Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is

# Scalable generalized median graph estimation and its manifold use in bioinformatics, clustering, classification, and indexing

David B. Blumenthal [a,*,1], Nicolas Boria [b,c,1], Sébastien Bougleux [b], Luc Brun [b], Johann Gamper [d,*], Benoit Gaüzère [e]

[a] *Technical University of Munich, TUM School of Life Sciences, Chair of Experimental Bioinformatics, Freising, Germany*
[b] *Normandie Université, ENSICAEN, UNICAEN, CNRS, GREYC, Caen, France*
[c] *Université Paris-Dauphine, CNRS, LAMSADE, Paris, France*
[d] *Free University of Bozen-Bolzano, Faculty of Computer Science, Bolzano, Italy*
[e] *Normandie Université, INSA Rouen Normandie, LITIS, Rouen, France*

A R T I C L E  I N F O

*Article history:*
Received 25 May 2020
Received in revised form 19 January 2021
Accepted 15 March 2021
Available online 27 March 2021
Recommended by Limsoon Wong

*MSC:*
68R01
05C85
90C59

*Keywords:*
Generalized median graphs
Graph edit distance
Graph similarity search
Clustering
Classification
Indexing

A B S T R A C T

In this paper, we present GMG-BCU — a local search algorithm based on block coordinate update for estimating a generalized median graph for a given collection of labeled or unlabeled input graphs. Unlike all competitors, GMG-BCU is designed for both discrete and continuous label spaces and can be configured to run in linear time w.r.t. the size of the graph collection whenever median node and edge labels are computable in linear time. These properties make GMG-BCU usable for applications such as differential microbiome data analysis, graph classification, clustering, and indexing. We also prove theoretical properties of generalized median graphs, namely, that they exist under reasonable assumptions which are met in almost all application scenarios, that they are in general non-unique, that they are *NP*-hard to compute and *APX*-hard to approximate, and that no polynomial $\alpha$-approximation exists for any $\alpha$ unless the graph isomorphism problem is in *P*. Extensive experiments on six different datasets show that our heuristic GMG-BCU always outperforms the state of the art in terms of runtime or quality (on most datasets, both w.r.t. runtime and quality), that it is the only available heuristic which can cope with collections containing several thousands of graphs, and that it shows very promising potential when used for the aforementioned applications. GMG-BCU is freely available on GitHub: https://github.com/dbblumenthal/gedlib/.

## 1. Introduction

Labeled graphs provide a powerful structure to represent, process, and analyze data in a wide variety of scientific domains such as cancer detection [1], keyword spotting in handwritten documents [2], and many more [3]. However, determining fundamental tools such as a distance or an average graph is non-trivial. Given a space $\mathbb{G}$ of labeled simple graphs, the graph edit distance (GED) is a natural choice for comparing two graphs [4,5]. It measures the minimal amount of distortion needed to transform one graph into another by means of edit operations. Given cost functions for the edit operations, GED can be defined as the minimum cost of a node map (i.e., a relation between the nodes of the source and the target graph) that specifies for each node whether it should be substituted, inserted, or deleted. Computing GED is *NP*-hard [6] and still cannot be solved in a reasonable time for graphs exceeding a dozen of vertices, even for simple cost functions [7]. Therefore, in practice, various heuristics are employed that provide tight lower or upper bounds in polynomial time [5,8–13].

In this paper, we consider the even more difficult problem to compute a generalized median graph for a given collection of graphs. Intuitively, a generalized median graph is a graph that, among all graphs contained in the (infinitely large) space of graphs, minimizes the overall GED to the graphs contained in the collection. Generalized median graphs can be viewed as representatives or prototypes of the graphs contained in the collection and can hence be employed in all application scenarios that build upon the computation of representatives. Formally, they are defined as follows [14]:

**Definition 1** (*Generalized Median Graph*)**.** A graph $G^\star \in \mathbb{G}$ is a *generalized median graph* for a finite collection of graphs $\mathcal{G} \subseteq \mathbb{G}$

* Corresponding authors.
*E-mail addresses:* david.blumenthal@wzw.tum.de (D.B. Blumenthal), nicolas.boria@dauphine.fr (N. Boria), sebastien.bougleux@unicaen.fr (S. Bougleux), luc.brun@ensicaen.fr (L. Brun), johann.gamper@unibz.it (J. Gamper), benoit.gauzere@insa-rouen.fr (B. Gaüzère).
1 D.B.B. and N.B. contributed equally to this work.

https://doi.org/10.1016/j.is.2021.101766

if and only if $G^\star \in \arg\min_{G \in \mathbb{G}} \mathrm{SOD}(G, \mathcal{G})$, where $\mathrm{SOD}(G, \mathcal{G}) := \sum_{G^p \in \mathcal{G}} \mathrm{GED}(G, G^p)$ is the *sum of distances* from a graph $G \in \mathbb{G}$ to all graphs $G^p \in \mathcal{G}$.

Computing generalized medians is a difficult task for most kinds of objects — even for strings, it is *NP*-hard [15,16]. For graphs, the situation is even worse, because here already evaluating the underlying distance measure is a hard problem. For this reason, exact algorithms have to explore a huge search space [17–19] and their application is very limited.

To estimate median graphs in a reasonable computational time, several methods estimate the optimal SOD via a local search around an initial candidate graph, using genetic search [14,18,20], greedy search based on partitioning vertices of different graphs [21], local search [22], or linear programming [23]. However, these methods are restricted to finite node and edge label alphabets [14], are designed for very small graphs [21], require special constant edit costs [22], or have a very high computational cost [23]. A second family of heuristics [24,25] exploits techniques from spectral graph theory [26] to construct an estimated generalized median graph as a combination of the modal matrices of the graphs contained in $\mathcal{G}$. The main drawbacks of these approaches are that they expect all input graphs to be of the same order, that they only support real-valued node and edge labels, and that they do not support node insertions and deletions. Yet another strategy is based on graph embedding into vector spaces [27,28] with the distances between graphs as coordinates [18,29–32]. Then a representative is computed in the embedding space, and an estimated median graph is constructed by going back to the original space of graphs. Embedding based approaches overcome the limitations of the ones mentioned before, but have to compute all pairwise distances at initialization and hence run in at least quadratic time w. r. t. the size of the graph collection. Moreover, just as for all other existing heuristics, no formal link between the employed algorithmic techniques and the definition of a generalized median graph is provided.

This paper introduces a new local search algorithm GMG-BCU that estimates a **g**eneralized **m**edian **g**raph via a **b**lock **c**oordinate **u**pdate (BCU). Given an initial candidate or a set of initial candidates, GMG-BCU iterates through the following three steps: (1) update the node maps, keeping the current median graph fixed; (2) update the median graph, keeping the current node maps and the current order of the median graph fixed; (3) update the order of the median graph. Each step decreases the SOD, which implies that the obtained estimated median graph is a local optimum. Moreover, GMG-BCU is generic as it makes only very mild assumptions on the labels and the edit costs which, to the best of our knowledge, are met in all applications where GED is used to solve real-world problems [3]. Unlike existing approaches, GMG-BCU can be configured to run in linear time w. r. t. the size of the graph collection $\mathcal{G}$ whenever median node and edge labels can be computed in linear time. For instance, this is the case if the labels are symbolic, if they are real-valued vectors, or if the label spaces are of constant size. In most application scenarios, our algorithm hence scales to large graph collections.

We also prove theoretical properties of the problem of computing generalized median graphs that are implicitly taken for granted in previous works but are nonetheless non-trivial. More precisely, we show that generalized median graphs exist under the mild assumptions that median node and edge labels exist and that the insertion and deletion costs are constant (cf. Theorem 1). We also prove that generalized median graphs are in general not unique, that they are *NP*-hard to compute and *APX*-hard to approximate, and that no polynomial time $\alpha$-approximation algorithm exists for any $\alpha$ unless the graph isomorphism is in *P*. As an outlook, we discuss potential applications. Here, we show to use GMG-BCU for computing representative graphs for microbiome data, which can then be used for downstream differential data analysis. We also point out to the fact that, once scalable estimators are available, generalized median graphs can be used beneficially for tackling higher-level problems such as graph clustering, classification, and indexing. Finally, we report the results of an extensive empirical evaluation. The evaluation shows that our algorithm clearly outperforms existing competitors, and that it yields extremely promising results when used within the aforementioned applications scenarios. In sum, the paper contains the following contributions:

- We show that generalized median graphs exist under mild assumptions, that they are in general non-unique, and that they are hard to compute and approximate.
- We present GMG-BCU, a scalable and efficient local search algorithm based on BCU that produces locally optimal estimated generalized median graphs and can be configured to run in linear time w. r. t. the size of the graph collection whenever median node and edge labels are computable in linear time.
- We point out to the fact that, thanks to our new efficient algorithm, generalized median graphs can be beneficially applied in an application from bioinformatics as well as for higher-level problems such as graph clustering, classification, and indexing.
- We report the results of extensive experiments, showing that GMG-BCU clearly outperforms the state of the art.
- A well-documented, ready-to-use C++ implementation of GMG-BCU is freely available at https://github.com/dbblumenthal/gedlib/.

The remainder of this paper is organized as follows: In Section 2, we discuss related work. In Section 3, we introduce important concepts and notations. In Section 4, we present our theoretical results. In Section 5, we introduce the algorithm GMG-BCU. In Section 6, we discuss potential applications. In Section 7, we report the results of the experiments. Section 8 concludes the paper. The proofs of all presented theorems and propositions are contained in the appendix. The paper builds upon the results published in [33], where a preliminary version of GMG-BCU without order update, theoretical contributions, and applications was presented.

## 2. Related work

In this section, we provide an extensive overview of existing methods for heuristically estimating generalized median graphs (Sections 2.1 to 2.3). We also discuss some state of the art methods for closely related problems (Section 2.4). While many heuristics for estimating generalized median graphs have been suggested during the past years, to the best of our knowledge, all of them exhibit one or several of the following major shortcomings:

- There is no clear theoretical link between the algorithmic techniques employed by existing heuristics and the definition of a generalized median graph.
- Existing heuristics only support very restricted types of graphs and edit costs.
- Existing heuristics do not scale to large collections of graphs.

### 2.1. Spectral methods

Spectral methods [24,25] only apply to graphs of the same order with real-valued node and edges values. They are also restricted to the weighted graph matching problem, which can be viewed as a particular form of GED where node deletions and insertions are forbidden and the costs of all other edit operations are defined as the squared Euclidean distances between the labels. If these restrictions are met, graphs can be represented by modal matrices that are constructed as the concatenation of the adjacency matrices' eigenvectors, which are sorted in non-increasing order w. r. t. the eigenvalues. If two graphs are isomorphic or nearly isomorphic and there are no multiple eigenvalues, the graph matching problem is then equivalent to an instance of the linear sum assignment problem that aligns the two sets of nodes such that the absolute correlation between the graphs' modal matrices is maximized [26].

This equivalence is used in [24] to formulate spectral versions of the medoid graph problem and the generalized median graph problem. While spectral medoids can be computed in polynomial time, the spectral median problem is indeed a median problem for multisets of real-valued vectors and hence no polynomial algorithm is available [34]. Also, when graphs are not nearly isomorphic, the spectral alignment of their nodes induces a suboptimal alignment of the graphs, and thus the link to the medoid or to the generalized median graph is broken. Within this context, an iterative algorithm for estimating generalized median graphs is presented in [24]. Given two initial graphs from the collection, the algorithm computes the average aligned eigenvalues and modal matrices and constructs an intermediate median graph from the average eigendecomposition. This process is repeated with the intermediate median and an unselected graph until all graphs have been used. However, the resulting graph depends on the order in which the graphs are selected, and the links to both the spectral median problem and the generalized median graph problem are unclear.

An improved version of this approach is presented in [25], where median graphs are defined from a weighted average of the aligned eigenvalues and modal matrices. While this enables to reconstruct several candidate medians by mixing different scales of the structure of the graphs, there is still no formal link to the generalized median graph problem. Moreover, also this method does not overcome the strong restrictions shared by all spectral methods mentioned above.

### 2.2. Methods based on graph embedding into vector spaces

Another family of algorithms for estimating generalized median graphs builds upon an embedding of the input graphs into vector spaces [18,29–32]. All heuristics that fall into this category rely on the same embedding technique described in [28,35]: Each graph $G$ of the collection $\mathcal{G}$ is encoded as a vector $x_G \in \mathbb{R}^{|\mathcal{G}|}$, defined as $x_G := (\mathrm{GED}(G, G'))_{G' \in \mathcal{G}}$. The embedding preserves distance ratios, and the encoding vectors are linearly independent. The geometric median of the embeddings is then considered as a representative for the collection in the embedding space, and a median graph estimate in the graph space is constructed by heuristically solving the pre-image problem to find a graph whose embedding is close to the representative of the embeddings. Let $G^\star$ be a generalized median graph or weighted mean of two graphs $G$ and $G'$. To tackle the pre-image problem, embedding based heuristics assume that the embedding $x_{G^\star}$ of $G^\star$ lies on the segment that joins the embeddings $x_G$ and $x_{G'}$ of, respectively, $G$ and $G'$ [36]. Since weighted means can be constructed from optimal edit paths, this allows to interpolate $G^\star$ in the graph space. Note that, in practice, only suboptimal edit paths can be computed.

How exactly this pre-image problem is addressed is where different embedding based methods vary from each other. Two fast heuristics are proposed in [18,29]. The first one (LINEAR), constructs $G^\star$ from the projection of the geometric median on the segment joining its two nearest graphs w. r. t. the Euclidean distance between their embeddings. The second heuristic (TRIANGULAR) starts by selecting the three closest embeddings to the geometric median, and constructs an intermediate median graph by projecting one of these three embeddings on the segment joining the other two. $G^\star$ is then constructed by projecting the median of the three embeddings on the segment joining the embeddings of intermediate median and the first graph. Inspired by these heuristics, in [30], it is suggested to recursively construct $G^\star$ by projections onto hyperplanes of decreasing dimensionality until dimension 3 or 2 is reached. A version that returns the best intermediate median is also reported. While it improves the previous heuristics in terms of quality, it is more time consuming due to the computation of the SOD at each step.

Extensions of this approach are presented in [32]. The first extension groups the graphs into pairs (BEST-LINEAR) or triplets (BEST-TRIANGULAR) in non-decreasing order w. r. t. the Euclidean distance to the geometric median in the embedding space. Then, intermediate medians are constructed by applying LINEAR or TRIANGULAR to each pair or each triplet, and the process is repeated using these latter graphs only until one graph remains. Finally, the intermediate median with the smallest SOD is returned. The second extension searches for a better intermediate by considering the weighted means between the current intermediate median and its nearest neighbors in the embedding space. This process is then repeated with the new intermediate median until the improvement in the SOD is sufficiently small. Due to the huge number of SOD computations, this second extension is far too time consuming for large graphs or collections.

Constructing a median graph with embedding based methods has three severe drawbacks. Firstly, the link between the geometric median and the generalized median graph is unclear. Secondly, GED has to be solved many times — for all pairs of graphs in the collection for constructing the embeddings, and eventually several times for computing the SODs. Thirdly, the labels of the obtained estimated median are always drawn from the labels present in the collection, which leads to a loose SOD if the node or edge label space is continuous. Despite these drawbacks, these methods are more flexible than those described in the previous sections and are generic w. r. t. the spaces of labels, the edit operations, and the edit costs. Also, the experimental evaluation reported in [18] shows that, in comparison to spectral methods and genetic search, embedding based methods exhibit a good tradeoff between runtime and quality. For these reasons, they are actually the most competitive methods. A comparison with our algorithm is reported in Section 7.3.

### 2.3. Methods based on genetic and linear programming and greedy and local search

To estimate median graphs in a reasonable computational time, several methods compute suboptimal generalized median graphs using heuristics. In [14], a genetic algorithm is proposed. This method is restricted to real-valued node and edge labels and assumes that the squared Euclidean distance is used for the edit costs. In [18,20], this approach has been improved via new bounding techniques that reduce the size of the search space.

In [21], a greedy heuristic is proposed that splits the objective function into node and edge costs and optimizes both of them separately. Its main drawback is that it yields highly suboptimal median graph estimates, because it ignores the dependencies between node and edge edit operations. Also, its effectiveness has empirically been demonstrated only on very small graphs with up to 9 nodes.

In [23], a linear programming based algorithm is suggested. In a first step, the problem of computing a generalized median graph is formulated as an integer linear program. Subsequently, the continuous relaxation is solved via standard linear programming solvers, and the continuous solution is projected back to the discrete space. A disadvantage of this method is that the loss of the final projection step is hard to control. Moreover, the linear programming model is computationally very expensive because it has a huge number of variables and constraints, and the practical efficiency of the method critically depends on the employed solver.

In [22], two heuristics based on local search are proposed. The first one starts the search with the minimum common supergraph of the graph collection whose median should be computed. Then, the current candidate graph is iteratively improved in the direction of the node deletion that yields the largest improvement in terms of SOD. The second heuristic explores more search directions by randomly deleting nodes. Both heuristics can be used only in very restricted scenarios, as they allow neither node nor edge substitutions.

### 2.4. Other approaches for closely related problems

There are also several works that address problems which are slightly different from but closely related to the problem of heuristically computing generalized median graphs.

In [37], an algorithm for heuristically computing fuzzy generalized median graphs is proposed. Like the approaches presented in the previous section, the algorithm is based upon graph embedding into vector spaces, but uses a different embedding technique that is designed for the special case of graphs with fuzzy node and edge labels such as "small" or "big". Moreover, the algorithm does not employ GED for computing the embeddings or the median, which implies that there is no direct link to generalized median graphs as introduced in Definition 1.

Another closely related problem is the common labeling problem [38–40]. This problem asks to find a multiple assignment between the nodes of the graphs in the collection and the elements of a virtual set of nodes such that the sum of costs between all pairs of graphs aligned by the assignment is minimized. An approximate median graph can then be constructed from the multiple assignment. If the edit costs are metric, the optimal SOD can be upper bounded in terms of the multiple assignment [41]. Moreover, if graphs have real-valued labels and the squared Euclidean distance is used for the edit costs, the approximate median corresponds to the generalized median. Since the common labeling problem is *NP*-hard, several heuristics have been proposed in [39,40].

Finally, several recent works study the closely related problem to heuristically compute a graph that minimizes the Fréchet mean for graphs with real-valued labels, where the Fréchet mean is either defined based on the squared GED with specific edit costs [42], or in terms of the Gromov–Wasserstein distance for complete graphs [43,44]. Like the algorithm presented in this paper, these works employ block coordinate update from an initial candidate. While the obtained results are promising, the main drawback of these approaches is that the order of the computed representative is fixed by the order of the initial candidate.

## 3. Preliminaries

In this section, we introduce important concepts and notations. Table 1 provides a concise overview of the most frequent notations. The precise definitions are contained in the following subsections.

**Table 1**
Notation table.

| Syntax | Semantics |
|---|---|
| $\mathcal{L}_V$ | Set of node labels |
| $\mathcal{L}_E$ | Set of edge labels |
| $A \in \mathcal{A}_n$ | Adjacency matrix for graph of order $n$ |
| $\varphi \in \mathcal{L}_V^n$ | Vector of node labels for graph of order $n$ |
| $\Phi \in \mathcal{L}_E^{n\times n}$ | Matrix of edge labels for graph of order $n$ |
| $\mathbb{G}$ | All simple graphs with labels from $\mathcal{L}_V$ and $\mathcal{L}_E$ |
| $\mathcal{G} \subset \mathbb{G}$ | Finite collection of graphs |
| $\pi \in \Pi_{n,n'}$ | Node map from graph of order $n$ to graph of order $n'$ |
| $\boldsymbol{\pi} \in \Pi_n^{\mathcal{G}}$ | Node maps from graph of order $n$ to all graphs in $\mathcal{G}$ |

**Table 2**
Edit operations and edit cost functions.

| Edit operation type | Edit cost function |
|---|---|
| Substituting nodes | $c_{\text{ns}} : \mathcal{L}_V \times \mathcal{L}_V \to \mathbb{R}_{\geq 0}$ |
| Deleting nodes | $c_{\text{nd}} : \mathcal{L}_V \to \mathbb{R}_{\geq 0}$ |
| Inserting nodes | $c_{\text{ni}} : \mathcal{L}_V \to \mathbb{R}_{\geq 0}$ |
| Substituting edges | $c_{\text{es}} : \mathcal{L}_E \times \mathcal{L}_E \to \mathbb{R}_{\geq 0}$ |
| Deleting edges | $c_{\text{ed}} : \mathcal{L}_E \to \mathbb{R}_{\geq 0}$ |
| Inserting edges | $c_{\text{ei}} : \mathcal{L}_E \to \mathbb{R}_{\geq 0}$ |

### 3.1. Labeled graphs

In this paper, we consider simple graphs (directed or undirected) which may or may not have labels on the nodes and on the edges. Let $\mathcal{L}_V$ and $\mathcal{L}_E$ be sets of node and edge labels, respectively. There is no restriction on the label types; the labels might be symbols, numbers, multi-dimensional vectors, strings, or anything else. Then, a graph $G$ of order $n$ is defined as a triplet $G := (A, \varphi, \Phi)$:

- $A := (A_{i,j})_{i,j=1}^n \in \mathcal{A}_n$ is a graph's adjacency matrix, where $\mathcal{A}_n := \{A \in \{0,1\}^{n\times n} \mid \operatorname{diag}(A) = \mathbf{0}\}$ is the set of all adjacency matrices for simple graphs of order $n$.
- Nodes correspond to integers $i \in [n] := \{1, \ldots, n\}$.
- Edges correspond to pairs $(i, j) \in [n] \times [n]$ with $A_{i,j} = 1$.
- $\varphi := (\varphi_i)_{i=1}^n \in \mathcal{L}_V^n$ is the vector of node labels.
- $\Phi := (\Phi_{i,j})_{i,j=1}^n \in \mathcal{L}_E^{n\times n}$ is the matrix of edge labels.

