwise, the edit cost functions are evaluated on the fly. The member function `set_method()` selects one of the GED algorithms available in GEDLIB. Some algorithms accept options, which can be passed to `set_method()` as a string of the form `"[--<option> <arg>] [...]"`. Calls to `init_method()` initialize the selected method for runs between graphs contained in the environment, and calls to `run_method()` run the method between two specified graphs. The results of the runs (lower and upper bounds, runtimes, etc.) can be accessed via various getter member function.



```
UserNodeID,UserNodeLabel,UserEdgeLabel

ged::GEDEnv

...                   // misc. variables

+ add_graph()         // adds a graph to the environment
+ add_node()          // adds a node to a previously added graph
+ add_edge()          // adds an edge to a previously added graph
+ load_gxl_graphs()   // loads graphs given as GXL files
+ set_edit_costs()    // selects the edit costs
+ init()              // initializes the environment
+ set_method()        // selects the GED method
+ init_method()       // initializes the selected GED method
+ run_method()        // runs the selected GED method
...                   // misc. member functions
```

Fig. 2. The user interface `ged::GEDEnv`.

4 Abstract Classes for Implementing GED Algorithms

Generic Interface. Figure 3 details the abstract class template *`ged::GEDMethod`*, which provides the generic interface for implementing GED. The interface is defined by the virtual member functions starting with the prefix *`ged_`*. We here describe only the most important virtual member functions; the remaining ones are detailed in the documentation: *`ged_run_()`* runs the method between two input graphs, *`ged_init_()`* initializes the methods for the graphs that have been

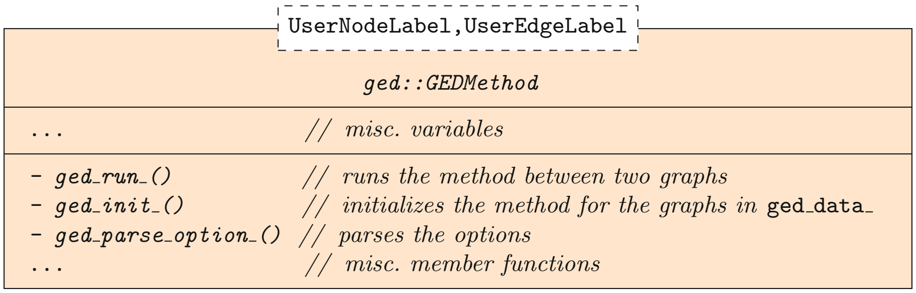

UserNodeLabel,UserEdgeLabel
ged::GEDMethod
... // misc. variables
- ged_run_() // runs the method between two graphs
- ged_init_() // initializes the method for the graphs in ged_data_
- ged_parse_option_() // parses the options
... // misc. member functions

Fig. 3. The generic interface *ged::GEDMethod*.

added to the environment, and *ged_parse_option_()* parses the options of the method. The following existing algorithms already implemented in GEDLIB are directly derived classes of *ged::GEDMethod*: `ged::BranchTight` [2], `ged::HED` [17], `ged::Partition` [32], `ged::Hybrid` [32], `ged::SimulatedAnnealing` [30], `ged::BranchCompact` [32], `ged::AnchorAwareGED` [14].

Interface for Methods Based on the Linear Sum Assignment Problem with Error-Correction. A popular approach for approximating GED is to use transformations to the *linear sum assignment problem with error-correction* (LSAPE). An instance of LSAPE consists of a cost matrix $\mathbf{C} = (c_{i,k}) \in \mathbb{R}_{\geq 0}^{(n+1)\times(m+1)}$. The task is to compute a mapping π from rows to columns, such that each row except for $n+1$ and each column expect for $m+1$ is covered exactly once and $\mathbf{C}(\pi) := \sum_{(i,k)\in\pi} c_{i,k}$ is minimized. LSAPE can be solved optimally in cubic time [10]; in GEDLIB, we use the LSAPE toolbox [8] for solving LSAPE.

If LSAPE is used for approximating GED(G, H), n and m are set to $|V^G|$ and $|V^H|$, the first $|V^G|$ rows of $\mathbf{C}$ are associated with the nodes of G, the first $|V^H|$ columns of $\mathbf{C}$ are associated with the nodes of H, and the last rows and columns are associated with dummy nodes used for codifying node insertions and deletions. With this setup, each LSAPE solution π corresponds to a *node map between* G and H, which, in turn, induces an edit path and hence an upper bound for GED(G, H) [6]. LSAPE based heuristics for GED try to achieve tight upper bounds by encoding structural information of the input graphs into $\mathbf{C}$. Moreover, some of them construct $\mathbf{C}$ such that $\min_\pi \mathbf{C}(\pi)$ lower bounds GED.

Figure 4 shows the abstract class template *ged::LSAPEBasedMethod*, which provides the interface for implementing heuristics of this kind. The interface is defined by the virtual member functions starting with the prefix *lsape_*. The most important one is *lsape_populate_instance_()*, which populates the LSAPE instance $\mathbf{C}$. The following algorithms implemented in GEDLIB are directly derived classes of *ged::LSAPEBasedMethod*: `ged::Bipartite` [26], `ged::Branch` [2], `ged::BranchFast` [2], `ged::Node` [21], `ged::BranchUniform` [32], `ged::Ring` [3], `ged::Subgraph` [12], `ged::Walks` [18]. Additionally, all derived classes of *ged::LSAPEBasedMethod* can be run with the node centralities suggested in [27].

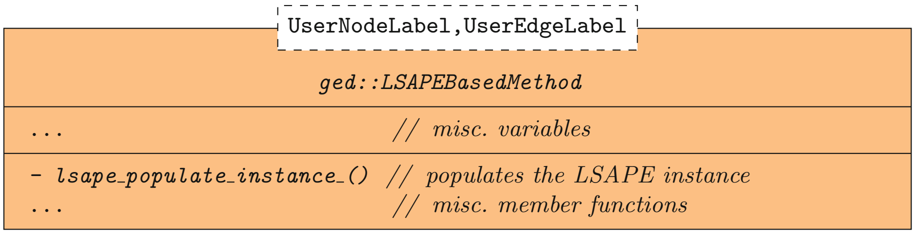

UserNodeLabel,UserEdgeLabel
ged::LSAPEBasedMethod
... // misc. variables
- lsape_populate_instance_() // populates the LSAPE instance
... // misc. member functions

Fig. 4. The interface *ged::LSAPEBasedMethod* for methods based on LSAPE.

Interface for Methods Based on Machine Learning. Recently, it has been suggested to use deep neural networks or support vector machines for carrying out the transformation from GED to LSAPE. Given two graphs G and H, feature vectors are constructed for all node substitutions, deletions, and insertions, and the matrix $\mathbf{C}$ is defined as $c_{i,k} \coloneqq 1 - p^{\star}(i,k)$. Here, $p^{\star}(i,k)$ is the confidence of a machine learning framework (either a deep neural network or a support vector machine) that the feature vector associated to the node edit operation corresponding to row i and column k is contained in an optimal node map.

Figure 5 details the abstract class template *ged::MLBasedMethod*, which provides the interface for algorithm adopting this paradigm. For implementing the interface, it suffices to override the virtual member functions starting with the prefix *ml_*. The most important ones are the three virtual member functions of the form *ml_populate_*_feature_vector_()*, which construct the feature vectors associated to the node edit operations. Derived classes of *ged::MLBasedMethod* do not have to implement the machine learning frameworks, as *ged::MLBasedMethod* offers support for artificial deep neural networks (using FANN [24]) and support vector machines (using LIBSVM [13]). The following algorithms implemented in GEDLIB are directly derived classes of *ged::MLBasedMethod*: `ged::BipartiteML` [28], `ged::RingML` [4].