For all $n \in \mathbb{N}$, $\mathbb{G}_n[\mathcal{L}_V, \mathcal{L}_E] := \{(A, \varphi, \Phi) \in \mathcal{A}_n \times \mathcal{L}_V^n \times \mathcal{L}_E^{n\times n}\}$ denotes the set all the graphs of order $n$ generated from $\mathcal{L}_V$ and $\mathcal{L}_E$. The set of all graphs for $\mathcal{L}_V$ and $\mathcal{L}_E$ is defined as $\mathbb{G}[\mathcal{L}_V, \mathcal{L}_E] := \bigcup_{n\in\mathbb{N}} \mathbb{G}_n[\mathcal{L}_V, \mathcal{L}_E]$. Since $\mathcal{L}_V$ and $\mathcal{L}_E$ are arbitrary but fixed, we usually use the short-hand notations $\mathbb{G}_n := \mathbb{G}_n[\mathcal{L}_V, \mathcal{L}_E]$ and $\mathbb{G} := \mathbb{G}[\mathcal{L}_V, \mathcal{L}_E]$. Note that $A$ and $\Phi$ are symmetric for undirected graphs. Therefore, whenever undirected graphs are being studied, $\mathcal{A}_n$ and $\mathcal{L}_E^{n\times n}$ denote the sets of symmetric adjacency and edge label matrices, respectively. Also, $\varphi$ and $\Phi$ are by definition independent from each others.

### 3.2. Graph edit distance and node maps

The graph edit distance (GED) is classically defined as the minimum cost of an edit path $P$ from a source graph $G$ to a target graph $G'$ [4,45]. An edit path from a graph $G$ of order $n$ to a graph $G'$ of order $n'$ is a sequence of edit operations that transforms $G$ into a graph isomorphic to $G'$. There are six edit operations: Substituting a node or and edge from $G$ by a node or an edge from $G'$, deleting an isolated node or an edge from $G$, and inserting an isolated node or an edge between two existing nodes into $G'$. As specified in Table 2, each type of edit operations comes with its own edit cost function. Note that the substitution functions respect $c_{\text{ns}}(x, x) = 0$ and $c_{\text{es}}(y, y) = 0$, for all $x \in \mathcal{L}_V$ and all $y \in \mathcal{L}_E$.

Intuitively, $\mathrm{GED}(G, G,')$ can be defined as the minimum cost of an edit from $G$ to $G'$, where the cost of an edit path is defined as the sum of the costs of its contained edit operations (cf. Definition 10 in Appendix A for the formal definition). Although this path based definition is conceptually very simple, it is impractical for algorithmic purposes. This is because, firstly, there are infinitely many edit paths from a source to a target graph, and, secondly, one has to solve the graph isomorphism problem for recognizing an edit path as such. Thus, algorithms for GED use an alternative definition based on the concept of error-correcting matchings [4], also called node maps [7,46].

**Definition 2** (*Node Map*)**.** Let $\epsilon$ be a special symbol that denotes dummy nodes, and $G$ and $G'$ be graphs of order $n$ and $n'$, respectively. A relation $\pi \subseteq ([n] \cup \{\epsilon\}) \times ([n'] \cup \{\epsilon\})$ is called *node map* from $G$ to $G'$, if and only if the following conditions hold:

- For each $i \in [n]$, there is exactly one $k \in [n'] \cup \{\epsilon\}$ with $(i, k) \in \pi$. We denote this node by $\pi(i)$.
- For each $k \in [n']$, there is exactly one $i \in [n] \cup \{\epsilon\}$ with $(i, k) \in \pi$. We denote this node by $\pi^{-1}(k)$.

The set of all node maps from a graph of order $n$ to a graph of order $n'$ is denoted by $\Pi_{n,n'}$.

A node map $\pi \in \Pi_{n,n'}$ can be used to define for all nodes and edges of graphs $G$ and $G'$ of order $n$ and $n'$ whether they have to be substituted, deleted, or inserted. Let $i, j \in [n]$, $k, l \in [n']$, and $A_{i,j} = A'_{k,l} = 1$. If $\pi(i) = k$, $i$ has to be substituted by $k$; if $\pi(i) = \epsilon$, $i$ has to be deleted; and if $\pi^{-1}(k) = \epsilon$, $k$ has to be inserted. Similarly for the edges: If $(\pi(i), \pi(j)) = (k, l)$, $(i, j)$ is substituted by $(k, l)$; if $A'_{\pi(i),\pi(j)} = 0$, $(i, j)$ has to be deleted; and if $A_{\pi^{-1}(k),\pi^{-1}(l)} = 0$, $(k, l)$ has to be inserted. The node map $\pi$ hence induces the edit cost

$$c(\pi, G, G') := c_V(\pi, \varphi, \varphi') + c_E(\pi, A, \Phi, A', \Phi'),$$

with node edit costs $c_V(\pi, \varphi, \varphi')$ and edge costs $c_E(\pi, A, \Phi, A', \Phi')$. The node costs are defined as

$$\begin{aligned} c_V(\pi, \varphi, \varphi') := &\sum_{i\in[n]} \delta_{\pi(i)} c_{\mathrm{ns}}(\varphi_i, \varphi'_{\pi(i)}) \\ &+ (1 - \delta_{\pi(i)}) c_{\mathrm{nd}}(\varphi_i) \\ &+ \sum_{k\in[n']} (1 - \delta_{\pi^{-1}(k)}) c_{\mathrm{ni}}(\varphi'_k), \end{aligned}$$

where $\delta_l$ equals 1 just in case $l \neq \epsilon$, and 0, otherwise [33]. Similarly, the edge costs are defined as

$$\begin{aligned} &c_E(\pi, A, \Phi, A', \Phi') := \\ &\alpha \cdot \Bigg[ \sum_{i\in[n]} \sum_{j\in[n]} \Big[ \delta_{\pi(i)}\delta_{\pi(j)} \big[ A_{i,j} A'_{\pi(i),\pi(j)} c_{\mathrm{es}}(\Phi_{i,j}, \Phi'_{\pi(i),\pi(j)}) \\ &+ A_{i,j}(1 - A'_{\pi(i),\pi(j)}) c_{\mathrm{ed}}(\Phi_{i,j}) \\ &+ (1 - A_{i,j}) A'_{\pi(i),\pi(j)} c_{\mathrm{ei}}(\Phi'_{\pi(i),\pi(j)}) \big] \\ &+ (1 - \delta_{\pi(i)}\delta_{\pi(j)}) A_{i,j} c_{\mathrm{ed}}(\Phi_{i,j}) \Big] \\ &+ \sum_{k\in[n']} \sum_{l\in[n']} (1 - \delta_{\pi^{-1}(k)}\delta_{\pi^{-1}(l)}) A'_{k,l} c_{\mathrm{ei}}(\Phi'_{k,l}) \Bigg], \end{aligned}$$

where $\alpha := 1/2$, if the graphs are undirected, and $\alpha := 1$, otherwise [33].

It has been shown that, under mild constraints that can be assumed to hold w.l.o.g. [4,5,7,46], the following node map based definition of GED is equivalent to the intuitive, edit path based definition:

**Definition 3** ($\mathrm{GED}$ – *Operational Definition*)**.** The *graph edit distance* from a graph $G$ of order $n$ to a graph $G'$ of order $n'$ is defined as $\mathrm{GED}(G, H) := \min_{\pi\in\Pi_{n,n'}} c(\pi, G, G')$.

Definition 3 renders GED algorithmically accessible, because node maps can be handled more easily than edit paths. In particular, it implies that each node map yields an upper bound for GED.

### 3.3. Generalized median graphs and sum of distances

Definition 1 given above provides a very intuitive understanding of the generalized median graph problem, but does not provide us with a starting point for an efficient heuristic. Let $G \in \mathbb{G}$ be a graph of order $n$ and $\mathcal{G} := \{G^p \mid p \in [|\mathcal{G}|]\} \subseteq \mathbb{G}$ be a finite collection of graphs. To come up with an alternative for Definition 1, we use the node map based definition of GED to re-write $\mathrm{SOD}(G, \mathcal{G})$ as a sum of independent minimizations. To this purpose, we define the sum of distances from a graph $G$ to a collection of graphs $\mathcal{G}$ *given fixed node maps from G to all graphs contained in* $\mathcal{G}$ as

$$\mathrm{SOD}(\boldsymbol{\pi}, G, \mathcal{G}) := \sum_{p\in[|\mathcal{G}|]} c(\pi^p, G, G^p),$$

where $\boldsymbol{\pi} \in \Pi_n^{\mathcal{G}} := \Pi_{n,n^1} \times \cdots \times \Pi_{n,n^{|\mathcal{G}|}}$ is a vector that contains a node map $\pi^p$ from $G$ to each graph $G^p \in \mathcal{G}$. We can then express $\mathrm{SOD}(G, \mathcal{G})$ as

$$\mathrm{SOD}(G, \mathcal{G}) = \min_{\boldsymbol{\pi}\in\Pi_n^{\mathcal{G}}} \mathrm{SOD}(\boldsymbol{\pi}, G, \mathcal{G}),$$

which immediately implies that $G^\star \in \mathbb{G}$ is a generalized median for $\mathcal{G}$ if and only if we have

$$\mathrm{SOD}(G^\star, \mathcal{G}) = \min_{n\in\mathbb{N}} \min_{\boldsymbol{\pi}\in\Pi_n^{\mathcal{G}}} \min_{G\in\mathbb{G}_n} \mathrm{SOD}(\boldsymbol{\pi}, G, \mathcal{G}),$$

Recalling that the constituents $A$, $\varphi$, and $\Phi$ of a graph $G = (A, \varphi, \Phi)$ are defined independently, we eventually obtain the following alternative characterization of a generalized median graph:

**Corollary 1.** *A graph $G^\star \in \mathbb{G}$ is a generalized median graph for a finite collection of graphs $\mathcal{G} \in \mathbb{G}$ in the sense of Definition 1, if and only if the following equation holds:*

$$\mathrm{SOD}(G^\star, \mathcal{G}) = \min_{n\in\mathbb{N}} \min_{\boldsymbol{\pi}\in\Pi_n^{\mathcal{G}}} \min_{\substack{A\in\mathcal{A}_n \\ \varphi\in\mathcal{L}_V^n \, \Phi\in\mathcal{L}_E^{n\times n}}} \mathrm{SOD}(\boldsymbol{\pi}, G, \mathcal{G})$$

## 4. Theoretical results

In this section, we prove several theoretical properties of the problem of computing generalized median graphs. All proofs of the presented theorems and propositions are contained in the appendix.

First, we show that generalized median graphs exist if median node and edge labels exist and the insertion and deletion costs are constant. More precisely, we prove the following Theorem 1:

**Theorem 1.** *If the conditions (C1) and (C2) hold, then the generalized median graph problem has a solution.*

*(C1) The deletion and insertion costs $c_{\mathrm{nd}}$, $c_{\mathrm{ni}}$, $c_{\mathrm{ed}}$, and $c_{\mathrm{ei}}$ are constant.*
*(C2) The problem $\min_{x\in\mathcal{L}} \sum_{y\in L} c(x, y)$ has a solution for any non-empty multiset $L \subseteq \mathcal{L}$ and each $(\mathcal{L}, c) \in \{(\mathcal{L}_V, c_{\mathrm{ns}}), (\mathcal{L}_E, c_{\mathrm{es}})\}$.*

The following Proposition 1 states that (C2) is also necessary for the existence of generalized median graphs. If it does not hold, we can construct instances that do not have a solution.

**Proposition 1.** *If (C2) is not guaranteed to hold, then there is a space of graphs $\mathbb{G}$, edit costs $c_{\mathrm{ns}}$, $c_{\mathrm{nd}}$, $c_{\mathrm{ni}}$, $c_{\mathrm{es}}$, $c_{\mathrm{ed}}$, and $c_{\mathrm{ei}}$, and a collection of graphs $\mathcal{G} \subset \mathbb{G}$, such that the generalized median graph problem does not have a solution.*

Next, we show that generalized median graphs are in general non-unique. More precisely, we prove that it is possible to construct arbitrarily large instances of the generalized median graph problem such that each graph in the collection solves the problem.

**Proposition 2.** *There is a space of graphs $\mathbb{G}$, edit costs $c_{\mathrm{ns}}$, $c_{\mathrm{nd}}$, $c_{\mathrm{ni}}$, $c_{\mathrm{es}}$, $c_{\mathrm{ed}}$, and $c_{\mathrm{ei}}$, such that, for each $n \in \mathbb{N}$, there is a collection $\mathcal{G} \subset \mathbb{G}$ with $|\mathcal{G}| = 2^{\binom{n}{2}}$, such that each graph $G \in \mathcal{G}$ is a generalized median graph for $\mathcal{G}$.*

We also provide a formal proof of the *NP*-hardness of the problem of computing generalized median graphs. In [14], it is claimed that this result follows from the *NP*-hardness of GED. This, however, is not the case. The reason for this is that, while the *NP*-hardness of GED indeed immediately implies that the problem of computing the optimal SOD is *NP*-hard (see proof of Proposition 3 in Appendix D), it could in principle be the case that the generalized median graph which yields the optimal SOD can be found in polynomial time. Since computing $\mathrm{SOD}(G, \mathcal{G})$ is *NP*-hard even for fixed $G \in \mathbb{G}$, this would not contradict the *NP*-hardness of the problem of computing the optimal SOD. The following Theorem 2 states that this is not the case, i. e., that computing a generalized median graph is *NP*-hard even if we are not interested in its SOD.

**Theorem 2.** *Even for unlabeled graphs and uniform edit costs, the problem of computing a generalized median graph is NP-hard.*

Finally, we show that the closely related problem of computing the optimal SOD is not only hard to solve exactly, but also hard to approximate.

**Proposition 3.** *Even for unlabeled graphs and constant edit costs, the problem of computing the optimal* SOD *is APX-hard and there is no polynomial time $\alpha$-approximation algorithm for any $\alpha$, unless the graph isomorphism problem is in P.*

The question whether the graph isomorphism problem is in *P* has been open for decades; the fastest currently available algorithm runs in quasi-polynomial time [47]. In fact, it is a prime candidate for an *NP*-intermediate problem that is neither in *P* nor *NP*-complete. In view of this and because of Proposition 3, it is hence unrealistic to aim at polynomial time algorithms that compute approximate median graphs whose SODs can be bounded w. r. t. the optimum. All one can reasonably strive for are heuristics that yield tight SODs in practice.

## 5. `GMG-BCU`: A scalable algorithm for estimating generalized median graphs

In this section, we present the main contribution of this paper: the local search based generalized median graph estimator `GMG-BCU`. We first give an overview of the algorithm (Section 5.1), then provide detailed explanations for the most important subroutines (Sections 5.2 to 5.3), and eventually discuss initialization strategies, termination criteria, convergence, and runtime complexity (Sections 5.4 to 5.6). `GMG-BCU` extends and improves the algorithm presented in [33] by introducing an extra optimization step w. r. t. the order of the median graph instead of optimizing only within a fixed order, and by considering the initialization as an input of the algorithm (the algorithm presented in [33] always uses medoid initialization).

### *5.1. Overview and decomposition into subproblems*

Block coordinate update (BCU) is a standard optimization technique that considers blocks of variables, each representing a subproblem. Starting with initial values for at least one block, it alternately optimizes w. r. t. each block and repeats the process until stability. In our setting, the reformulation of the generalized median problem given in Corollary 1 naturally yields the following three blocks of variables:

- The order $n \in \mathbb{N}$ of the median graph.
- The vector of node maps $\boldsymbol{\pi} \in \Pi_n^{\mathcal{G}}$ from the median to the graphs in the collection.
- The median graph's adjacency matrix $A \in \mathcal{A}_n$, node labels $\varphi \in \mathcal{L}_V^n$, and edge labels $\Phi \in \mathcal{L}_E^{n \times n}$.

Starting from an initial estimated median graph $G^\star$, our algorithm `GMG-BCU` hence minimizes the SOD by sequentially solving the following subproblems until a termination criterion is met:

(P1) Update the node maps $\boldsymbol{\pi} := (\pi^p)_{p \in [|\mathcal{G}|]} \in \Pi_n^{\mathcal{G}}$, keeping the median and the order fixed. That is, for all $p \in [|\mathcal{G}|]$, compute $\pi^p := \arg\min_{\pi \in \Pi_n} c(\pi, G^\star, G^p)$.
(P2) Update the median graph $G^\star := (A^\star, \varphi^\star, \Phi^\star) \in \mathbb{G}_n$, keeping the order and the node maps fixed. That is, compute $\varphi^\star := \arg\min_{\varphi \in \mathcal{L}_V^n} \sum_{p \in [|\mathcal{G}|]} c_V(\pi^p, \varphi, \varphi_p)$ and $(A^\star, \Phi^\star) := \arg\min_{(A,\Phi) \in \mathcal{A}_n \times \mathcal{L}_E^{n \times n}} \sum_{p \in [|\mathcal{G}|]} c_E(\pi^p, A, \Phi, A_p, \Phi_p)$.
(P3) Update the order $n \in \mathbb{N}$.

Each step decreases the SOD, which ensures that `GMG-BCU` converges to a local optimum. To solve (P2), we have to assume that the constraints (C1) (insertion and deletion costs are constant) and (C2) (median node and edge labels exist) hold; and that we have access to an algorithm `median-label` that computes optimal or close-to-optimal median labels. How this can be done depends on the label domains $\mathcal{L}_V$ and $\mathcal{L}_E$ and on the substitution cost functions $c_{\mathrm{ns}}$ and $c_{\mathrm{es}}$. In Section 5.2, we discuss how to implement `median-label` for standard choices of $\mathcal{L}_V$, $\mathcal{L}_E$, $c_{\mathrm{ns}}$, and $c_{\mathrm{es}}$.

Note that (C2) is really only a technical constraint, as violating it requires to choose the label domains and substitution costs in a very unnatural way (cf. Proof of Proposition 1 in D). Moreover, Proposition 1 implies that the entire problem of computing generalized median graphs is ill-posed if (C2) does not hold. In contrast to that, (C1) is a real constraint. If it does not hold, generalized median graphs might still exist, but `GMG-BCU` is not guaranteed to converge and hence should not be used. However, to the best of our knowledge, (C1) is respected in all application scenarios where GED is used to solve real-world problems [3]. Moreover, as pointed out in Section 2 above, most existing competitors make far more demanding assumptions (all graphs of the same order, real-valued scalars as node and edge labels, constant substitution costs). Under the assumption that the constraints are met, the three subproblems can be solved as follows:

*Subproblem (P1).* This subproblem is equivalent to $|\mathcal{G}|$ independent GED problems. Solving it is hence straightforward. Since exact GED algorithms are unusable even for relatively small graphs and collections, (P1) is relaxed by using a heuristic $\lceil \mathrm{GED} \rceil$ that computes a node map with induced upper bound for GED. The choice of this heuristic has a huge effect on the scalability of the method w. r. t. the size of the collection and the order of the contained graphs. In Section 7, we evaluate this effect empirically. Cf. [10] for an extensive survey on GED heuristics. Implementations of around 30 different heuristics are available in the open-source C++ library GEDLIB [48]. Note that it is mandatory that the employed GED heuristic computes a node map. Consequently, approaches such as the recently proposed GNN based methods [11,12] that only compute estimates for GED cannot be used.

*Subproblem (P2).* This subproblem is usually computationally cheaper but more interesting from an algorithmic point of view. Since (C1) holds, Lemma 2 tells us that (P2) corresponds to finding the median label of a multiset of labels, for each node and each pair of nodes of the current median. By (C2), these problems have a solution, which we compute by calling the algorithm `median-label` described in Section 5.2.