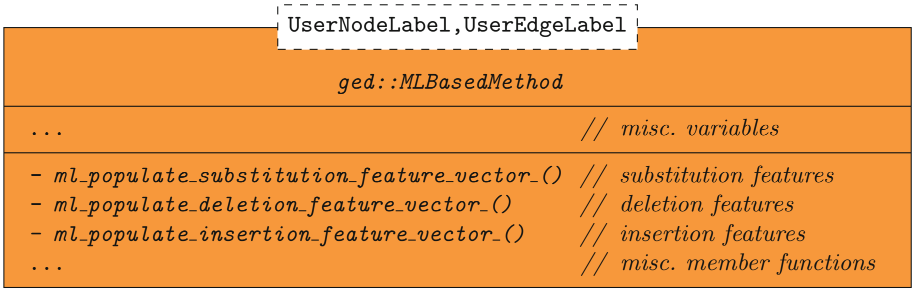

UserNodeLabel,UserEdgeLabel
ged::MLBasedMethod
... // misc. variables
- ml_populate_substitution_feature_vector_() // substitution features
- ml_populate_deletion_feature_vector_() // deletion features
- ml_populate_insertion_feature_vector_() // insertion features
... // misc. member functions

Fig. 5. The interface *ged::MLBasedMethod* for LSAPE based methods that use machine learning techniques for populating their LSAPE instances.

Interface for Methods Based on Mixed Integer Programming. Another approach for exactly or approximately computing GED is to rephrase the problem of computing $\mathrm{GED}(G, H)$ as a mixed integer programming (MIP) problem. $\mathrm{GED}(G, H)$ can then be computed exactly by calling an MIP solver. Alternatively, lower bounds for $\mathrm{GED}(G, H)$ can be obtained by solving the linear programming (LP) relaxations of the MIP formulations.

Figure 6 shows the abstract class template *ged::MIPBasedMethod*, which provides the interface for GED algorithms that use MIP formulations. The virtual member functions that define the interface start with the prefix *mip_*. The most important one is *mip_populate_model_()*, which constructs the employed MIP formulation and must be overridden by all derived classes. In GEDLIB, we use Gurobi [20] as our MIP and LP solver. Gurobi is commercial software but offers a free academic license. For users who cannot obtain a license for Gurobi, the installation script distributed with GEDLIB offers the option to install GEDLIB without *ged::MIPBasedMethod* and its derived classes. The following algorithms implemented in GEDLIB are directly derived classes of *ged::MIPBasedMethod*: `ged::F1` [23], `ged::F2` [23], `ged::CompactMIP` [6], `ged::BLPNoEdgeLabels` [21].

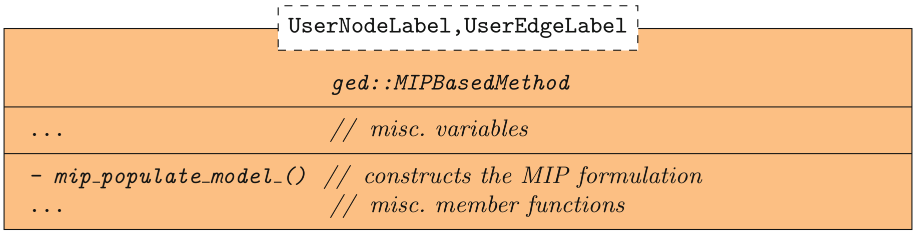
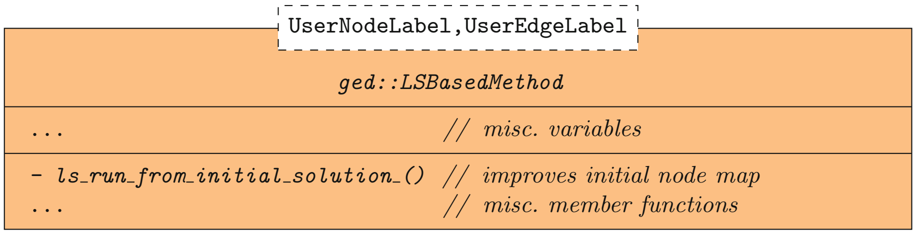

UserNodeLabel,UserEdgeLabel
ged::MIPBasedMethod
...
// misc. variables
- mip_populate_model_()
// constructs the MIP formulation
...
// misc. member functions

Fig. 6. The interface *ged::MIPBasedMethod* for methods based on MIP.

Interface for Methods Based on Local Search. Another popular approach for upper bounding GED is to use variants of local search to systematically vary a previously computed or randomly generated node map, such that the cost of the induced edit path decreases. Figure 7 shows the abstract class template *ged::LSBasedMethod*, which provides the interface for algorithms using local search. The prefix *ls_* marks the virtual member functions defining the interface. The most important one is *ls_run_from_initial_solution_()*, which runs the local search from an initial node map. The following algorithms implemented in GEDLIB are directly derived classes of *ged::LSBasedMethod*: `ged::IPFP` [5, 9, 11], `ged::BPBeam` [16, 29], `ged::Refine` [31]. Moreover, *ged::LSBasedMethod* provides support for running all derived classes with parallel multi-start as suggested in [15], and stochastic generators as suggested in [7].

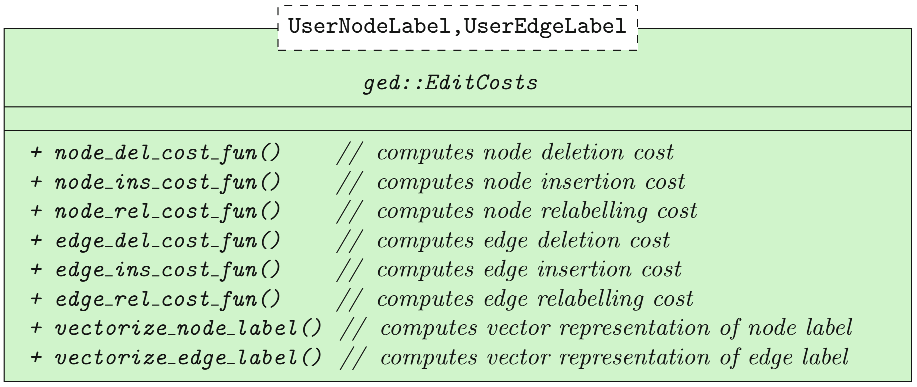


```
UserNodeLabel,UserEdgeLabel
ged::LSBasedMethod
...                                  // misc. variables
- ls_run_from_initial_solution_()    // improves initial node map
...                                  // misc. member functions
```

Fig. 7. The interface *ged::LSBasedMethod* for methods based on local search.

5 Abstract Class for Implementing Edit Costs

Figure 8 shows the abstract class template *ged::EditCosts*, which provided the interface for implementing edit cost functions. The virtual member functions **_del_cost_fun()* compute the cost of deleting a node or an edge with a given label, the functions **_ins_cost_fun()* compute the insertions costs, and the functions **_rel_cost_fun()* compute the costs for relabeling a node or an edge. The functions *vectorize_*_label()* return vector representations of the node and the edge labels, which are required by some methods. In GEDLIB, edit costs are available for the datasets AIDS, FINGERPRINT, GREC, LETTER, MUTAGENICITY, and PROTEIN from the IAM Graph Database [25], for the datasets ACYCLIC, ALKANE, PAH, and MAO from GREYC's Chemistry Dataset (available at https://brunl01.users.greyc.fr/CHEMISTRY/), and for the dataset CMU-GED from the Graph Data Repository for Graph Edit Distance [1]. We also provide constant edit cost functions that can be used with any data.



```
UserNodeLabel,UserEdgeLabel
ged::EditCosts

+ node_del_cost_fun()       // computes node deletion cost
+ node_ins_cost_fun()       // computes node insertion cost
+ node_rel_cost_fun()       // computes node relabelling cost
+ edge_del_cost_fun()       // computes edge deletion cost
+ edge_ins_cost_fun()       // computes edge insertion cost
+ edge_rel_cost_fun()       // computes edge relabelling cost
+ vectorize_node_label()    // computes vector representation of node label
+ vectorize_edge_label()    // computes vector representation of edge label
```

Fig. 8. The interface *ged::EditCosts* for implementing edit costs.