*Subproblem (P3).* This subproblem is even trickier. Contrary to the other problems, it is not possible to update the order without modifying the node maps and the current median graph. We therefore propose to either remove nodes and their incident edges or add isolated nodes, together with their labels, such that the SOD is reduced and the other variables are impacted as little as possible. The proposed algorithm, denoted by `update-order`, is detailed in Section 5.3.

*Macrostructure of the proposed algorithm.* Algorithm 1 summarizes the main steps of the proposed algorithm `GMG-BCU`. After initializing the current median $G^\star$, the vector of distances $\mathbf{d}$, and the vector of node maps $\boldsymbol{\pi}$ (lines 1 to 2), it iteratively updates the node maps (subproblem (P1), lines 4 to 6), then the median (subproblem (P2), lines 7 to 23), and finally the order (subproblem (P3), lines 24 to 25). Note that, due to the suboptimality of the heuristic $\lceil\mathrm{GED}\rceil$, the node maps are updated only if they provide better distances than the best previously computed node maps. Also note that, due to its high computational cost, the procedure `update-order` is conditionally called only if the median graph $G^\star$ has not undergone any modification in the current iteration, i.e., if a local optimum has been reached for the current order. The only exception is the very first iteration, where the `GMG-BCU` always calls `update-order`. This ensures that the algorithm converges towards a reasonable order in an early stage, which makes the algorithm a lot faster when the initial median is much smaller or larger than the optimum. Without this special treatment of the first iteration, it can happen that the algorithm unnecessarily spends a lot time to optimize the current median within the initial order, which can be far from optimal (especially if random initialization is chosen).

The computation of the median terminates once a suitably defined termination criterion is met (cf. Section 5.4 for details). Finally, the node maps and distances from the obtained median to the graphs in the collection $\mathcal{G}$ can optionally be tightened via a second heuristic $\lceil\mathrm{GED}\rceil'$ that produces tighter upper bounds than the heuristic $\lceil\mathrm{GED}\rceil$ employed during the main while-loop (lines 27 to 30). An analysis of `GMG-BCU`'s runtime complexity is provided in Section 5.6.

**Algorithm 1:** `GMG-BCU`

```
In: Set $\mathcal{G} \subset \mathbb{G}$ of graphs, an initial graph $G_0 \in \mathbb{G}_n$.
Out: Graph $G^\star \in \mathbb{G}$ with $\mathrm{SOD}(G^\star, \mathcal{G}) \leq \mathrm{SOD}(G_0, \mathcal{G})$, vector of
     distances $\mathbf{d}$, vector of node maps $\boldsymbol{\pi}$.
1  $G^\star := (A^\star, \varphi^\star, \Phi^\star) := G_0$; $n :=$ order of $G^\star$;
2  $\mathbf{d} := (d^p)_{p\in[|\mathcal{G}|]}$; $\boldsymbol{\pi} := (\pi^p)_{p\in[|\mathcal{G}|]}$; $it := 1$;
3  while termination criterion not met do
4      for $p \in [|\mathcal{G}|]$ do
5          $\pi := \lceil\mathrm{GED}\rceil(G^\star, G^p)$;
6          if $c(\pi) < d^p$ then $d^p := c(\pi)$; $\pi^p := \pi$;
7      if nodes are labeled then
8          for $i \in [n]$ do
9              $L := \emptyset$;
10             for $p \in [|\mathcal{G}|]$ do
11                 if $\pi^p(i) \neq \epsilon$ then append $\varphi^p_{\pi^p(i)}$ to $L$;
12             $\varphi^\star_i :=$ median-label$(L, c_{\mathrm{ns}})$;
13     for $(i, j) \in [n]^2$ do
14         $L := \emptyset$;
15         for $p \in [|\mathcal{G}|]$ do
16             if $\pi^p(i) \neq \epsilon$ and $\pi^p(j) \neq \epsilon$ then
17                 if $A^p_{\pi^p(i),\pi^p(j)} = 1$ then
18                     append $\Phi^p_{\pi^p(i),\pi^p(j)}$ to $L$
19         if edges are labeled then
20             $\Phi^\star_{i,j} :=$ median-label$(L, c_{\mathrm{es}})$;
21             $s := \sum_{\phi\in L} c_{\mathrm{es}}(\Phi^\star_{i,j}, \phi)$;
22             $A^\star_{i,j} := (c_{\mathrm{ei}} + c_{\mathrm{er}})|L| - c_{\mathrm{er}}|\mathcal{G}| > s$;
23         else $A^\star_{i,j} := (c_{\mathrm{ei}} + c_{\mathrm{er}})|L| - c_{\mathrm{er}}|\mathcal{G}| > 0$;
24     if $G^\star$ unchanged in current iteration or $it = 1$ then
25         $(G^\star, \boldsymbol{\pi}, \mathbf{d}) :=$ update-order$(G^\star, \boldsymbol{\pi}, \mathbf{d}, \mathcal{G}, it)$;
26     $it := it + 1$;
27 if tightening required by user then
28     for $p \in [|\mathcal{G}|]$ do
29         $\pi := \lceil\mathrm{GED}\rceil'(G^\star, G^p)$;
30         if $c(\pi) < d^p$ then $d^p := c(\pi)$; $\pi^p := \pi$;
31 return $G^\star$, $\mathbf{d}$, $\boldsymbol{\pi}$;
```

### *5.2. Subproblem (P2): Updating the median graph within the current order*

This section details subproblem (P2) for specific types of labels and substitution cost functions. For each node $i$ and each pair of nodes $(i, j)$, this problem asks to find a label which minimizes the sum of substitution costs to all the labels of a multiset $\mathcal{M}$ ($\mathcal{L}_i$ or $\mathcal{L}_{i,j}$). That is, we have to solve

$$x^\star := \arg\min_{x\in\mathcal{L}} \sum_{y\in\mathcal{M}} c_{\mathrm{ls}}(x, y) \tag{1}$$

with $(\mathcal{M}, \mathcal{L}, c_{\mathrm{ls}}) = (\mathcal{L}_i, \mathcal{L}_V, c_{\mathrm{ns}})$ or $(\mathcal{L}_{i,j}, \mathcal{L}_E, c_{\mathrm{es}})$. We assume that we have access to an algorithm `median-label` that computes either the real median element of a multiset of labels or a good approximation of it. For updating the topology of the current median, let $x^\star :=$ `median-label`$(\mathcal{L}_{i,j}, c_{\mathrm{ls}})$ for a pair of nodes $(i, j) \in [n]^2$. Lemma 2 tells us that by updating the adjacency matrix $A^\star$ as

$$A^\star_{i,j} := \left[ (c_{\mathrm{ei}} + c_{\mathrm{ed}})|\mathcal{L}_{i,j}| - c_{\mathrm{ed}}|\mathcal{G}| > \sum_{y\in\mathcal{L}_{i,j}} c_{\mathrm{es}}(x^\star, y) \right], \tag{2}$$

we obtain a new candidate median graph which is optimal for the current order and the current node maps. This implies that the SOD decreases whenever solving subproblem (P2) leads to modifications of the current median. In the remainder of this section, we present six examples of how to implement `median-label` for some specific label types and cost functions.

*Symbolic labels with constant edit costs.* If $c_{\mathrm{ls}}(x, y) = c_{\mathrm{s}} \cdot (1 - \delta_{x=y})$, where $c_{\mathrm{s}} > 0$ is a constant, any mode of $\mathcal{M}$ solves Eq. (1). That is, we can set $x^\star := \arg\max_{x\in\mathcal{L}[\mathcal{M}]} h_{\mathcal{M}}(x)$, where $h_{\mathcal{M}} : \mathcal{M} \to \{0, \ldots, |\mathcal{M}|\}$, $h_{\mathcal{M}}(x) := \sum_{y\in\mathcal{M}} \delta_{x=y}$ is the histogram of all the labels in $\mathcal{M}$, and $\mathcal{L}[\mathcal{M}]$ is the restriction of $\mathcal{L}$ to the elements appearing in $\mathcal{M}$. This case corresponds to the standard constant costs associated with graphs that represent chemical compounds. Since the size of $\mathcal{M}$ never exceeds the size of $\mathcal{G}$, the histograms can be computed in $O(|\mathcal{G}|)$ time for each node $i$ and for each pair of nodes $(i,j)$. For each pair of nodes $(i,j)$, we also have to decide whether or not to include the respective edge in the median $G^\star$. Also note that Eq. (2) simplifies to

$$A^\star_{i,j} := \Big[(c_{\mathrm{ei}} + c_{\mathrm{ed}})|\mathcal{L}_{i,j}| - c_{\mathrm{ed}}|\mathcal{G}| > c_{\mathrm{es}}\big(|\mathcal{L}_{i,j}| - h_{\mathcal{L}_{i,j}}(x^\star)\big)\Big],$$

which implies that updating $\varphi^\star$ and $(\Phi^\star, A^\star)$ requires $O(n|\mathcal{G}|)$ and $O(n^2|\mathcal{G}|)$ time, respectively.

*Unlabeled edges.* This case can viewed as a special case of the previous one with a unique label for all the edges, e.g., with $\mathcal{L}_E = \{1\}$. In this case, we have $h_{\mathcal{L}_{i,j}}(1) = |\mathcal{L}_{i,j}|$, for each pair of nodes $(i,j)$. Therefore, Eq. (2) can be further simplified to

$$A^\star_{i,j} := \big[(c_{\mathrm{ei}} + c_{\mathrm{ed}})|\mathcal{L}_{i,j}| - c_{\mathrm{ed}}|\mathcal{G}| > 0\big].$$

*Real scalar-valued labels with absolute differences as edit costs.* Assume that $\mathcal{L} \subseteq \mathbb{R}$ and that the substitution costs are given as $c_{\mathrm{ls}}(x,y) = |x-y|$. Then any $x^\star \in [y_{\lfloor (n+1)/2\rfloor}, y_{\lceil (n+1)/2\rceil}] \cap \mathcal{L} \supseteq \{y_{\lfloor (n+1)/2\rfloor}, y_{\lceil (n+1)/2\rceil}\}$ solves Eq. (1), where $(y_s)_{s=1}^{|\mathcal{M}|}$ is an ordering of $\mathcal{M}$. If $\mathcal{L}$ is a closed interval, $x^\star$ can be set to the median $x^\star := (y_{\lfloor (n+1)/2\rfloor} + y_{\lceil (n+1)/2\rceil})/2$.

*Real vector-valued labels with (squared) euclidean distances as edit costs.* If $\mathcal{L}$ is a compact subset of $\mathbb{R}^m$, the Euclidean norm $\|x-y\|_2$ or its square $\|x-y\|_2^2$ are typical choices for the substitutions costs $c_{\mathrm{ls}}(x,y)$. For $c_{\mathrm{ls}}(x,y) = \|x-y\|_2^2$, the optimal label $x^\star$ corresponds to the geometric mean $x^\star := (\sum_{y\in\mathcal{M}} y)/|\mathcal{M}|$ of $\mathcal{M}$. For $c_{\mathrm{ls}}(x,y) = \|x-y\|_2$, $x^\star$ is $\mathcal{M}$'s geometric median. While there is no closed-form solution for the geometric median, several numerical algorithms exist. In our experiments (Section 7), we used a modified version of Weiszfeld's algorithm [49,50].

*Label sets of constant size.* If $|\mathcal{L}|$ is finite and constant w.r.t. the size $|\mathcal{G}|$ of the graph collection, Eq. (2) can be solved exactly by computing $\sum_{y\in\mathcal{M}} c_{\mathrm{ls}}(x,y)$ for each $x \in \mathcal{L}$.

*String labels with levenshtein distances as edit costs.* If $\mathcal{L}$ is the set of all strings over a finite alphabet $\Sigma$ and $c_{\mathrm{ls}}$ is the Levenshtein distance, solving Eq. (1) is *NP*-hard even if $|\Sigma| = 2$ [51]. However, several heuristics exist that compute close-to-optimal solutions in polynomial time [52,53].

### 5.3. Subproblem (P3): Updating the order

Once the current median $G^\star \in \mathbb{G}_n$ and the node maps $\boldsymbol{\pi} := (\pi^p)_{p\in[|\mathcal{G}|]}$ have been updated as described in the previous sections, the algorithm `update-order` aims at reducing the SOD by incrementally decreasing or increasing the order of $G^\star$. It does so by deleting one node and all its incident edges at a time, or by inserting an isolated node with a given label at a time. In other words, `update-order` iteratively performs a local search around $(G^\star, \pi)$ w.r.t. a minimal variation of the order. Hence, the SOD again decreases whenever `update-order` modifies the current median.

Algorithm 2 gives an overview of the algorithm. In a first step, `update-order` incrementally deletes nodes from the current median whose deletions lead to a reduced SOD (lines 2 to 6). In a second step, nodes are inserted incrementally as long as doing so decreases the SOD (lines 8 to 12). Note that the second step is carried out only in the first iteration of the main BCU or if no node was deleted in the first step. The reason for this specification is that, in the first iteration, the current median and the current node maps are still far from optimal and it might hence happen that is beneficial to both delete and insert nodes. In later iterations, this happens very rarely. Therefore, the computationally expensive insertion routine should be called only if the cheaper deletion routine did not find a node to delete. In the following Sections 5.3.1 and 5.3.2 , we give detailed descriptions of the deletion and insertion routines.

**Algorithm 2:** `update-order`

```
In: Set $\mathcal{G} \subset \mathbb{G}$ of graphs, initial graph $G^\star \in \mathbb{G}_n$, list of node
    maps $\boldsymbol{\pi}$, list of distances $\mathbf{d}$, iterator $it$.
Ensures: Updates $G^\star$, $\boldsymbol{\pi}$, and $\mathbf{d}$.
1  $n :=$ order of $G^\star$;
2  while $\exists i \in [n]$ with $\Delta^-_i < 0$ do
3      $i^\star := \arg\min_i \Delta^-_i$;
4      delete $i^\star$ from $G^\star$;
5      update $\boldsymbol{\pi}$ and $\mathbf{d}$;
6      $n := n - 1$;
7  if a node was deleted and $it > 1$ then return;
8  while $\exists (z,\varphi) \in Z \times \mathcal{L}_V$ with $\Delta^+_{z,\varphi} < 0$ do
9      $(z^\star, \varphi^\star) := \arg\min_{(z,\varphi)} \Delta^+_{z,\varphi}$;
10     add node with label $\varphi^\star$ to $G^\star$;
11     use $z^\star$ to update $\boldsymbol{\pi}$ and $\mathbf{d}$;
12     $n := n + 1$;
```

#### 5.3.1. Decreasing the order

Let $i \in [n]$ be any node of $G^\star$, and let $G^{\star-i} \in \mathbb{G}_{n-1}$ be the graph obtained from $G^\star$ by deleting $i$ and its incident edges. For each candidate median $G^{\star-i}$, we consider the collection of updated node maps $\tilde{\boldsymbol{\pi}}_i := (\tilde{\pi}^p_i)_{p\in[|\mathcal{G}|]}$. The variation of the SOD induced by removing node $i$ from $G^\star$ is hence given by

$$\Delta^-_i := \mathrm{SOD}(\tilde{\boldsymbol{\pi}}_i, G^{\star-i}, \mathcal{G}) - \mathrm{SOD}(\boldsymbol{\pi}, G^\star, \mathcal{G}).$$

Note that $\Delta^-_i$ can be obtained without computing the entire induced cost of each mapping $\tilde{\pi}^p_i \in \tilde{\boldsymbol{\pi}}_i$. A much more efficient way to compute this value consists in analyzing the local modifications of the edit operations induced by removing $i$ from $G^\star$ and shifting from node maps $\boldsymbol{\pi}$ to $\tilde{\boldsymbol{\pi}}_i$. Each node map $\pi^p$ that substitutes $i$ results in a node map $\tilde{\pi}^p_i$ that inserts it instead, and each node map $\pi^p$ that deletes $i$ results in a node map $\tilde{\pi}^p_i$ which ignore index $i$. The same holds for the edges that are incident to $i$. All these local modifications are summed up by the following formula:

$$\begin{aligned}\Delta^-_i := \sum_{p\in[|\mathcal{G}|]} \Bigg[ & \delta_{\pi^p(i)}\big(c_{\mathrm{ni}} - c_{\mathrm{ns}}(\varphi^\star_i, \varphi^p_{\pi^p(i)})\big) - (1-\delta_{\pi^p(i)})c_{\mathrm{nd}} \\ & + \sum_{j\in[n]} A^\star_{i,j}\Big[\delta_{\pi^p(i)}\delta_{\pi^p(j)}A^p_{\pi^p(i),\pi^p(j)}\big(c_{\mathrm{ei}} \\ & - c_{\mathrm{es}}(\phi_{i,j}, \phi^p_{\pi^p(i)\pi^p(j)})\big) \\ & - c_{\mathrm{ed}}\big(1 - \delta_{\pi^p(i)}\delta_{\pi^p(j)}A^p_{\pi^p(i),\pi^p(j)}\big)\Big]\Bigg]\end{aligned}$$

#### 5.3.2. Increasing the order

Checking whether augmenting the order of the current median might yield a lower SOD proves a more complex task. We allow the algorithm to add a single isolated node to the median, and the objective is to infer both the label and the assignments of the new node that minimize the variation of the SOD. If the minimal variation is negative, a new node will be added to the current median. For each graph $G^p \in \mathcal{G}$ and its current node map $\pi^p$, we define $I^p$ as the set of indices of nodes that are inserted in $\pi^p$, plus the dummy index $\epsilon$. Let $Z = I^1 \times I^2 \times \cdots \times I^{|\mathcal{G}|}$. Each element $z := \{z^1, \ldots, z^{|\mathcal{G}|}\} \in Z$ encodes a configuration that describes how to map the new potential node in each graph of the dataset, where $z^p$ is the index of the node in $G^p$ to which the new potential node in $G^\star$ is mapped. In order to check whether augmenting the order of $G^\star$ yields a lower SOD, one should identify the couple $(z^\star, \varphi^\star) \in Z \times \mathcal{L}_V$ defined as follows:

$$(z^\star, \varphi^\star) := \arg\min_{(z,\varphi)\in Z\times\mathcal{L}_V} \Delta^+_{z,\varphi} \tag{3}$$

$$\Delta^+_{z,\varphi} := \sum_{\substack{p\in[|\mathcal{G}|]\\ z^p=\epsilon}} c_{\text{nd}} + \sum_{\substack{p\in[|\mathcal{G}|]\\ z^p\neq\epsilon}} \left(c_{\text{ns}}(\varphi,\varphi^p_{z^p}) - c_{\text{ni}}\right) \tag{4}$$

$\Delta^+_{z,\varphi}$ denotes the variation of the SOD induced by adding a node with index $|G^\star|+1$ and label $\varphi$ to $G^\star$ and setting $\pi^p(|G^\star|+1) := z^p$ in each assignment $\pi^p$. In other words, if $\Delta^+_{z,\varphi} < 0$ for some couple $(z,\phi)$ (and *a fortiori* for the optimal couple $(z^\star,\varphi^\star)$), the SOD can be lowered by applying the modifications we just described.

The problem described by Eqs. (3) and (4) is intractable in the general case, but some efficient strategies can be derived for some specific labels and cost functions. We now present a polynomial optimal strategy for symbolic labels and an efficient heuristic strategy for the general case.

*Symbolic node labels with constant edit costs.* For graphs with symbolic node labels and constant node substitution costs $c_{\text{ns}}$ it is possible to identify an optimal couple $(z^\star,\varphi^\star)$ in polynomial time. Let $z(\varphi)$ be the number of nodes with label $\varphi$ in a configuration $z$, and $z(\varepsilon)$ be the number of dummy nodes in it. Eq. (4) can then be rewritten as follows:

$$\Delta^+_{z,\varphi} := z(\varepsilon)c_{\text{nd}} + (|\mathcal{G}| - z(\varepsilon))(c_{\text{ns}} - c_{\text{ni}}) - z(\varphi)c_{\text{ns}}$$

If we consider $z$ as a fixed parameter, the above expression is clearly minimized when $z(\varphi)$ is maximal. In other words, for any given configuration $z$, the label $\varphi^\star(z)$ that minimizes $\Delta^+_{z,\varphi}$ is the most frequent label among nodes of $z$. This implies that we have $z^\star = \arg\min_{z\in Z}\Delta^+_z$, where

$$\Delta^+_z := z(\varepsilon)c_{\text{nd}} + (|\mathcal{G}| - z(\varepsilon))(c_{\text{ns}} - c_{\text{ni}}) - z(\varphi^\star(z))c_{\text{ns}}$$

is the impact of the couple $(z,\varphi^\star(z))$ on the SOD.

Since $\Delta^+_z$ is minimal if and only if $z(\varphi^\star(z))$ is maximal, an optimal configuration $z^\star$ must contain as many occurrences of the label $\varphi^\star(z)$ as possible. For a couple $(z^\star,\varphi^\star)$ to be optimal, $\varphi^\star$ must hence be selected as a label that appears in the greatest number of sets $I^p$. Moreover, for each set $I^p$ that contains a node $k$ with label $\varphi^\star$, we must set $z^{\star p} := k$.