6 Conclusions and Future Work

In this paper, we have presented GEDLIB, a C++ library for GED computations. GEDLIB currently implements 24 different GED algorithms and 9 different edit cost functions designed for datasets which are widely used in the research community. In the future, we will provide Python and MATLAB bindings for better usability. Moreover, we would like to encourage authors of algorithms and edit costs that are not implemented in GEDLIB to commit their work to GEDLIB.

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. http://www.rfai.li.univ-tours.fr/PublicData/GDR4GED/home.html
2. Blumenthal, D.B., Gamper, J.: Improved lower bounds for graph edit distance. IEEE Trans. Knowl. Data Eng. **30**(3), 503–516 (2018). https://doi.org/10.1109/TKDE.2017.2772243
3. Blumenthal, D.B., Bougleux, S., Gamper, J., Brun, L.: Ring based approximation of graph edit distance. In: Bai, X., Hancock, E.R., Ho, T.K., Wilson, R.C., Biggio, B., Robles-Kelly, A. (eds.) S+SSPR 2018. LNCS, vol. 11004, pp. 293–303. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-97785-0_28
4. Blumenthal, D.B., Bougleux, S., Gamper, J., Brun, L.: Upper bounding GED via transformations to LSAPE based on rings and machine learning (2018, submitted)
5. Blumenthal, D.B., Daller, E., Bougleux, S., Brun, L., Gamper, J.: Quasimetric graph edit distance as a compact quadratic assignment problem. In: ICPR 2018, pp. 934–939 (2018)
6. Blumenthal, D.B., Gamper, J.: On the exact computation of the graph edit distance. Pattern Recognit. Lett. (2018). https://doi.org/10.1016/j.patrec.2018.05.002
7. Boria, N., Bougleux, S., Brun, L.: Approximating GED using a stochastic generator and multistart IPFP. In: Bai, X., Hancock, E.R., Ho, T.K., Wilson, R.C., Biggio, B., Robles-Kelly, A. (eds.) S+SSPR 2018. LNCS, vol. 11004, pp. 460–469. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-97785-0_44
8. Bougleux, S., Brun, L.: Linear sum assignment with edition. arXiv:1603.04380 [cs.DS] (2016). https://bougleux.users.greyc.fr/lsape/
9. Bougleux, S., Brun, L., Carletti, V., Foggia, P., Gaüzère, B., Vento, M.: Graph edit distance as a quadratic assignment problem. Pattern Recognit. Lett. **87**, 38–46 (2017). https://doi.org/10.1016/j.patrec.2016.10.001
10. Bougleux, S., Gaüzère, B., Blumenthal, D.B., Brun, L.: Fast linear sum assignment with error-correction and no cost constraints. Pattern Recognit. Lett. (2018). https://doi.org/10.1016/j.patrec.2018.03.032
11. Bougleux, S., Gaüzère, B., Brun, L.: Graph edit distance as a quadratic program. In: ICPR 2016, pp. 1701–1706 (2016). https://doi.org/10.1109/ICPR.2016.7899881
12. 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