For the sets $I^p$ that do not contain any node with label $\varphi^\star$, the cases $c_{\text{ns}} \le c_{\text{nd}} + c_{\text{ni}}$ and $c_{\text{ns}} > c_{\text{nd}} + c_{\text{ni}}$ have to be distinguished. In the first case, $\Delta^+_z$ positively depends on $z(\varepsilon)$. In order to minimize $\Delta^+_{z^\star}$, an optimal configuration $z^\star$ must hence contain as few dummy nodes as possible. This is achieved by setting $z^{\star p} := \epsilon$, if $I^p = \{\epsilon\}$, and arbitrarily setting $z^{\star p} := k$, if $I^p$ contains a node $k \neq \epsilon$. In the second case, $\Delta^+_z$ negatively depends on $z(\varepsilon)$. In order to minimize $\Delta^+_{z^\star}$, an optimal configuration $z^\star$ must contain as many dummy nodes as possible. This is achieved by setting $z^{\star p} := \epsilon$ for all remaining sets $I^p$ that do not contain a node with label $\varphi^\star$.

The whole procedure runs in $O(\sum_{p\in|\mathcal{G}|}|I^p|)$ time, as each set $I^p$ must be scanned in order to identify the label $\varphi^\star$ that appears in the greatest number of sets.

*General case.* For the general case, we make use of the fact that, for each fixed $\varphi \in \mathcal{L}_V$, the optimal configuration $z_\varphi$ can be constructed as follows: For each $p \in [|\mathcal{G}|]$, if $I_p = \{\epsilon\}$, we must set $z^p_\varphi := \epsilon$. Otherwise, let $k^\star := \arg\min_{k\in I^p\setminus\{\epsilon\}} c_{\text{ns}}(\varphi,\varphi^p_k)$. The minimization of $\Delta^+_{z,\varphi}$ imposes to set $z^p_\varphi$ to $\epsilon$ if $c_{\text{nd}}+c_{\text{ni}} \le c_{\text{ns}}(\varphi,\varphi^p_{k^\star})$ and to $k^\star$ otherwise. Moreover, for each fixed configuration $z \in Z$, the optimal node label $\varphi_z$ can be computed as the median of the labels $\{\varphi^p_{z^p} \mid p \in [|\mathcal{G}|] \wedge z^p \neq \epsilon\}$ of all nodes that are substituted under $z$. That is, inferring $\varphi_z$ amounts to computing a median label, as described in Section 5.2. Given a set $C = \{\varphi_1,\ldots,\varphi_K\}$ of initial candidate labels, we hence obtain an estimate of the optimal solution by running the following BCU from each candidate:

(Step 1) Compute optimal configuration $z := \arg\min_{\tilde{z}\in Z}\Delta^+_{\tilde{z},\varphi}$ for current node label $\varphi$.

(Step 2) Compute optimal node label $\varphi := \arg\min_{\tilde{\varphi}\in\mathcal{L}_V}\Delta^+_{z,\tilde{\varphi}}$ for current configuration $z$.

The initial candidate set $C$ is obtained by running a $K$-medians algorithm on the set $\bigcup_{p\in[|\mathcal{G}|]}\{\varphi^p_k \mid k \in I^p \wedge k \neq \epsilon\}$ of labels of all inserted nodes. Once all BCUs have terminated, the best encountered couple is returned.

## 5.4. Termination criteria and convergence

The algorithm GMG-BCU presented in the previous section can be run with various termination criteria. We consider the following ones:

- *Global convergence*: stop if no improving node map is found and the median $G^\star$ has not been modified in the current iteration.
- *Median-wise convergence*: stop if the median $G^\star$ has not been modified for a constant number $I'$ of consecutive iterations.
- *Time limit*: stop if a time limit $T$ has been reached.
- *Maximal number of iterations*: stop if a maximal number of iterations $I$ has been reached.

GMG-BCU should always be run with the global and the median-wise convergence criteria. Note that the median-wise convergence criterion ensures that the descent stops when it has converged median-wise, even if better and better node maps can still be found. This may happen with complex GED instances and non deterministic GED heuristics. Initial tests showed that it is a good choice to set the number of consecutive iterations without modification of the median to $I' := 3$.

The time limit $T$ and the maximal number of iterations $I$ can be used in addition to the convergence criteria to achieve a desired tradeoff between runtime and accuracy. Since GMG-BCU is an anytime algorithm, it always returns a solution even if interrupted before convergence. However, the experiments reported in Section 7 indicate that enforcing a maximum number of iterations is usually not required, as GMG-BCU typically converges after few iterations. The following Proposition 4 states that, even if run only with the global convergence criterion, GMG-BCU is guaranteed to converge after finitely many iterations.

**Proposition 4.** *If run with the global convergence criterion, GMG-BCU converges after at most $\lfloor(\text{SOD}(G_0,\mathcal{G}) - \text{SOD}^\star)/\varepsilon\rfloor$ iterations, where $G_0$ is the initial candidate graph, $\text{SOD}^\star$ is the optimal SOD, and $\varepsilon$ is the convergence threshold used by the global convergence criterion.*

## 5.5. Initialization

Since GMG-BCU is a local search algorithm, it can be run with various strategies for generating the initial candidate $G_0$. In this paper, we consider the following strategies:

- The medoid or set-median of the collection $\mathcal{G}$.
- Graphs with minimum, maximum, and mean order in the collection $\mathcal{G}$.
- A multi-start version which randomly picks a constant number $\kappa$ of initial graphs from $\mathcal{G}$, applies BCU to each of them, and selects a resulting graph with a minimum SOD.

Initializing with the medoid has the advantage that the initial SOD is already pretty low. However, it is costly in terms of runtime, since all pairwise distances between graphs in $\mathcal{G}$ must be computed. This can be avoided if we initialize with a graph of minimum, maximum, or mean order — of course, at the price of a higher initial SOD. The multi-start version yields a good tradeoff between runtime and accuracy. It is faster than medoid initialization (cf. Section 5.6) and achieves a better SOD than initializing with just one graph.

### 5.6. Complexity analysis

Let $n^{\max} := \max_{p\in[|\mathcal{G}|]} n^p$ be the size of the largest graph in the collection $\mathcal{G}$, and $n^\star$ be the maximal size of the median $G^\star$ as the BCU runs. Note that, although for very exotic cost functions (deletion for free) it might happen that $n^\star = O(n^{\max}|\mathcal{G}|)$, we usually have $n^\star = O(n^{\max})$. Furthermore, let $I^\star$ be the maximal number of iterations required by one of the descents, and $\omega(n, n')$ and $\omega'(n, n')$ be the complexities of upper bounding the GED from a graph of order $n$ to a graph of order $n'$ using the heuristics $\lceil\text{GED}\rceil$ and $\lceil\text{GED}\rceil'$ employed, respectively, during the BCU and during the final tightening phase. Finally, let $\omega''(n^\star, n^{\max}, |\mathcal{G}|)$ be the complexity of updating the node and edge labels as well as the order in one iteration of the BCU. Then we can state the following proposition:

**Proposition 5.** *GMG-BCU runs in* $O(\delta_{\text{medoid}}\omega(n^{\max}, n^{\max})|\mathcal{G}|^2 + \kappa I^\star[\omega(n^\star, n^{\max})|\mathcal{G}| + \omega''(n^\star, n^{\max}, |\mathcal{G}|)] + \delta_{\text{tightening}}\omega'(n^\star, n^{\max})|\mathcal{G}|)$ *time, where* $\kappa$ *denotes the number of initial candidate graphs, and* $\delta_{\text{medoid}}, \delta_{\text{tightening}} \in \{0, 1\}$ *indicate, respectively, whether or not medoid initialization and final tightening are employed.*

Note that the complexity $\omega''(n^\star, n^{\max}, |\mathcal{G}|)$ depends on the label spaces and on the edit cost functions and hence cannot be specified in more detail. In the case of symbolic labels and constant edit cost functions, $\omega''$ is linear in $|\mathcal{G}|$. The same holds if the sizes of the label spaces are constant w. r. t. $|\mathcal{G}|$. In the case of real-valued vectors, $\omega''$ comprises the complexity of Weiszfeld's algorithm and $K$-medians estimation for real-valued vectors. By enforcing a constant number of iterations, these subroutines can be set up to run in linear time w. r. t. $|\mathcal{G}|$, too. We hence obtain the following corollary, which states that GMG-BCU can be configured to run in linear time w. r. t. the size of the collection $\mathcal{G}$:

**Corollary 2.** *Assume that median node and edge labels can be computed in linear time w. r. t.* $|\mathcal{G}|$*, which, for instance, is the case if the labels are symbolic, if they are real-valued vectors, or if the label spaces are of constant size w. r. t.* $|\mathcal{G}|$*. Then GMG-BCU runs in linear time w. r. t.* $|\mathcal{G}|$ *if a constant maximal number of iterations* $I$ *is enforced and GMG-BCU is run with any initialization strategy other than medoid initialization.*

## 6. Applications

In this section, we describe how to use our median graph estimator GMG-BCU for an application from bioinformatics, namely, the computation of representative graphs for differential analysis of microbiome data (Section 6.1). We also discuss how to use GMG-BCU as a subroutine in higher-order applications such as graph clustering (Section 6.2), classification (Section 6.3), and indexing (Section 6.4). Note that, since the main contribution of this paper is the heuristic GMG-BCU itself, these discussions will necessarily remain incomplete. They are mainly intended as proofs of concept which show that our new scalable algorithm GMG-BCU opens new possibilities for employing generalized median graphs in various application scenarios.

### 6.1. Computing representative graphs for differential microbiome data analysis

In differential microbiome analysis, microbiome data is used to gain insights into pathological traits. For instance, given the relative abundances of different types of microbes present in stool samples, recent works have individuated microbes with explanatory value for clinical variables such as case vs. control or treatment response vs. no response in type 2 diabetes [54] or inflammatory bowel disease [55]. Since microbiome data is compositional, the relative abundances are incomparable across samples [56]. For this reason, the raw input data is usually normalized into log-transformed co-occurrence ratios [57], which is then fed into for downstream data analysis algorithms.

One commonly used approach for further processing the preprocessed data is the analysis of co-occurrence networks, where the nodes are annotated with the microbe types and the log-transformed co-occurrence ratios are used as edge weights [57–59]. However, existing methods based on co-occurrence networks have been shown to perform poorly in practice, which can possibly be explained by the fact that microbial types are modeled in a binary fashion (identical vs. non-identical) and gradual (dis-)similarities between them are hence ignored [60]. A natural way to overcome this problem is to define node and edge substitution costs as, respectively, normalized phylogenetic distances (i. e., genetic dissimilarities) between the microbes and absolute differences between the normalized log-transformed co-occurrence ratios. Representatives for all classes of the clinical variable of interest (e. g., one median each for controls and cases) can then be computed as generalized median graphs, and subsequently be used in downstream differential graph analysis.

Of course, this approach can be successful only if the obtained generalized median graphs are indeed good representatives of their classes, that is, if their SODs are as tight as possible. To achieve tight SODs, it is crucial that the employed heuristic can handle the features of the input graphs, i. e., is designed for graphs with varying order, constantly many symbolic node labels, and continuous edge labels. This is the case for our algorithm GMG-BCU but for none of the existing competitors.

### 6.2. Clustering

Let $\mathbb{O}$ be a collection of objects (real-valued vectors, strings, graphs, etc.), and $d_{\mathbb{O}} : \mathbb{O} \times \mathbb{O} \to \mathbb{R}$ be a distance measure. Two classical techniques for decomposing a collection $\mathcal{O} \subset \mathbb{O}$ into $K$ disjoint clusters are $K$-medoids and $K$-medians. $K$-medoids (suboptimally) computes a $K$-sized set $\mathcal{F}^\star = \{o_i^\star \mid i \in [K]\}$ of focal objects $o_i^\star \in \mathcal{O}$ that minimizes the expression $\text{SOD}(\mathcal{F}, \mathcal{O}) := \sum_{o\in\mathcal{O}} \min_{o_i\in\mathcal{F}} d_{\mathbb{O}}(o_i, o)$ among all $K$-sized sets $\mathcal{F} = \{o_i \mid i \in [K]\} \subseteq \mathcal{O}$ of focal objects. Each object $o \in \mathcal{O}$ is assigned to its closest focal object, and the $K$ clusters are defined as the sets of objects that are assigned to the same focal object. $K$-medians works just like $K$-medoids, except for the fact that focal objects are now also allowed to be contained in $\mathbb{O} \setminus \mathcal{O}$. In the sequel, we briefly review implementations of $K$-medoids and $K$-medians (Section 6.2.1), and then discuss how to apply them to the problem of clustering a collection of graphs $\mathcal{G}$ (Section 6.2.2).

#### 6.2.1. Implementing $K$-medoids and $K$-medians

The classical implementation of $K$-medoids is PAM (partitioning around medoids) [61]. Given an initial $K$-sized set of focal objects $\mathcal{F} \subseteq \mathcal{O}$, PAM computes the pair $(o_i^\star, o^\star) \in \mathcal{F} \times (\mathcal{O} \setminus \mathcal{F})$ that minimizes the cost delta $\Delta_{\mathcal{F}}(o_i, o) := \text{SOD}(\mathcal{F}, \mathcal{O}) - \text{SOD}((\mathcal{F} \setminus \{o_i\}) \cup \{o\}, \mathcal{O})$ among all $(o_i, o) \in \mathcal{F} \times (\mathcal{O} \setminus \mathcal{F})$. If $\Delta_{\mathcal{F}}(o_i^\star, o^\star) > 0$, $\mathcal{F}$ is replaced by $(\mathcal{F} \setminus \{o_i^\star\}) \cup \{o^\star\}$ and the process iterates. Otherwise, $\mathcal{F}$ is returned. The main drawback of PAM its runtime complexity is quadratic, because $d_{\mathbb{O}}$ has to be evaluated $|\mathcal{O}|^2$ times [62].

A faster implementation of $K$-medoids has been proposed in [63]. Given an initial $K$-sized set of focal objects $\mathcal{F} \subseteq \mathcal{O}$, the following variant of Lloyd's Algorithm [64] is run until the clusters converge or maximal number of iterations has been reached: (1) Assign each object $o \in \mathcal{O}$ to its closest focal object $o_i \in \mathcal{F}$. (2) Update each focal object $o_i \in \mathcal{F}$ as the medoid of all objects $o \in \mathcal{O}$ that are currently assigned to it. This algorithm is faster in practice, but in the worst case still requires $O(|\mathcal{O}|^2)$ evaluations of $d_{\mathbb{O}}$.

$K$-medians is typically implemented in a similar fashion [65]. The only difference is that, inside Lloyd's Algorithms, the focal $o_i \in \mathcal{F}$ is set to (an estimation of) the generalized median of all objects $o \in \mathcal{O}$. In the worst case, the standard implementation of $K$-medians requires $O(\omega(|\mathcal{O}|))$ evaluations of $d_{\mathbb{O}}$, where $\omega(\kappa)$ denotes the number of distance computations required to compute the generalized median for a collection of size $\kappa$. It hence inherits its runtime complexity from the algorithm employed to compute the generalized medians: If this algorithm runs in linear time w.r.t. $|\mathcal{O}|$, then so does the standard implementation of $K$-medians.

#### 6.2.2. Clustering collections of graphs

Prima facie, the implementations of $K$-medoids presented in Section 6.2.1 can straightforwardly be applied to the task of clustering a collection of graphs $\mathcal{G}$ ($\mathbb{G}$ and GED take the roles of $\mathbb{O}$ and $d_{\mathbb{O}}$, respectively). However, running $K$-medoids requires to carry out at least $O(|\mathcal{G}|^2)$ GED computations. Because of the complexity of computing GED, this is infeasible for large collections of graphs, even if suboptimal GED heuristics are used.

If, instead of using $K$-medoids, we use $K$-medians in combination with our heuristic `GMG-BCU`, we obtain a graph clustering algorithm that can be configured to require only $O(|\mathcal{G}|)$ GED computations and hence scales to large collections of graphs. To ensure the linear runtime behavior, we just have to run our median graph estimator from a constant number of randomly generated initial solutions, enforce a maximal number of iterations both for our median graph estimator and for the $K$-medians implementation, and use $K$-means++ initialization [66] for sampling the initial focal graphs. Note that none of the other median graph estimators suggested in the literature can be set up to yield a graph clustering heuristic that runs in linear time.

### 6.3. Classification and data reduction

Assume that we are given a collection $\mathcal{G} = \{G^p \mid p \in [|\mathcal{G}|]\}$ of data graphs, each of which is known to belong to a class $cl(G^p) \in [N]$, where $N \in \mathbb{N}$. At query time, we are given a graph $H \in \mathbb{G}$ whose class is unknown, and we want to decide to which class $i \in [N]$ it should be assigned. A popular and very simple approach for this task is $k$ nearest neighbor ($k$-NN) classification. Given a query graph $H$, we first determine the $k$ graphs in $\mathcal{G}$ that are closest to $H$ w.r.t. (a heuristically computed estimation of) GED, and then assign $H$ the class that occurs most frequently among them.

A drawback of $k$-NN classification is that, unless $\mathcal{G}$ is supported by an appropriate index structure (cf. Section 6.4 below), $|\mathcal{G}|$ GED computations have to be carried out at query time. One way to mitigate this problem is to partition $\mathcal{G}$ into the subsets $\mathcal{G}_i := \{G^p \in \mathcal{G} \mid cl(G^p) = i\}$ of data graphs with class $i$, then run $K$-medians clustering on each $\mathcal{G}_i$ for some $K \ll |\mathcal{G}|/N$, and finally construct a set of representative graphs as the union $\bigcup_{i\in[N]} \mathcal{F}_i$ of the focal graphs obtained for each class. At query time, we can then run $k$-NN (or any other classification algorithm that benefits from data reduction) between the query graph $H$ and the $K \cdot N \ll |\mathcal{G}|$ representatives.

This approach is feasible for large collections of graphs only if the employed $K$-medians clustering algorithm runs sufficiently fast. In particular, it is highly desirable that the required number of GED computations is linear w.r.t. the size of the graph collection. As detailed in Section 6.2.2 above, this can be achieved by using $K$-medians clustering in combination with our algorithm `GMG-BCU`.

### 6.4. Indexing

As mentioned in the previous section, we are often interested in answering proximity queries such as $k$-NN or range queries (find all data graphs that are sufficiently close to the query graph $H$) over a collection $\mathcal{G}$ of data graphs. As argued in the previous section, one strategy to avoid that $\text{GED}(H, G^p)$ has to be computed for each data graph $G^p \in \mathcal{G}$ is to use data reduction. Another option is to pre-compute an index structure that allows quick query answering.

In the remainder of this section, we elaborate on the second option. More precisely, we show that with the help of `GMG-BCU`, one can construct metric trees [67] to index collections of graphs. We first provide exact definitions of graph proximity queries (Section 6.4.1), then briefly discuss existing index structures (Section 6.4.2) and metric trees (Section 6.4.3), and finally show how to use `GMG-BCU` based metric trees for processing graph proximity queries (Section 6.4.4).

#### 6.4.1. Graph proximity queries

Given a set of objects $\mathbb{O}$ and a distance measure $d_{\mathbb{O}} : \mathbb{O} \times \mathbb{O} \to \mathbb{R}_{\geq 0}$, $k$-NN and range queries are typically defined as follows:

**Definition 4** (*k-NN Query*)**.** Given a collection $\mathcal{O} \subset \mathbb{O}$, a query object $q \in \mathbb{O}$, and a constant $k \in \mathbb{N}$, compute a $k$-sized result set $\mathcal{R} \subseteq \mathcal{O}$ such that, for each object $o \in \mathcal{O} \setminus \mathcal{R}$, there is an object $o' \in \mathcal{R}$ with $d_{\mathbb{O}}(q, o') \leq d_{\mathbb{O}}(q, o)$.

**Definition 5** (*Range Query*)**.** Given a collection $\mathcal{O} \subset \mathbb{O}$, a query object $q \in \mathbb{O}$, and a range $\tau \in \mathbb{R}_{\geq 0}$, compute a result set $\mathcal{R} \subseteq \mathcal{O}$ such that $o \in \mathcal{R} \Leftrightarrow d_{\mathbb{O}}(q, o) \leq \tau$ holds for all objects $o \in \mathcal{R}$.

The problems with these definitions when applied to GED is that answering them requires to exactly compute the distance measure $d_{\mathbb{O}}$. This is practically impossible if $d_{\mathbb{O}} = \text{GED}$. We hence have to come up with alternative definitions that account for the exponential complexity of computing GED. For $k$-NN queries, the natural thing to do is to replace GED by a proxy $\widetilde{\text{GED}}$ (i.e., a lower and an upper bound for GED) that can be computed efficiently:

**Definition 6** (*k-NN Query for* GED)**.** Given a collection of graphs $\mathcal{G} \subset \mathbb{G}$, a query graph $H \in \mathbb{G}$, and a constant $k \in \mathbb{N}$, compute a $k$-sized result set $\mathcal{R} \subseteq \mathcal{G}$ such that, for all $G^p \in \mathcal{G} \setminus \mathcal{R}$, there is a $G^{p'} \in \mathcal{R}$ with $\widetilde{\text{GED}}(H, G^{p'}) \leq \widetilde{\text{GED}}(H, G^p)$, where $\widetilde{\text{GED}}$ is a proxy for GED.

For range queries, there are two alternatives: We can either replace GED by a proxy $\widetilde{\text{GED}}$ as before. Alternatively, we can require the query to return a partition $(\mathcal{V}, \mathcal{U}, \mathcal{F})$ of the input collection $\mathcal{G}$ such that $\mathcal{V}$ contains the graphs $G^p \in \mathcal{G}$ for which $\text{GED}(H, G^p) \leq \tau$ has been verified, $\mathcal{F}$ contains the graphs $G^p \in \mathcal{G}$ that have been filtered, and $\mathcal{U}$ contains the graphs $G^p \in \mathcal{G}$ for which we are uncertain whether or not $\text{GED}(H, G^p) \leq \tau$.

**Definition 7** (GED *Range Query — First Variant*)**.** Given a collection of graphs $\mathcal{G} \subset \mathbb{G}$, a query graph $H \in \mathbb{G}$, and a range $\tau \in \mathbb{R}_{\geq 0}$, compute a result set $\mathcal{R} \subseteq \mathcal{G}$ such that $G^p \in \mathcal{R} \Leftrightarrow \widetilde{\text{GED}}(H, G^p) \leq \tau$ holds for all $G^p \in \mathcal{G}$, where $\widetilde{\text{GED}}$ is a proxy for GED.

**Definition 8** (GED *Range Query — Second Variant*)**.** Given a collection of graphs $\mathcal{G} \subset \mathbb{G}$, a query graph $H \in \mathbb{G}$, and a range $\tau \in \mathbb{R}_{\geq 0}$, compute a partition $(\mathcal{V}, \mathcal{U}, \mathcal{F})$ of $\mathcal{G}$ such that $G^p \in \mathcal{V} \Rightarrow \text{GED}(H, G^p) \leq \tau$ and $G^p \in \mathcal{F} \Rightarrow \text{GED}(H, G^p) > \tau$ hold for all $G^p \in \mathcal{G}$.

### 6.4.2. Existing index structures

In the past years, several index structures that support GED range queries as defined in Definition 7 have been proposed in the literature [68–72]. However, all of these index structures are applicable only in very restricted scenarios:

- All existing index structures only support uniform edit costs, i. e., they assume all edit operations to have the same cost.
- Each existing index structure only supports one specific proxy GED for GED.

The first restriction constitutes a severe shortcoming, because, to the best of our knowledge, the assumption that the edit costs are uniform is not met in any application where GED is used to solve real-world problems [3]. The second restriction is highly problematic, too, because the supported proxies are much looser than other proxies for GED that have been proposed in the literature [10].

### 6.4.3. Metric trees

Metric trees are index structures that support proximity queries in pseudometric spaces $(\mathbb{O}, d_{\mathbb{O}})$. Recall that for $(\mathbb{O}, d_{\mathbb{O}})$ to be a pseudometric space, the distance $d_{\mathbb{O}}$ must respect the constraints $d_{\mathbb{O}}(o, o') \geq 0$, $d_{\mathbb{O}}(o, o') = d_{\mathbb{O}}(o', o)$, and $d_{\mathbb{O}}(o, o') + d_{\mathbb{O}}(o', o'') \geq d_{\mathbb{O}}(o, o'')$ for all $o, o', o'' \in \mathbb{O}$. It is easy to show that the following Proposition 6 holds for pseudometric spaces:

**Proposition 6.** *Let $(\mathbb{O}, d_{\mathbb{O}})$ be a pseudometric space, $\lfloor d_{\mathbb{O}} \rfloor$ and $\lceil d_{\mathbb{O}} \rceil$ be lower and upper bounds for $d_{\mathbb{O}}$, and $q, o, o_i \in \mathbb{O}$. Then we have $d_{\mathbb{O}}(q, o_i) \geq d_{\mathbb{O}}(q, o) - \lceil d_{\mathbb{O}} \rceil(o, o_i) \geq \lfloor d_{\mathbb{O}} \rfloor(q, o) - \lceil d_{\mathbb{O}} \rceil(o, o_i)$ and $d_{\mathbb{O}}(q, o_i) \leq d_{\mathbb{O}}(q, o) + \lceil d_{\mathbb{O}} \rceil(o, o_i) \leq \lceil d_{\mathbb{O}} \rceil(q, o) + \lceil d_{\mathbb{O}} \rceil(o, o_i)$.*

The key idea behind metric trees is to organize a given collection $\mathcal{O} \subset \mathbb{O}$ in a tree-like structure. At query time, the inequalities stated in Proposition 6 are employed to filter or verify branches of the search tree. Various metric trees have been suggested in the literature; cf. [67] for an extensive survey. In the remainder of this section, we illustrate metric trees by describing the so-called monotonous bisector trees (MBST) presented in [73,74]. There are more efficient metric trees, but MBSTs are conceptually simple and hence perfectly suited for illustrating how metric trees work.

**Definition 9** (*Nodes of MBSTs*)**.** Let $\mathcal{O} \subset \mathbb{O}$ be a collection of objects. A node $\mathcal{N}$ of an MBST $\mathcal{T}$ for $\mathbb{O}$ is a 5-tuple of the form $\mathcal{N} = (o, \mathcal{C}, r, \mathcal{N}_1, \mathcal{N}_2)$, where $o \in \mathbb{O}$ is a focal object, $\mathcal{C} \subseteq \mathcal{O}$ contains (pointers to) all objects covered by $\mathcal{N}$, the radius $r \in \mathbb{R}_{\geq 0}$ upper bounds $d_{\mathbb{O}}(o, o_i)$ for all $o_i \in \mathcal{C}$, and $\mathcal{N}_1$ and $\mathcal{N}_2$ are pointers to $\mathcal{N}$'s children with $\mathcal{N}_1.\mathcal{C} \cup \mathcal{N}_2.\mathcal{C} = \mathcal{C}$, $\mathcal{N}_1.\mathcal{C} \cap \mathcal{N}_2.\mathcal{C} = \emptyset$, and $\mathcal{N}_1.o = o$, or $\mathcal{N}_1 = \mathcal{N}_2 = \texttt{null}$ if $\mathcal{N}$ is a leaf node.

MBSTs are constructed recursively from root to leafs. The root is initialized as $(o, \mathcal{O}, r, \texttt{null}, \texttt{null})$, where $o$ is a suitably defined focal object (e.g., the medoid, the median, or the center of $\mathcal{O}$). If $|\mathcal{N}.\mathcal{C}|$ exceeds a size limit, a new focal object $o_2$ is computed, $\mathcal{N}.\mathcal{C}$ is partitioned into sub-clusters $\mathcal{C}_1 := \{o' \in \mathcal{N}.\mathcal{C} \mid d_{\mathbb{O}}(o, o') \leq d_{\mathbb{O}}(o_2, o')\}$ and $\mathcal{C}_2 := \mathcal{N}.\mathcal{C} \setminus \mathcal{C}_1$, the radii $r_1$ and $r_2$ are computed for $\mathcal{C}_1$ and $\mathcal{C}_2$, and $\mathcal{N}$'s children are set to $\mathcal{N}.\mathcal{N}_1 := (o, \mathcal{C}_1, r_1, \texttt{null}, \texttt{null})$ and $\mathcal{N}.\mathcal{N}_2 := (o_2, \mathcal{C}_2, r_2, \texttt{null}, \texttt{null})$. The construction routine is then recursively called on $\mathcal{N}.\mathcal{N}_1$ and $\mathcal{N}.\mathcal{N}_2$.

MBSTs naturally support range queries: In each iteration, a node $\mathcal{N} = (o, \mathcal{C}, r, \mathcal{N}_1, \mathcal{N}_2)$ is popped from a set *OPEN*, which is initialized with the root and, throughout the algorithm, contains all unprocessed nodes. Next, the distance $d_{\mathbb{O}}(q, o)$ between the query object $q$ and $\mathcal{N}$'s focal object $o$ is computed, and four cases are distinguished:

(MBST-1) $d_{\mathbb{O}}(q, o) > \tau + r$
(MBST-2) $d_{\mathbb{O}}(q, o) \leq \tau - r$
(MBST-3) $\neg(1) \wedge \neg(2) \wedge \neg(\mathcal{N}_1 = \mathcal{N}_2 = \texttt{null})$
(MBST-4) $\neg(1) \wedge \neg(2) \wedge (\mathcal{N}_1 = \mathcal{N}_2 = \texttt{null})$

Proposition 6 implies that the entire subtree rooted at $\mathcal{N}$ can be pruned, if we are in the case (MBST-1), and that it has been verified, if we are in the case (MBST-2). In the case (MBST-3), $\mathcal{N}$ is an inner node whose covered subtree cannot be pruned and has not been verified. Hence, $\mathcal{N}$'s children $\mathcal{N}_1$ and $\mathcal{N}_2$ are added to *OPEN*. In the case (MBST-4), $\mathcal{N}$ is a leaf whose covered data graphs cannot be pruned and have not been verified. Hence, $d_{\mathbb{O}}(q, o_i)$ has to be computed for all $o_i \in \mathcal{C}$.

Metric trees in general and MBSTs in particular also support $k$-NN queries. In fact, most existing $k$-NN algorithms that are designed for pseudometric spaces use range queries as their main building blocks and can hence be adapted to any metric tree [67]. In this paper, we therefore do not discuss how to use metric trees for answering $k$-NN queries, but instead refer to [67].

### 6.4.4. Using metric trees for graph proximity queries

In this section, we show how $K$-medians graph clustering allows to use metric trees for indexing graph collections. For the sake of simplicity, we again focus on MBSTs. Slight modifications of the proposed techniques can be used in combination with any metric tree that is defined for general pseudometric spaces.

If we want to use metric trees for indexing collections of graphs, we have to ensure that $(\mathbb{G}, \text{GED})$ is a pseudometric space. The following Proposition 7 states that this is indeed the case if the edit costs are symmetric. This is the case in most applications [3].

**Proposition 7.** *If (C3) holds,* GED *is a pseudometric on $\mathbb{G}$.*

(C3) *The edit costs are symmetric, i. e., we have $c_{\text{nd}} = c_{\text{ni}}$, $c_{\text{ed}} = c_{\text{ei}}$, $c_{\text{ns}}(\varphi, \varphi') = c_{\text{ns}}(\varphi', \varphi)$ for all $\varphi, \varphi' \in \mathcal{L}_V$, and $c_{\text{es}}(\varphi, \varphi') = c_{\text{es}}(\varphi', \varphi)$ for all $\varphi, \varphi' \in \mathcal{L}_E$.*

For constructing an MBST for a collection of graphs $\mathcal{G} \subset \mathbb{G}$, we initialize the root as $(G, \mathcal{G}, r, \texttt{null}, \texttt{null})$. $G$ is a heuristically computed generalized median graph for $\mathcal{G}$, and the radius $r$ is set to $r := \max_{G^p \in \mathcal{G}} \lceil \text{GED} \rceil(G, G^p)$, where $\lceil \text{GED} \rceil(G, G^p)$ is a polynomially computable upper bound for $\text{GED}(G, G^p)$ (e.g., the upper bound used by GMG-BCU). To split the collection of covered graphs of a node $\mathcal{N} = (G, \mathcal{C}, r, \texttt{null}, \texttt{null})$ with $|\mathcal{C}| > M$, we compute a new focal graph $G_2$ as a generalized median for the collection $\mathcal{C}_{\text{far}} := \{G^p \in \mathcal{C} \mid \lceil \text{GED} \rceil(G, G^p) \geq s + \rho \cdot (r - s)\}$ of graphs that are far away from $G$, where $s := \min_{G^p \in \mathcal{C}} \lceil \text{GED} \rceil(G, G^p)$ and $\rho \in (0, 1]$ is a hyper-parameter. We then construct clusters $\mathcal{C}_1 := \{G^p \in \mathcal{C} \mid \lceil \text{GED} \rceil(G, G^p) \leq \lceil \text{GED} \rceil(G_2, G^p)\}$ and $\mathcal{C}_2 := \mathcal{C} \setminus \mathcal{C}_1$, compute the radii $r_1 := \max_{G^p \in \mathcal{C}_1} \lceil \text{GED} \rceil(G, G^p)$ and $r_2 := \max_{G^p \in \mathcal{C}_2} \lceil \text{GED} \rceil(G_2, G^p)$, and set $\mathcal{N}.\mathcal{N}_1 := (G, \mathcal{C}_1, r_1, \texttt{null}, \texttt{null})$ and $\mathcal{N}.\mathcal{N}_2 := (G_2, \mathcal{C}_2, r_2, \texttt{null}, \texttt{null})$.

GED range queries in the sense of Definition 8 can then be answered as follows. Since exactly computing GED is infeasible, at query time, we compute lower and upper bounds $\lfloor \text{GED} \rfloor(H, G)$ and $\lceil \text{GED} \rceil(H, G)$, where $H$ is the query graph and $G$ is the focal graph of our current MBST node $\mathcal{N} = (G, \mathcal{C}, r, \mathcal{N}_1, \mathcal{N}_2)$. We distinguish the four cases (MBST-1) to (MBST-4) with $d_{\mathbb{O}}(q, o)$ substituted by $\lfloor \text{GED} \rfloor(H, G)$ and $\lceil \text{GED} \rceil(H, G)$ in, respectively, (MBST-1) and (MBST-2). Proposition 6 implies that, in the cases (MBST-1) and (MBST-2), we can, respectively, add $\mathcal{C}$ to the set $\mathcal{F}$ of all filtered graphs or to the set $\mathcal{V}$ of all verified graphs. In the case (MBST-3), we add $\mathcal{N}_1$ and $\mathcal{N}_2$ to *OPEN*. In the case (MBST-4), we compute $\lfloor \text{GED} \rfloor(H, G^p)$ and $\lceil \text{GED} \rceil(H, G^p)$ for each data graph $G^p \in \mathcal{C}$, and add $G^p$ to $\mathcal{F}$, if $\lfloor \text{GED} \rfloor(H, G^p) > \tau$, to $\mathcal{V}$, if $\lceil \text{GED} \rceil(H, G^p) \leq \tau$, and to $\mathcal{U}$, otherwise.

GED range queries in the sense of Definition 7 can be processed similarly. The main difference is that, depending on whether $\widetilde{\text{GED}}$ is a lower or an upper bound for GED, we can use the inner nodes $\mathcal{N} = (G, \mathcal{C}, r, \mathcal{N}_1, \mathcal{N}_2)$ only for verifying or only for pruning: If $\widetilde{\text{GED}}$ is a lower bound, we compute upper bounds $\lceil\text{GED}\rceil(H, G)$ to verify the covered data graphs; if it is an upper bound, we compute lower bounds $\lfloor\text{GED}\rfloor(H, G)$ to prune the subtree rooted an $\mathcal{N}$. Algorithms for answering GED $k$-NN queries in the sense of Definition 6 can be built upon algorithms for answering GED range queries in the sense of Definition 7. For the details, we again refer to [67].

Again note that, in theory, one could use metric trees to index collections of graphs without having access to a linear time median graph estimator such as GMG-BCU: For instance, if using MBSTs, one could define the new focal graph $G_2$ as the medoid of $\mathcal{C}_{\text{far}}$. However, just as for $K$-medoids graph clustering, the decisive disadvantage of this alternative approach is that it requires to carry out $O(|\mathcal{G}|^2)$ GED computations, which is practically infeasible for large collections of graphs.

## 7. Empirical evaluation

In this section, we report the results of an extensive empirical evaluation of GMG-BCU. In Section 7.1, we describe the setup of the experiments. In Section 7.2, we report how different initialization strategies and GED heuristics affect the performance of our algorithm. In Section 7.3, we compare our approach to state of the art methods for estimating generalized median graphs. In Section 7.4, we discuss how our algorithm performs when used for classification, clustering, and indexing as described in Section 6. Finally, Section 7.5 provides a brief summary of the most important experimental findings.

### 7.1. Experimental setup

#### 7.1.1. Datasets

We tested on three widely used benchmark datasets, two datasets containing synthetically generated graphs, and one microbiome dataset from a biomedical application. Table 3 summarizes the most important properties of the datasets. In the following paragraphs, we describe them in more detail. All datasets and edit cost functions respect constraints (C1) to (C3).

*Benchmark datasets.* We tested on the widely used benchmark datasets LETTER, AIDS, and MUTA from the IAM Graph Database Repository [75]. The graphs in AIDS and MUTA represent molecules that do or do not exhibit activity against HIV and mutagenicity, respectively. They have symbolic labels both on the nodes and on the edges. The graphs in LETTER represent distorted drawings of all capital roman letters that can be drawn with straight lines only. They have two-dimensional Euclidean node labels and unlabeled edges. We used the following edit cost functions suggested in [28,76]: For AIDS and MUTA, we used constant node and edge substitution cost functions $c_{\text{ns}} \equiv 4$ and $c_{\text{es}} \equiv 1$, node insertion and deletion costs $c_{\text{ni}} = c_{\text{nd}} = 2$, and edge deletion and insertion costs $c_{\text{ei}} = c_{\text{ed}} = 1$. For LETTER, we set the node and edge insertion and deletion costs to $c_{\text{ni}} = c_{\text{nd}} = 0.675$ and $c_{\text{ei}} = c_{\text{ed}} = 0.425$, and used two different node substitution costs, namely Euclidean costs $c_{\text{ns}}(\varphi, \varphi') = 0.75 \cdot \|\varphi - \varphi'\|_2$ and squared Euclidean costs $c_{\text{ns}}(\varphi, \varphi') = 0.75 \cdot \|\varphi - \varphi'\|_2^2$. All constants are chosen as suggested in [28,76]. In the sequel, LETTER-EUCL and LETTER-SQUARED denote, respectively, the variants of LETTER with Euclidean and squared Euclidean node substitution costs.

*Synthetically generated datasets.* We also tested on the two synthetically generated datasets S-MOL and AIDS-EDIT to evaluate scalability (S-MOL) and quality (AIDS-EDIT). The dataset S-MOL contains $5 \cdot 10^4$ synthetic molecule graphs, which were generated as follows: In a first step, the order $n$ of the graph $G$ was randomly selected from the range $\{10, \dots, 15\}$. Subsequently, we randomly generated a Prüfer sequence of length $n-2$, constructed $G$ as the corresponding tree on $n$ nodes, and randomly selected node labels (modeling atoms) from $\{1, \dots, 5\}$ and edge labels (modeling valence) from $\{1, 2\}$. For the tests on S-MOL, we used the same edit cost functions [28,76] as for AIDS and MUTA. The AIDS-EDIT dataset was generated such that we (almost) know the optimal SOD for each sub-collection. This was achieved as follows: In a first step, we randomly selected a seed graph $G_0$ from the AIDS dataset. Subsequently, we generated all graphs $H$ that can be reached from $G_0$ via exactly one edit operation, computed lower bounds for $\lfloor\text{GED}\rfloor(H_1, H_2)$ for each pair $(H_1, H_2)$ of generated graphs, and created a conflict graph $X$ with edges $(H_1, H_2)$ for all $(H_1, H_2)$ with $\lfloor\text{GED}\rfloor(H_1, H_2) = 0$. In the last step, we heuristically computed a maximum independent set $S$ for $X$ and put all 542 surviving graphs $H \in S$ in the dataset AIDS-EDIT. If used with uniform edit costs, this construction ensures that all graphs contained in AIDS-EDIT are at distance 1 from $G_0$ and at distance at least 1 from each other. Therefore, the optimal SOD for each sub-collection $\mathcal{G}$ of AIDS-EDIT is lower-bounded by $|\mathcal{G}| - 1$ and upper-bounded by $|\mathcal{G}|$.

*Microbiome dataset.* Finally, we tested on the real-world dataset IBD which contains graphs that model the gut microbiome of 133 patients with inflammatory bowel disease (IBD) and 6 controls [77]. The graphs and edit costs were derived from the raw data as described in Section 6.1 above. The IBD patients are further subdivided into a group of patients who responded to hematopoietic stem cell transplantation (HSCT) and a group of patients who did not respond to HSCT.