13. Chang, C.C., Lin, C.J.: LIBSVM: a library for support vector machines. ACM Trans. Intell. Syst. Technol. **2**(3), 27 (2011). https://doi.org/10.1145/1961189.1961199. https://www.csie.ntu.edu.tw/~cjlin/libsvm/
14. Chang, L., Feng, X., Lin, X., Qin, L., Zhang, W.: Efficient graph edit distance computation and verification via anchor-aware lower bound estimation. arXiv:1709.06810 [cs.DB] (2017)
15. Daller, É., Bougleux, S., Gaüzère, B., Brun, L.: Approximate graph edit distance by several local searches in parallel. In: ICPRAM 2018, pp. 149–158 (2018). https://doi.org/10.5220/0006599901490158
16. 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
17. Fischer, A., Suen, C.Y., Frinken, V., Riesen, K., Bunke, H.: Approximation of graph edit distance based on Hausdorff matching. Pattern Recognit. **48**(2), 331–343 (2015). https://doi.org/10.1016/j.patcog.2014.07.015
18. Gaüzère, B., Bougleux, S., Riesen, K., Brun, L.: Approximate graph edit distance guided by bipartite matching of bags of walks. In: Fränti, P., Brown, G., Loog, M., Escolano, F., Pelillo, M. (eds.) S+SSPR 2014. LNCS, vol. 8621, pp. 73–82. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44415-3_8
19. Guennebaud, G., Jacob, B., et al.: Eigen v3 (2010). http://eigen.tuxfamily.org
20. Gurobi Optimization, LLC: Gurobi optimizer reference manual (2018). http://www.gurobi.com
21. 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). https://doi.org/10.1109/TPAMI.2006.152
22. Lee, L., Lumsdaine, A., Siek, J.: The Boost Graph Library: User Guide and Reference Manual (2002). https://www.boost.org/doc/libs/1_68_0/libs/graph/doc/index.html
23. Lerouge, J., Abu-Aisheh, Z., Raveaux, R., Héroux, P., Adam, S.: New binary linear programming formulation to compute the graph edit distance. Pattern Recognit. **72**, 254–265 (2017). https://doi.org/10.1016/j.patcog.2017.07.029
24. Nissen, S.: Implementation of a fast artificial neural network library (FANN). Technical report, Department of Computer Science, University of Copenhagen (DIKU) (2003). http://leenissen.dk/fann/wp/
25. Riesen, K., Bunke, H.: IAM graph database repository for graph based pattern recognition and machine learning. S+SSPR 2008. LNCS, vol. 5342, pp. 287–297. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89689-0_33. http://www.fki.inf.unibe.ch/databases/iam-graph-database
26. Riesen, K., Bunke, H.: Approximate graph edit distance computation by means of bipartite graph matching. Image Vis. Comput. **27**(7), 950–959 (2009). https://doi.org/10.1016/j.imavis.2008.04.004
27. Riesen, K., Bunke, H., Fischer, A.: Improving graph edit distance approximation by centrality measures. ICPR 2014, pp. 3910–3914 (2014). https://doi.org/10.1109/ICPR.2014.671
28. Riesen, K., Ferrer, M.: Predicting the correctness of node assignments in bipartite graph matching. Pattern Recognit. Lett. **69**, 8–14 (2016). https://doi.org/10.1016/j.patrec.2015.10.007

29. Riesen, K., Fischer, A., Bunke, H.: Combining bipartite graph matching and beam search for graph edit distance approximation. In: El Gayar, N., Schwenker, F., Suen, C. (eds.) ANNPR 2014. LNCS (LNAI), vol. 8774, pp. 117–128. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11656-3_11
30. 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
31. Zeng, Z., Tung, A.K.H., Wang, J., Feng, J., Zhou, L.: Comparing stars: on approximating graph edit distance. PVLDB **2**(1), 25–36 (2009). https://doi.org/10.14778/1687627.1687631
32. Zheng, W., Zou, L., Lian, X., Wang, D., Zhao, D.: Efficient graph similarity search over large graph databases. IEEE Trans. Knowl. Data Eng. **27**(4), 964–978 (2015). https://doi.org/10.1109/TKDE.2014.2349924