#### 7.1.2. Test protocol and implementation

For each dataset $\mathcal{D}$ except IBD and each $p \in \{10, 20, \dots, 90\}$, we randomly sampled five collections $(\mathcal{D}_i^p)_{i\in[5]}$ containing $p$% of the graphs. For each $p$, we ran the tests on all five collections and aggregated the results. First, we evaluated various configurations of GMG-BCU on the datasets LETTER-EUCL, AIDS, and MUTA. We then selected the three best performing configurations and compared them against the state of the art on these three datasets and additionally on LETTER-SQUARED, AIDS-EDIT, and S-MOL. Subsequently, we compared the IBD prototypes computed by the configurations of GMG-BCU to those computed by the state of the art, and preliminarily evaluated the performance of one GMG-BCU configuration within heuristics for graph clustering, classification and indexing. All methods were implemented in C++ and were parallelized to run in six threads. They are distributed on GitHub along with all datasets and test scripts: https://github.com/dbblumenthal/gedlib/. Experiments were run on a machine with 80 Intel(R) Xeon(R) Gold 6148 CPU @ 2.40 GHz processors with two cores each and 384 GB of main memory running GNU/Linux.

### 7.2. Evaluation of different configurations of GMG-BCU

In a first set of experiments, we evaluated different configurations of GMG-BCU. We used three different heuristics for computing the node maps and upper bounds $\lceil\text{GED}\rceil$ during the main while-loop of our algorithm: BRANCH-FAST [8], 2-REFINE [9], and IPFP [46,78,79]. We selected these heuristics, because it has been shown in [10] that all of them provide upper bounds that are Pareto optimal on the range from computationally-cheap-but-rather-loose to tight-but-computationally-expensive (with BRANCH-FAST yielding the cheapest and IPFP yielding

**Table 3**
Overview of test datasets.

| Dataset | # Graphs | # Classes | # Nodes | | | | |
|---|---|---|---|---|---|---|---|
| | | | min | max | *mean* | *std* | *median* |
| LETTER | 2250 | 15 | 1 | 9 | 4.7 | 1.3 | 5 |
| AIDS | 1500 | 2 | 2 | 95 | 15.7 | 13.8 | 11 |
| MUTA | 4299 | 2 | 4 | 98 | 29.2 | 14.6 | 27 |
| S-MOL | 50000 | 1 | 10 | 15 | 12.5 | 1.7 | 13 |
| AIDS-EDIT | 542 | 1 | 32 | 33 | 32.1 | 0.3 | 32 |
| IBD | 139 | 3 | 23 | 190 | 95.8 | 44.5 | 91.0 |
| Dataset | Labels | | # Edges | | | | |
| | Nodes | Edges | min | max | *mean* | *std* | *median* |
| LETTER | vectors | none | 0 | 9 | 4.5 | 1.6 | 5 |
| AIDS | symb. | symb. | 1 | 103 | 16.2 | 15.1 | 11 |
| MUTA | symb. | symb. | 3 | 105 | 30.1 | 15.4 | 28 |
| S-MOL | symb. | symb. | 9 | 14 | 11.5 | 1.7 | 12 |
| AIDS-EDIT | symb. | symb. | 32 | 34 | 33.2 | 0.4 | 33 |
| IBD | symb. | scalars | 28 | 5529 | 710.8 | 736.1 | 499 |

the tightest upper bound). We tested medoid, minimum, mean, and maximum order initialization, as well as random initialization where the number of initial solutions was varied across the set $\{1, 2, 4, 8, 16, 32\}$. Each configuration was initially run with a time limit of 10 min and was tested with and without tightening, where IPFP was chosen to compute the improved final upper bounds $\lceil \text{GED} \rceil'$. Subsequently, the three overall best configurations were run with varying time limits between 1 min and 10 min.

*Number of initial solutions for randomly initialized configuration.* Fig. 1 shows the effect of varying the number of initial solutions and the GED heuristic used during the BCU on randomly initialized configurations on the $(\mathcal{D}_i^{90})_{i\in[5]}$ collections. On LETTER, the choice of the GED heuristic had little influence on the SOD; on AIDS, 2-REFINE and IPFP yielded tighter SODs than BRANCH-FAST; and on MUTA, using 2-REFINE led to the best SODs. Using IPFP on MUTA resulted in loose SODs, because the time limit was almost always exceeded already during the first iteration of the BCU. For all GED heuristics, increasing the number of initial solutions to a value higher than eight had no significant marginal gain. For all other experiments, we therefore fixed the number of initial solutions for randomly initialized configurations to eight.

*Number of iterations.* The first row of Fig. 2 shows that the maximal number of iterations of the configurations. To improve the readability of the plots, we only show the results of those configurations that were not Pareto dominated on at least one dataset. We see that the maximal number of iterations mainly depends on the label types. Since the graphs contained in AIDS and MUTA have symbolic node and edge labels, the label domains $\mathcal{L}_V$ and $\mathcal{L}_E$ are finite, which leads to a very small number of iterations. Graphs contained in LETTER are labeled with Euclidean coordinates, i. e., $\mathcal{L}_V$ is continuous, which increases the number of iterations. In contrast to that, the average size of the graphs turns out not to increase the maximal number of iterations. Although LETTER graphs are smaller than AIDS and MUTA graphs, GMG-BCU required more iterations on LETTER-EUCL than on MUTA and AIDS. Another important observation is that, on all datasets, the number of iterations is independent of the size of the graph collection $\mathcal{G}$, of the initialization strategy, and also of the GED heuristic used during the BCU. (The low number of iterations of configurations that use medoid initialization on MUTA and AIDS is due to the fact that these methods exceeded the time limit during initialization.) Unless medoid initialization is employed, the runtime behavior of GMG-BCU is hence linear in the size of $\mathcal{G}$, even if run without a maximal number of iteration.

*Effect of algorithm used during descent.* The second row of Fig. 2 shows how the non-dominated configurations performed w. r. t. mean runtime and SOD. Interestingly, the dominated configurations include all configurations that use IPFP during the BCU, although IPFP has been shown to produce tighter upper bounds for GED than BRANCH-FAST and 2-REFINE [9,10]. The reason for this is that IPFP is by far the slowest of these heuristics and therefore very often reached the time limit already after very few iterations of the BCU. Most GED heuristics led to pretty similar SODs on all datasets; in terms of SOD, GMG-BCU is hence robust w. r. t. the choice of this heuristic. On LETTER-EUCL, the best configuration used BRANCH-FAST during the BCU; on AIDS and MUTA, the best configuration used 2-REFINE. Another interesting observation is that, on LETTER-EUCL, different GED heuristics did not have a great effect on the runtime. The reason for this is that the graphs contained in LETTER are very small and that their node labels are Euclidean coordinates (cf. Table 3). In this setting, computing node maps via any GED heuristic is fast and evaluating median-label$(\cdot)$ hence becomes the dominant operation. This operation is the same for all GED heuristics. On the other datasets, computing the node maps is the dominant operation. Therefore, configurations that use the fast GED heuristic BRANCH-FAST during the BCU and do not carry out the IPFP-based tightening clearly outperformed all other configurations in terms of runtime.

*Effect of initialization type.* The second row of Fig. 2 also shows that, on all datasets, randomly initialized configurations with tightening computed the tightest SODs, and that also among the fast configurations without tightening the best results were achieved with random initialization. In view of these findings, we included the following three configurations in the time limit tests and in the comparison against the state of the art:

| | |
|---|---|
| GMG-BCU (I) | Random initialization, BRANCH-FAST during BCU, with tightening (yielded best mean SOD on LETTER). |
| GMG-BCU (II) | Random initialization, 2-REFINE during BCU, with tightening (yielded best mean SODs on AIDS and MUTA). |
| GMG-BCU (III) | Random initialization, BRANCH-FAST during BCU, without tightening (yielded best mean SOD among fast configurations). |

*Effect of time limit.* Fig. 3 shows how varying the time limit affects the performance of the three selected configurations GMG-BCU (I), GMG-BCU (II), and GMG-BCU (III). Both on LETTER-EUCL and on MUTA, we observe that, even if interrupted before regular termination, the SODs do not increase dramatically. This implies that, if computation time is critical, GMG-BCU can safely be run

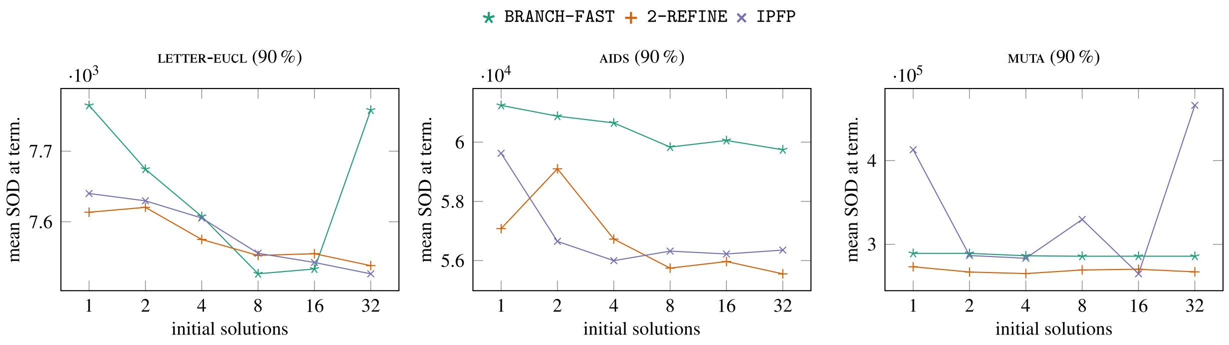

**Fig. 1.** Effect of number of initial solutions used for random initialization on mean SODs at termination with time limit set to 10 min.

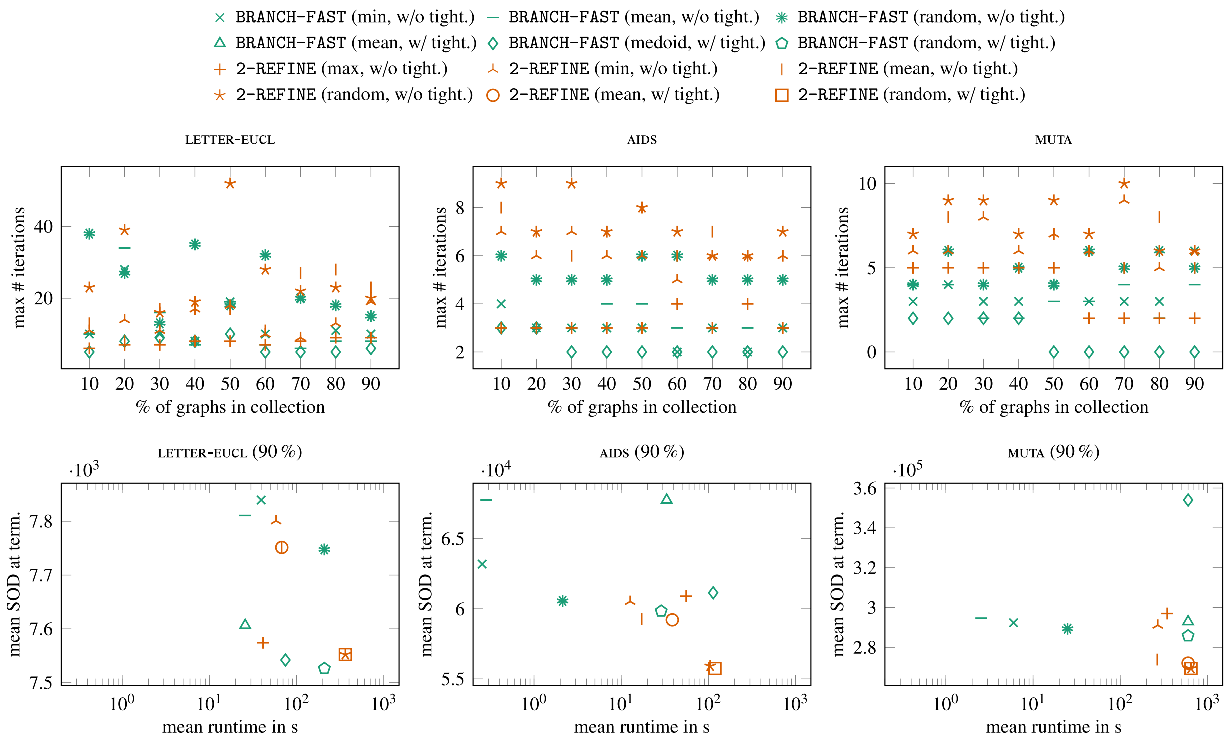

**Fig. 2.** Effect of configurations with time limit set to 10 min. Configurations that were dominated on all benchmark datasets are not displayed. Since tightening does not affect the maximal number of iterations, in the first row, we only show the results for the configurations without tightening if both the version with and the version without tightening were not dominated on all datasets.

with a small time limit and still returns an estimated median graph of good quality. On AIDS, all configurations except GMG-BCU (II) with time limits of 1 and 2 min terminated regularly.

### 7.3. Comparison against the state of the art

We compared GMG-BCU (I), GMG-BCU (II), and GMG-BCU (III) to LINEAR [18,29], BEST-LINEAR [18,29], TRIANGULAR [32], and BEST-TRIANGULAR [32]. We selected these methods, because they are the most competitive generic state of the art approaches, as detailed in Section 2. Moreover, we compared against the first prototype of our algorithm presented in [33] (denoted as PROTOTYPE in the sequel). Recall that GMG-BCU improves PROTOTYPE in that it allows initialization strategies other than medoid initialization and includes the heuristic `update-order` to optimize the order of the computed median graph.

#### 7.3.1. Runtime and scalability

Fig. 4 shows how the compared methods performed w.r.t. runtime and scalability. On all datasets except the two variants of LETTER, GMG-BCU (III) was orders of magnitudes faster than all tested competitors. Moreover, the results for AIDS and AIDS-EDIT clearly show that also the other two configurations GMG-BCU (I) and GMG-BCU (II) scale better to large collections of graphs than the competitors. On MUTA, this is less visible, because here all tested algorithms except GMG-BCU (III) reached the time limit of 10 min already on pretty small collections. On the largest dataset S-MOL, all competitors reached the time limit already on the smallest sub-collections. In contrast to that, the fast configuration GMG-BCU (III) terminated in around a minute even on the largest sub-collections containing 45 000 graphs.

The results for the two variants LETTER-EUCL and LETTER-SQUARED of the LETTER dataset are very different. Here, the tested competitors were significantly faster than the three configurations of GMG-BCU. There are two reasons for this. Firstly, the LETTER graphs are very small (cf. Table 3), which implies that computing node maps — the dominant operation in the state of the art approaches — is cheap. Secondly, the node labels of the LETTER graphs are Euclidean coordinates, i.e., the node label domain $\mathcal{L}_V$ is continuous. GMG-BCU optimizes over this domain,

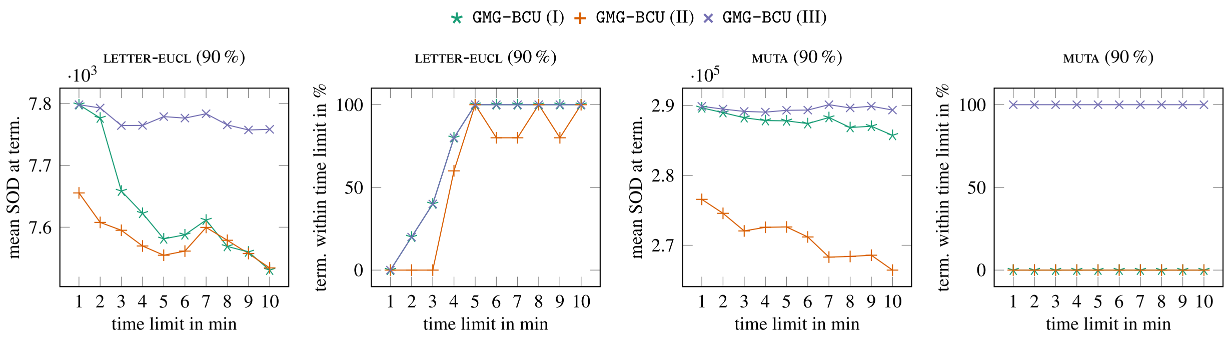

**Fig. 3.** Effect of time limit on SOD and on the number of runs which terminated regularly for configurations GMG-BCU (I), GMG-BCU (II), and GMG-BCU (III). The results for AIDS are not displayed, because here all configurations except GMG-BCU (II) with time limits of 1 and 2 min terminated regularly.

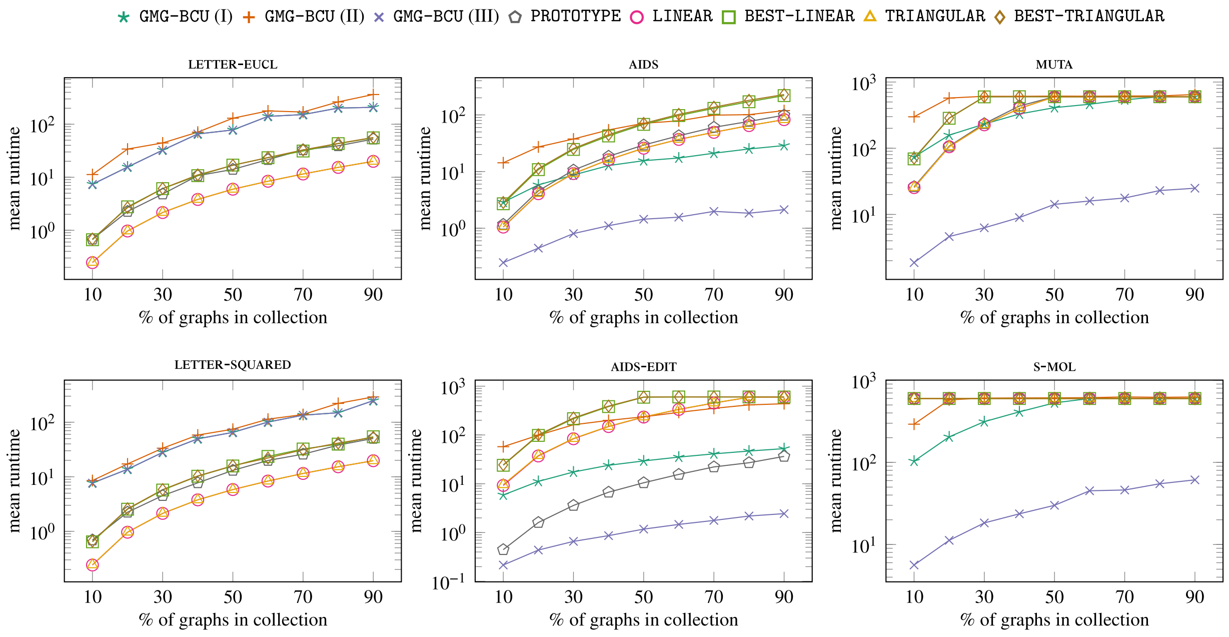

**Fig. 4.** Performance of GMG-BCU configurations w.r.t. runtime in comparison to state of the art heuristics and the first prototype of our algorithm presented in [33]. All heuristics were run with a time limit of 10 min.

which leads to an increased number of iterations w.r.t. instances with symbolic node labels (cf. Fig. 2). In contrast to that, the state of the art approaches always select the labels of the estimated median graph as (interpolations between) labels that are present in the graph collection. This improves the runtime, but negatively affects the quality of the estimated median graph, as detailed below.

When comparing the variants of GMG-BCU to the preliminary version PROTOTYPE presented in [33], we observe that PROTOTYPE was faster than all three GMG-BCU variants only on the two LETTER datasets, but was slower than GMG-BCU (III) on all other datasets and scaled worse to large graph collections than all three GMG-BCU variants. Again, this is as expected: Since PROTOTYPE uses medoid initialization and does not use update-order, it requires to compute $|\mathcal{G}|^2$ node maps for initialization and then terminates quickly as soon as the candidate median has converged within the initial order. Consequently, PROTOTYPE has a worse asymptotic runtime behavior than the three configurations of GMG-BCU and is faster only on small graph collections containing very small graphs such as LETTER-EUCL and LETTER-SQUARED.

*7.3.2. Quality*

Fig. 5 shows how varying the size of the graph collection affects the compared algorithms' deviations from the theoretically known optimal SOD (plot for AIDS-EDIT) or from the best SOD obtained by one of the tested algorithms (plots for all other datasets). As expected, the three configurations of GMG-BCU computed tighter SODs than the competitors on the two LETTER datasets with continuous node label domains. On the chemical datasets AIDS, MUTA, and S-MOL, all computed SODs were similar with GMG-BCU (II) performing slightly better than the other algorithms and configurations, on average. However, as indicated by the missing or partial curves in Fig. 5, all state of the art approaches failed to compute any results on the larger subcollections of MUTA and on all sub-collections of the large dataset S-MOL, because they reached the time limit while computing the initial GED based graph embeddings. In contrast to that, also the slower configurations GMG-BCU (I) and GMG-BCU (II) always returned an estimated median graph even if interrupted before regular termination, because GMG-BCU is an any-time algorithm. Moreover, on AIDS, MUTA, and S-MOL, all three configurations of GMG-BCU always yielded SODs which deviated from the best SOD by at most 11 %.

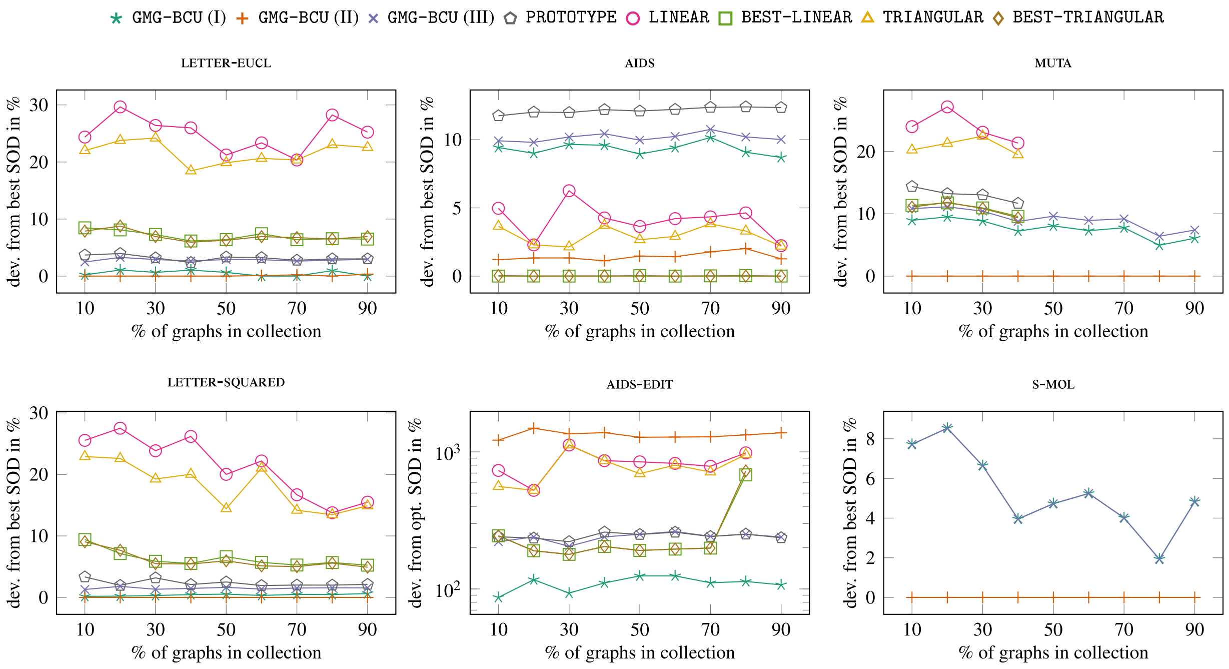

**Fig. 5.** Mean deviations from theoretically known optima (AIDS-EDIT) or best computed SODs (all other datasets). Missing or partial curves for the state of the art heuristics indicate that they reached the time limit of 10 min while computing the graph embeddings and hence did not return any candidate median. Similarly, missing curves for the prototype of our algorithm presented in [33] indicate that it reached the time limit during initialization and hence did not return a median graph.

On AIDS-EDIT, the SOD of the best performing algorithm GMG-BCU (I) exceeded the theoretically known optimal SOD by a factor of around 2 (i. e., deviated from the optimum by around 100 %). The SODs computed by the other algorithms and configurations were significantly looser. To correctly interpret these results, recall that, by construction, AIDS-EDIT is a very special dataset, since all contained graphs are extremely similar and at most one edit operation away from the optimal median. Because of this, already one incorrect node assignment in each of the node maps leads to a relative error of at least 2, even if the optimal median is found. Therefore, the empirical approximation factor of 2 is actually a pretty good result. The special structure of the AIDS-EDIT dataset also explains the poor performance of GMG-BCU (II), which yielded the tightest SODs on all but one of the other datasets. The reason is that GMG-BCU (II) uses the GED heuristic 2-REFINE, which computes node maps via local search from randomly initialized solutions. This randomized approach yields excellent results on natural GED instances [9], but fails to detect the fine-grained differences between the synthetic graphs contained in AIDS-EDIT.

The comparison of the three GMG-BCU variants to PROTOTYPE yields two findings. Firstly, PROTOTYPE failed to compute any medians on the larger sub-collections of MUTA as well as on all S-MOL sub-collections, because it reached the time limit during initialization. Secondly, and more interestingly, whenever PROTOTYPE did computed a median graph, it was outperformed in terms of quality even by the fast configuration GMG-BCU (III). This shows the effectiveness of the sub-routine update-order employed by GMG-BCU: Even though GMG-BCU (III) starts the local search from worse initial candidate medians than PROTOTYPE, it eventually finds a better solution because it is not restricted to the order of the initial candidates.

### 7.4. Performance of GMG-BCU in application scenarios

In a third set of experiments, we evaluated how GMG-BCU performs when used for computing representatives for microbiome data graphs, and when used as a sub-routine within higher-order algorithms for graph clustering, classification, and indexing. The prototype computation tests were carried out on the IBD dataset, the tests for clustering, classification, and indexing on the benchmark datasets LETTER-EUCL, AIDS, and MUTA. Just as the expositions in Section 6.2 to 6.4, the latter tests are intended as proofs of concept. Carrying out in-depth empirical evaluations is the task of future work, whose dedicated focus is the development of graph clustering, classification, or indexing techniques. For the IBD prototype tests, we used the configuration GMG-BCU (I), which yielded the best empirical approximation ratio on AIDS-EDIT. For all other application tests, we used the configuration GMG-BCU (III), i. e., the configuration that, on average, computed the tightest SODs among all fast configurations.

#### 7.4.1. Computation of representatives for IBD graphs

Table 4 shows the quality of the representatives for the three sub-groups of the IBD dataset computed by GMG-BCU and state of the art heuristics. For each sub-group, we show the obtained SODs and the deviations in percent from the best SOD. GMG-BCU computed the best representatives on all sub-groups; the SODs of the representatives computed by best competitor BEST-TRIANGULAR were between 10 % and 15 % looser. Like on the LETTER datasets, GMG-BCU outperformed the state of the art in terms of SOD, because the IBD graphs have continuous (edge) labels, and, unlike the competitors, GMG-BCU optimizes the labels of the estimated median over the continuous domain. Since the IBD dataset only contains 139 graphs, all heuristics terminated within a couple of seconds on all sub-groups.

To assess whether the obtained results are meaningful from a biological point of view, we had a closer look at the median graphs computed by GMG-BCU. It is well known that the diversity of the gut microbiome is decreased in IBD patients [80]. This is reflected in the computed median graphs: While the median graph for the control group contains 169 different OTUs, this number decreases to 67 for the HSCT responders and to 49 for the HSCT non-responders, i. e., the IBD patients with the most severe disease courses. We also had a look at the set $O_0$ of OTUs that are present in the control group median but missing

**Table 4**
Quality of representatives for IBD graphs.

| Method | Controls | | HSCT responders | | HSCT non-responders | |
|---|---|---|---|---|---|---|
| | SOD | Deviation in % | SOD | Deviation in % | SOD | Deviation in % |
| GMG-BCU | 4737.1 | 0.0 | 28005.1 | 0.0 | 11222.7 | 0.0 |
| LINEAR | 5456.2 | 15.2 | 32548.2 | 16.2 | 13384.4 | 19.3 |
| TRIANGULAR | 5456.2 | 15.2 | 32474.7 | 16.0 | 12999.9 | 15.8 |
| BEST-LINEAR | 5456.2 | 15.2 | 31207.3 | 11.4 | 12421.7 | 10.7 |
| BEST-TRIANGULAR | 5456.2 | 15.2 | 30932.7 | 10.5 | 12341.4 | 10.0 |

in the medians for the two IBD subgroups, and at the set $O_1$ of OTUs that are present in the one of the IBD medians but missing in the median for the control group. Interestingly, many OTUs contained in $O_0$ fall into the families Ruminococcaceae and Bacteroidaceae, for which reduced abundances have been observed in IBD patients [81]. At the same time, $O_1$ contains OTUs from the families Veillonellaceae and Coriobacteriaceae, for which increased abundances in IBD patients have been reported [81]. Finally, we analyzed set $O_3$ of OTUs present in the median for the HSCT responders but absent in the median for the non-responders. In line with the findings reported in [77], $O_3$ contains butyrate-producing bacteria from the Clostridiales order, as well as sulfate-reducing deltaproteobacteria.

### 7.4.2. Clustering

To evaluate the clustering performance, we clustered the 90% sub-collections of LETTER-EUCL, AIDS, and MUTA with $K$-medians clustering (cf. Section 6.2). We used $K$-means++ initialization and set the maximum number of iterations of Lloyd's algorithm to 10. For all datasets, we set $K$ to the number of different classes, i. e., $K = 2$ for MUTA and AIDS, and $K = 15$ for LETTER-EUCL. We recorded the runtime, the SOD, and the adjusted Rand index (ARI) w. r. t. the ground truth clustering. ARI measures the similarity between two clusterings [82]. Its range is $[-1, 1]$, and it evaluates to 1 if the two clusterings are identical. To have a baseline against which to compare, we also recorded these metrics for the very first clustering obtained by the $K$-means++ initialization.

Table 5 shows the mean outcomes of the experiments across the five test collections. On all datasets, $K$-medians clustering based on our BCU reduced the overall SOD by more than 30% w. r. t. the initial clustering obtained by $K$-means++ initialization. On LETTER-EUCL and AIDS, $K$-medians clustering significantly improved the ARI. This was not the case on MUTA, which can be explained by the fact that, on this dataset, tightness of bounds for GED is only very weakly correlated with clustering and classification performance [10]. Finally, the overall runtime was a couple of hours on LETTER-EUCL, a couple of minutes on MUTA, and a couple of seconds on AIDS. This is due to the facts that we computed 15 classes on LETTER-EUCL, but only 2 on AIDS and MUTA, and that computing the median labels is much more expensive than on AIDS and MUTA (cf. Section 7.2 above).

### 7.4.3. Classification and data reduction

Next, we tested how our algorithm performs when used for classification and data reduction as explained in Section 6.3. First, we split the benchmark datasets into disjoint collections of data and query graphs that we constrained to be balanced w. r. t. the classes. For each dataset and each class, we then computed 10 representatives of the data graphs via 10-medians clustering. Finally, we ran the following 1-NN query specifications (cf. Definition 6) for each query graph: (a) determine the nearest neighbor of each query graph among all data graphs; (b) determine the nearest neighbor from the set of representatives. Both specifications were set up to use the upper bound computed by BRANCH-FAST as the proxy $\widetilde{\mathrm{GED}}$ for GED.

Table 6 shows the mean classification coefficients (i. e., the proportion of correctly classified query graphs) and the mean query times. On all datasets, determining the nearest neighbor only among the set of representatives led to a speed-up of at least one order of magnitude. At the same time, the mean classification coefficient deteriorated only on MUTA, stayed unchanged on AIDS, and even slightly improved on LETTER-EUCL. Computing the representatives took a couple of minutes on AIDS and MUTA and a couple of hours on LETTER-EUCL.

### 7.4.4. Indexing

Finally, we tested how median based MBSTs as introduced in Section 6.4 perform at processing GED range queries in the sense of Definition 8. We again split the benchmark datasets into disjoint collections of data and query graphs. Subsequently, we computed median based MBST indices for the data graphs. Finally, for each query graph, we answered GED range queries for thresholds $\tau_p \in \{\tau_{0.01}, \tau_{0.025}, \tau_{0.05}, \tau_{0.075}, \tau_{0.1}\}$ (a) with the help of the pre-computed indices, and (b) via linear scans against all data graphs. To define the thresholds, we first estimated the GEDs between all query graphs as the means between the lower and the upper bounds returned by BRANCH-FAST. Subsequently, the estimated edit costs were sorted in increasing order, and $\tau_p$ was defined as the mean of the entries at positions $\lfloor p \cdot \mathit{length}(a) \rfloor$ and $\lceil p \cdot \mathit{length}(a) \rceil$ in the sorted array of estimated edit costs $a$. This construction mirrors the fact that, when issuing range queries, one is typically interested in retrieving objects that are close to the query object. All configurations of the range queries were set up to use the upper bound computed by IPFP and the lower bound returned by the linear programming based algorithm ADJ-LP suggested in [45] as, respectively, the upper and lower bounds $\lceil \mathrm{GED} \rceil$ and $\lfloor \mathrm{GED} \rfloor$ for GED (cf. Section 6.4.4). We selected these methods, because in [10] they are reported to produce very tight bounds for GED at affordable computational costs.

Fig. 6 shows the results of the indexing experiments. On all datasets, using median based MBSTs significantly reduced the mean query time. We also see that the runtime gain is especially high for small thresholds — that is, the thresholds one is typically interested in. This is not surprising, because more data graphs can be filtered for small thresholds. Building the index took a couple of minutes on AIDS and MUTA and a couple of hours on LETTER-EUCL.

## 7.5. Upshot of the experiments

The extensive experiments we reported in this section revealed that the proposed algorithm GMG-BCU clearly outperforms state of the art heuristics for computing generalized median graphs. The most important empirical findings are summarized in the following paragraphs.

### 7.5.1. Configuration of the proposed algorithm

*Initialization.* The best tradeoff between runtime and quality is achieved if GMG-BCU is initialized with a small number of randomly generated candidates.

*Termination criteria.* Running GMG-BCU with a maximal number of iterations is unnecessary, because GMG-BCU typically converges after a small number of iterations which is independent of the

**Table 5**
Results of clustering experiments.

| Dataset | GMG-BCU | | | $K$-means++ | | |
|---|---|---|---|---|---|---|
| | ARI | SOD | Runtime in s | ARI | SOD | Runtime in s |
| LETTER-EUCL (90 %) | 0.62 | 4437.8 | 11899.4 | 0.26 | 6388.8 | 2.9 |
| AIDS (90 %) | 0.43 | 47220.2 | 30.8 | 0.19 | 77540.0 | 0.5 |
| MUTA (90 %) | 0.00 | 224711.0 | 213.1 | 0.00 | 347107.8 | 1.7 |

**Table 6**
Results of classification experiments.

| Dataset | GMG-BCU representatives | | All data graphs | |
|---|---|---|---|---|
| | Classification coefficient | Query time in s | Classification coefficient | Query time in s |
| LETTER-EUCL (90 %) | 0.92 | 2.07e−3 | 0.90 | 2.54e−2 |
| AIDS (90 %) | 0.98 | 1.68e−3 | 0.98 | 5.26e−2 |
| MUTA (90 %) | 0.70 | 2.13e−3 | 0.80 | 5.45e−1 |

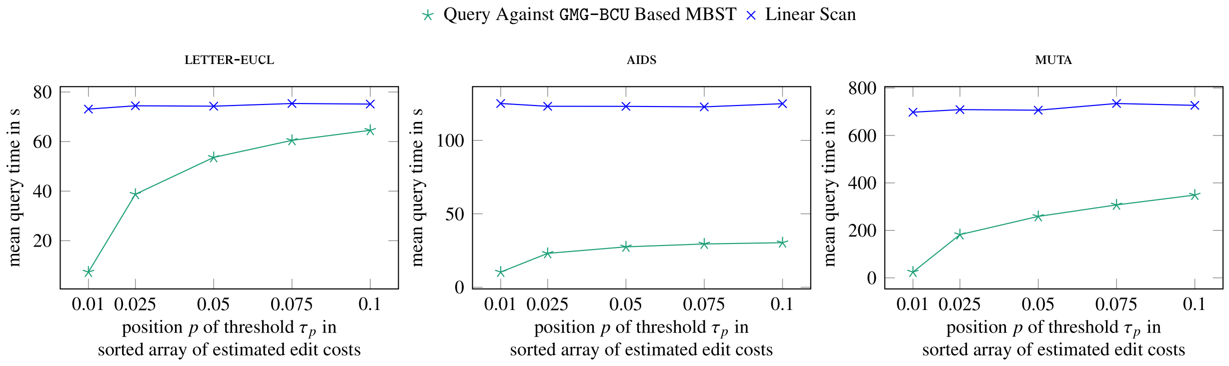

**Fig. 6.** Results of indexing experiments.

size of the graph collection $\mathcal{G}$. In contrast to that, the execution time of one iterations increases linearly with the size of $\mathcal{G}$. To control the maximal runtime of GMG-BCU, it is hence better to run it with a time limit. Since GMG-BCU quickly finds solutions of good quality, this time limit can safely be set to a small value, if necessary.

*Heuristic used during BCU.* The choice of the GED heuristic used during the BCU has only a small effect on the SOD but, on some datasets, a tremendous effect on the runtime. Therefore, it is recommendable to use GED heuristics that are optimized for speed rather than for tightness of the obtained upper bounds for GED.

*Final tightening.* If fast GED heuristics are used during the BCU, the final SOD can be improved by tightening the final node maps via more expensive GED heuristics. However, this configuration comes at the price of a significantly increased runtime and should hence only be used if runtime is not critical.

### *7.5.2. Comparison to state of the art*

*Runtime.* Whenever computing node maps between the input graphs is computationally more expensive than computing median labels, GMG-BCU can be configured to run orders of magnitudes faster than all competitors. State of the art approaches are faster only when used on relatively small datasets that contain very small graphs with continuous labels, but then compute looser SODs than our algorithm (cf. next paragraph).

*Quality.* If used for graphs with continuous labels, GMG-BCU computes tighter SODs than all evaluated competitors. If used for graphs with symbolic labels and constant edit costs, all obtained SODs are similar.

### *7.5.3. Performance in application scenarios*

*Computing representative graphs for differential microbiome data analysis.* GMG-BCU clearly computes the best representatives for all three sub-groups of the IBD dataset. The SODs of the representatives computed by state of the art heuristics are at least 10 % looser than the SODs of the representatives computed by GMG-BCU.

*Clustering.* On all test datasets except MUTA, $K$-medians clustering based on GMG-BCU yields a much better ARI than a randomized baseline that uses $K$-means++ initialization only. The poor performance on MUTA can be explained by the fact that, as reported in [10], the MUTA dataset seems to be poorly suited for GED based clustering techniques.

*Classification and data reduction.* On all test datasets, 1-NN can be sped-up by orders of magnitude if run against class-representatives obtained via GMG-BCU based $K$-medians clustering. At the same time, the classification coefficient drops only on MUTA and stays unchanged or even improves on the other test datasets.

*Indexing.* On all test datasets, GMG-BCU based MBSTs significantly speed up processing of GED range queries. This result is especially interesting, because, to the best of our knowledge, all existing indexing techniques that are designed to support GED range queries accept only uniform edit costs.

## 8. Conclusions and future work

In this paper, we presented a new algorithm GMG-BCU for estimating generalized median graphs. Unlike all existing approaches, GMG-BCU is designed to converge to a local optimum and is hence tightly linked to the definition of a generalized median graph. Moreover, it is generic, as it can be configured to use any GED heuristic and works for arbitrary node and edge labels as long as (exact or approximate) algorithms for computing median labels are available and the insertion and deletion costs are constant. Last but not least, GMG-BCU scales to large collections of graphs, because it can be configured to run in linear time w. r. t. the size of the graph collection and always returns a solution even if interrupted during the optimization.

We also proved that generalized median graphs are *NP*-hard to compute and *APX*-hard to approximate, that no polynomial $\alpha$-approximation algorithm exists unless the graph isomorphism problem is in *P*, that they exist if median labels exist and the insertion and deletion cost are constant, and that they are in general non-unique. Moreover, we provided proofs of concept of how to beneficially employ generalized median graphs for applications such as differential microbiome analysis, clustering, classification and data reduction, and indexing. Extensive experiments showed that our algorithm GMG-BCU clearly outperforms state of the art median graph estimators and also yields promising results when used for the aforementioned applications.

In future work, we will further explore how to beneficially employ GMG-BCU for differential microbiome data analysis. Here, the idea is to systematically compare the obtained generalized median graphs for the classes of the clinical variable of interest and hence individuate features that are predictive and/or causative for specific conditions. Moreover, we will build a compression tool for graph collections on top of GMG-BCU.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Data availability

The IBD data is available from BioProject under the accession code PRJNA565903: https://www.ncbi.nlm.nih.gov/bioproject/PRJNA565903/. All other data is available on GitHub: https://github.com/dbblumenthal/gedlib/.

## Acknowledgments

We would like to thank Markus List and Monica Matchado for pointing us to the IBD dataset. This work has been supported by *Région Normandie, France* under the project RIN AGAC, and by *Agence nationale de la recherche*, France under the project ESIGMA (ANR-17-CE23-0010). This work was supported by the Open Access Fund of the Free University of Bozen-Bolzano, Italy.

## Appendix A. Path based definition of the graph edit distance

Let $P_{\mathrm{ns}} \subset [n] \times [n']$, $P_{\mathrm{nd}} \subset [n]$, and $P_{\mathrm{ni}} \subset [n']$ be the sets of, respectively, node substitutions, deletions, and insertions contained in an edit path $P$ from a graph $G$ of order $n$ to a graph $G'$ of order $n'$. Similarly, let $P_{\mathrm{es}} \subset ([n] \times [n]) \times ([n'] \times [n'])$, $P_{\mathrm{ed}} \subset [n] \times [n]$, and $P_{\mathrm{ei}} \subset [n'] \times [n']$ be the sets of, respectively, edge substitutions, deletions, and insertions. Then the cost of the edit path $P$ is defined as

$$c(P) := \underbrace{\sum_{(i,k)\in P_{\mathrm{ns}}} c_{\mathrm{ns}}(\varphi_i, \varphi'_k)}_{\text{node substitutions}} + \underbrace{\sum_{((i,j),(k,l))\in P_{\mathrm{es}}} c_{\mathrm{es}}(\Phi_{i,j}, \Phi'_{k,l})}_{\text{edge substitutions}}$$
$$+ \underbrace{\sum_{i\in P_{\mathrm{nd}}} c_{\mathrm{nd}}(\varphi_i) + \sum_{(i,j)\in P_{\mathrm{ed}}} c_{\mathrm{ed}}(\Phi_{i,j})}_{\text{node and edge deletions}}$$
$$+ \underbrace{\sum_{k\in P_{\mathrm{ni}}} c_{\mathrm{ni}}(\varphi'_i) + \sum_{(k,l)\in P_{\mathrm{ei}}} c_{\mathrm{ei}}(\Phi'_{i,j})}_{\text{node and edge insertions}},$$

which leads to the following definition of GED:

**Definition 10** (GED — *Path Based Definition*)**.** The *graph edit distance* from a graph $G$ to a graph $G'$ is defined as $\mathrm{GED}(G, G') := \min_{P\in\Psi_{G,G'}} c(P)$, where $\Psi_{G,G'}$ is the set of all edit paths from $G$ to $G'$. Recall that an edit path $P$ from $G$ to $G'$ is a sequence of edit operations transforming $G$ into a graph isomorphic to $G'$, and that its cost $c(P)$ is defined as the sum of the costs of the contained edit operations.

## Appendix B. Proof of Theorem 1

The proof starts with Lemma 1, which shows that if (C1) holds, then for any graph with a sufficiently large order, there is always a graph of smaller order with a smaller SOD.

**Lemma 1.** *Let $\mathcal{P}_{\mathrm{fin}}(\mathbb{G})$ be the set of all finite multisets of $\mathbb{G}$ and assume that (C1) holds. Then there is a function*

$$N : \mathcal{P}_{\mathrm{fin}}(\mathbb{G}) \times \mathbb{R}^4_{\geq 0} \to \mathbb{N}$$

*such that for all $(\mathcal{G}, (c_{\mathrm{nd}}, c_{\mathrm{ni}}, c_{\mathrm{ed}}, c_{\mathrm{ei}})) \in \mathcal{P}_{\mathrm{fin}}(\mathbb{G}) \times \mathbb{R}^4_{\geq 0}$ and all graph $G \in \mathbb{G}$ of order $n > N(\mathcal{G}, (c_{\mathrm{nd}}, c_{\mathrm{ni}}, c_{\mathrm{ed}}, c_{\mathrm{ei}}))$ there is a graph $G' \in \mathbb{G}$ of order $n' \leq N(\mathcal{G}, (c_{\mathrm{nd}}, c_{\mathrm{ni}}, c_{\mathrm{ed}}, c_{\mathrm{ei}}))$ with $\mathrm{SOD}(G', \mathcal{G}) \leq \mathrm{SOD}(G, \mathcal{G})$.*

**Proof.** We construct a function $N$ that meets the requirements of the lemma. Let $(\mathcal{G}, C) \in \mathcal{P}_{\mathrm{fin}}(\mathbb{G}) \times \mathbb{R}^4_{\geq 0}$, with $C = (c_{\mathrm{nd}}, c_{\mathrm{ni}}, c_{\mathrm{ed}}, c_{\mathrm{ei}})$. We distinguish the following three cases:

Case (1) $c_{\mathrm{nd}} = c_{\mathrm{ed}} = 0$
Case (2) $c_{\mathrm{nd}} > 0$
Case (3) $c_{\mathrm{nd}} = 0 \wedge c_{\mathrm{ed}} > 0$

Case (1) Let $G'$ be the disjoint union of all the graphs contained in $\mathcal{G}$. Since nodes and edges can be deleted for free, we have $\mathrm{SOD}(G', \mathcal{G}) = 0$. So we can define

$$N(\mathcal{G}, C) := \sum_{p\in[|\mathcal{G}|]} n^p.$$

Case (2) Let $G$ be any graph of order $n > \max_p n^p$. To transform $G$ into each $G^p$, at least $n - n^p > 0$ nodes must be deleted with non-zero cost. So its SOD is bounded from below by

$$\mathrm{SOD}(G, \mathcal{G}) \geq \sum_{p\in[|\mathcal{G}|]} c_{\mathrm{nd}}(n - n^p) = c_{\mathrm{nd}}(n|\mathcal{G}| - \sum_{p\in[|\mathcal{G}|]} n^p).$$

This is an increasing function in the order $n$ of $G$. If $n$ is taken large enough, e. g., such that $c_{\mathrm{nd}}(n|\mathcal{G}| - \sum_p n^p) \geq UB(\mathcal{G}, (c_{\mathrm{ni}}, c_{\mathrm{ei}}))$, where $UB$ is defined as

$$\begin{aligned} UB(\mathcal{G}, (c_{\mathrm{ni}}, c_{\mathrm{ei}})) &:= \mathrm{SOD}(G_\emptyset, \mathcal{G}) \\ &= c_{\mathrm{ni}} \sum_{p\in[|\mathcal{G}|]} n^p + c_{\mathrm{ei}} \sum_{p\in[|\mathcal{G}|]} e^p \end{aligned}$$

and $e^p$ denotes the number of edges contained in $G^p \in \mathcal{G}$, then the SOD is greater for $G$ than for the empty graph $G_\emptyset$. So we can define

$$N(\mathcal{G}, C) := \left\lceil \frac{1}{|\mathcal{G}|} \left( \frac{1}{c_{\mathrm{nd}}} UB(\mathcal{G}, (c_{\mathrm{ni}}, c_{\mathrm{ei}})) + \sum_{p=1}^{|\mathcal{G}|} n^p \right) \right\rceil.$$

Case (3) By using analogous arguments to the ones employed in Case (2), we can show that if a graph $G$ contains more than

$$N'(\mathcal{G}, C) := \max\left\{ \binom{n^\star}{2}, \left\lceil \frac{2}{|\mathcal{G}|} \left( \frac{UB(\mathcal{G}, (c_{\mathrm{ni}}, c_{\mathrm{ei}}))}{c_{\mathrm{ed}}} + \sum_{p\in[|\mathcal{G}|]} \binom{n^p}{2} \right) \right\rceil \right\}$$

non-isolated nodes, where $n^\star := \max_{p\in[|\mathcal{G}|]} n^p$, then $\mathrm{SOD}(G_\emptyset, \mathcal{G}) < \mathrm{SOD}(G, \mathcal{G})$ holds. Now we define

$$N(\mathcal{G}, C) := N'(\mathcal{G}, C) + \sum_{p\in[|\mathcal{G}|]} n^p.$$

Let $G \in \mathbb{G}$ be a graph of order larger than $N(\mathcal{G}, C)$, and $V_{\mathrm{i}}$ and $V_{\text{n-i}}$ be the sets of $G$'s isolated and non-isolated nodes, respectively. If $|V_{\text{n-i}}| > N'(\mathcal{G}, C)$, we are done. Otherwise, we construct a graph $G'$ as follows: $G'$ contains $G$'s induced subgraph on $V_{\text{n-i}}$. Moreover, for each node $i$ in each graph $G^p \in \mathcal{G}$, $G'$ contains an isolated node with label $\varphi^p_i$. It is easy to see that $\mathrm{GED}(G', G^p) \leq \mathrm{GED}(G, G^p)$ holds for each $G^p \in \mathcal{G}$. Moreover, by construction, $G'$'s order does not exceed $N(\mathcal{G}, C)$, and we are done. □

Since the bound $N$ for the order does not depend on substitution cost, the search space for the order variable $n$ in Corollary 1 can be reduced to $[N]$. Also note that, since the edge labels are independent of the node labels, the third minimization in Corollary 1 can be decomposed into two independent minimizations:

$$\mathrm{SOD}(G^\star, \mathcal{G}) = \min_{n\in\mathbb{N}} \min_{\pi\in\Pi^{\mathcal{G}}_n} \Bigg[ \underbrace{\min_{\varphi\in\mathcal{L}^n_V} \sum_{p\in[|\mathcal{G}|]} c_V(\pi^p, \varphi, \varphi^p)}_{(\text{E1})} + \underbrace{\min_{(A,\Phi)\in\mathcal{A}_n\times\mathcal{L}^{n\times n}_E} \sum_{p\in[|\mathcal{G}|]} c_E(\pi^p, A, \Phi, A^p, \Phi^p)}_{(\text{E2})} \Bigg]$$

The following Lemma 2 shows that, if condition (C1) holds, then condition (C2) ensures that the subproblems (E1) and (E2) have a solution. In other word: the median graph exists if median labels for the substituted nodes and edges exist.

**Lemma 2.** *Assume that (C2) holds. Then the following statements hold:*

- *A vector of node labels $\varphi^\star \in \mathcal{L}^n_V$ solves subproblem (E1), if $\varphi^\star_i = \arg\min_{x\in\mathcal{L}_V} \sum_{y\in\mathcal{L}_i} c_{\mathrm{ns}}(x, y)$ holds for all $i \in [n]$, where $\mathcal{L}_i := \{\varphi^p_{\pi^p(i)} \mid p \in [|\mathcal{G}|] \wedge \pi^p(i) \neq \epsilon\}$ is the multiset of labels of $i$'s substituted nodes.*
- *A pair $(A^\star, \Phi^\star) \in \mathcal{A}_n \times \mathcal{L}^{n\times n}_E$ solves subproblem (E2), if $\Phi^\star_{i,j} = \arg\min_{y\in\mathcal{L}_E} \sum_{z\in\mathcal{L}_{i,j}} c_{\mathrm{es}}(y, z)$ and $A^\star_{i,j} = (\sum_{z\in\mathcal{L}_{i,j}} c_{\mathrm{es}}(\Phi^\star_{i,j}, z) < (c_{\mathrm{ei}} + c_{\mathrm{ed}})|\mathcal{L}_{i,j}| - c_{\mathrm{ed}}|\mathcal{G}|)$ hold for all $(i, j) \in [n] \times [n]$, where $\mathcal{L}_{i,j} := \{\Phi^p_{\pi^p(i),\pi^p(j)} \mid \pi^p(i) \neq \epsilon \wedge \pi^p(j) \neq \epsilon \wedge A^p_{\pi^p(i),\pi^p(j)} = 1\}$ is the multiset of labels of $(i, j)$'s substituted edges.*

**Proof.** By removing the constant terms in the expression of $c_V$, we obtain that the subproblem (E1) is equivalent to the following minimization problem:

$$\arg\min_{\varphi\in\mathcal{L}^n_V} \sum_{i\in[n]} \sum_{p\in[|\mathcal{G}|]} \delta_{\pi^p(i)} c_{\mathrm{ns}}(\varphi_i, \varphi^p_{\pi^p(i)})$$

Since the sum over $p$ is positive and independent for each $i$, this problem is equivalent to

$$\sum_{i\in[n]} \arg\min_{x\in\mathcal{L}_V} \sum_{y\in\mathcal{L}_i} c_{\mathrm{ns}}(x, y),$$

which implies the first part of the lemma by definition of $\mathcal{L}_i$.

For showing the second part of the lemma, note that, by removing the constant terms in $c_E$, we obtain that the subproblem (E2) is equivalent to the minimization problem

$$\arg\min_{(A,\Phi)\in\mathcal{A}_n\times\mathcal{L}^{n\times n}_E} \sum_{i\in[n]} \sum_{j\in[n]} f_{i,j}(A_{i,j}, \Phi_{i,j}), \tag{B.1}$$

where the functions $f_{i,j} : \{0, 1\}\times\mathcal{L}_E \to \mathbb{R}_{\geq 0}$ are defined as follows:

$$\begin{aligned} f_{i,j}(x, y) := {} & x \sum_{p\in[|\mathcal{G}|]} \delta_{\pi^p(i)}\delta_{\pi^p(j)} A^p_{\pi^p(i),\pi^p(j)} c_{\mathrm{es}}(y, \Phi^p_{\pi^p(i),\pi^p(j)}) \\ & + x \cdot c_{\mathrm{ed}} \sum_{p\in[|\mathcal{G}|]} 1 - \delta_{\pi^p(i)}\delta_{\pi^p(j)} A^p_{\pi^p(i),\pi^p(j)} \\ & + (1 - x) \cdot c_{\mathrm{ei}} \sum_{p\in[|\mathcal{G}|]} \delta_{\pi^p(i)}\delta_{\pi^p(j)} A^p_{\pi^p(i),\pi^p(j)} \\ = {} & x \sum_{z\in\mathcal{L}_{i,j}} c_{\mathrm{es}}(y, z) + x \left(|\mathcal{G}| - |\mathcal{L}_{i,j}|\right) c_{\mathrm{ed}} \\ & + (1 - x)|\mathcal{L}_{i,j}| c_{\mathrm{ei}} \end{aligned}$$

For all pairs $(i, j) \in [n] \times [n]$, these functions are positive and independent from each others. Therefore, the optimization problem given in Eq. (B.1) is equivalent to the problem

$$\sum_{i\in[n]} \sum_{j\in[n]} \arg\min_{(x,y)\in\{0,1\}\times\mathcal{L}_E} f_{i,j}(x, y).$$

The variable $x$ can only take two values. If $x = 0$ (no edge), we have $f_{i,j}(0, y) = c_{\mathrm{ei}}|\mathcal{L}_{i,j}|$ for each $y \in \mathcal{L}_E$. If $x = 1$, then $f_{i,j}(1, y)$ is minimized by any

$$y^\star := \arg\min_{y\in\mathcal{L}_E} \sum_{z\in\mathcal{L}_{i,j}} c_{\mathrm{es}}(y, z)$$

By definition of $\mathcal{L}_{i,j}$, this yields the second part of the lemma. □

We are now in the position to prove Theorem 1.

**Proof of Theorem 1.** Using (C1) and Lemma 1, the order $n$ of the median graph is bounded by $N(\mathcal{G}, (c_{\mathrm{nd}}, c_{\mathrm{ni}}, c_{\mathrm{ed}}, c_{\mathrm{ei}}))$. Moreover, it has been shown in [83] that $|\Pi_{n,n'}| \leq (n + n')!$ holds for all $n, n' \in \mathbb{N}$, which implies that $\Pi^{\mathcal{G}}_n$ is finite for each order $n$. Therefore, (C2) and Lemma 2 imply that the optimal median graph $(A, \varphi, \Phi)$ can be determined by enumerating all $n \in [N]$ and all $\boldsymbol{\pi} \in \Pi^{\mathcal{G}}_n$. This concludes the proof. □

## Appendix C. Proof of Theorem 2

Our proof uses a polynomial reduction from the *maximum common edge subgraph problem* (MCES). Given two unlabeled, undirected graphs $G_1$ and $G_2$ of equal order, MCES asks to compute a common subgraph $\overline{G}$ of $G_1$ and $G_2$ with a maximum number of edges. MCES is known to be *NP*-hard [84]. To facilitate notations, throughout this section, we use the notation $|G|$ to denote the number of edges of a graph $G$. We start the proof with two lemmata.

**Lemma 3.** *Let $\mathbb{G}$ be the set of all undirected, unlabeled graphs. If the edit costs are uniform, i. e., if $c_{\mathrm{nd}} = c_{\mathrm{ni}} = c_{\mathrm{ed}} = c_{\mathrm{ei}} = 1$, GED is a metric on $\mathbb{G}$.*

**Proof.** The lemma immediately follows from the definition of GED. A formal proof can be found in [45]. □

**Lemma 4.** *Let $G_1$ and $G_2$ be unlabeled graphs of the same order. If the edit costs are uniform, it holds that $\mathrm{GED}(G_1, G_2) = |G_1| + |G_2| - 2|\overline{G}|$, where $\overline{G}$ is a maximum common edge subgraph for $G_1$ and $G_2$.*

**Proof.** Since the graphs $G_1$ and $G_2$ have the same order, any optimal edit path between them performs edit operations on the edge set only. This implies the following simple observations:

- Given any edit path $P$ from $G_1$ to $G_2$, the graph $G(P)$ obtained by applying only the edge deletion operation of $P$ is a common edge subgraph of $G_1$ and $G_2$.

- For each common edge subgraph $G$ of $G_1$ and $G_2$, the edit path $P(G)$ from $G_1$ to $G_2$ obtained by first deleting all edges of $G_1 \setminus G$ and then inserting all edges of $G_2 \setminus G$ is a valid edit path, and its cost is $|G_1| + |G_2| - 2|G|$.

Combining these observations implies that there is a bijective relation between the set of common edge subgraphs and the set of unordered edit paths. Finding a common edge subgraph with maximal size hence amounts to finding an edit path with minimal cost and vice versa. This proves the lemma. □

We are now ready to prove Theorem 2.

**Proof of Theorem 2.** Let $(G_1, G_2)$ be an instance of MCES and $n$ be the order of $G_1$ and $G_2$. We construct an instance $(\mathbb{G}, \mathcal{G}, c_{\text{nd}}, c_{\text{ni}}, c_{\text{ed}}, c_{\text{ei}})$ of the generalized median graph problem as follows:

- $\mathbb{G}$ is the set of all unlabeled undirected graphs.
- The collection $\mathcal{G}$ is defined as $\mathcal{G} := \{G_1, G_2, G_3\}$, where $G_3$ is a graph with $n$ isolated unlabeled nodes.
- The edit costs are uniform.

Let $\overline{G}$ be a maximum common edge subgraph of $G_1$ and $G_2$ and $G^\star$ be a generalized median graph for $(\mathbb{G}, \mathcal{G}, c_{\text{nd}}, c_{\text{ni}}, c_{\text{ed}}, c_{\text{ei}})$. We have to show that $\overline{G}$ is a generalized median graph for $(\mathbb{G}, \mathcal{G}, c_{\text{nd}}, c_{\text{ni}}, c_{\text{ed}}, c_{\text{ei}})$ and that $G^\star$ is a maximum common edge subgraph of $G_1$ and $G_2$. This is the case if and only if the following claims are true:

Claim (1) It holds that $\text{SOD}(G^\star) = \text{SOD}(\overline{G})$.
Claim (2) $G^\star$ is a common subgraph of $G_1$ and $G_2$ and it holds that $|G^\star| = |\overline{G}|$.

We first prove Claim (1). Since $\text{SOD}(G^\star) \le \text{SOD}(\overline{G})$ holds by choice of $G^\star$, it suffices to show the inequality $\text{SOD}(G^\star) \ge \text{SOD}(\overline{G})$. By definition of SOD, we have

$$\begin{aligned}\text{SOD}(G^\star) &= \text{GED}(G^\star, G_1) + \text{GED}(G^\star, G_2) + \text{GED}(G^\star, G_3)\\ &\ge \big(\ \text{GED}(G_1, G_2) + \text{GED}(G_1, G_3)\\ &\quad +\ \text{GED}(G_2, G_3)\ \big)\ /2,\end{aligned}$$

where the inequality follows from Lemma 3. By noting that $\text{GED}(G_1, G_3)$ and $\text{GED}(G_2, G_3)$ are equal to $|G_1|$ and $|G_2|$, respectively, and applying Lemma 4 to the inequality above, we obtain the following lower bound for the $\text{SOD}(G^\star)$:

$$\text{SOD}(G^\star) \ge |G_1| + |G_2| - |\overline{G}| \tag{C.1}$$

On the other hand, the SOD of a maximum common subgraph $\overline{G}$ amounts to

$$\begin{aligned}\text{SOD}(\overline{G}) &= \text{GED}(\overline{G}, G_1) + \text{GED}(\overline{G}, G_2) + \text{GED}(\overline{G}, G_3)\\ &= |G_1 \setminus \overline{G}| + |G_2 \setminus \overline{G}| + |\overline{G}| = |G_1| + |G_2| - |\overline{G}|,\end{aligned}$$

which exactly matches the lower bound on the optimal SOD given by Eq. (C.1). We hence have $\text{SOD}(\overline{G}) \le \text{SOD}(G^\star)$, as required.

Next, we prove Claim (2), i.e., show that $G^\star$ is a common subgraph of $G_1$ and $G_2$ with $|G^\star| = |\overline{G}|$. We first bound the cardinality of $G^\star$ by

$$\begin{aligned}|G^\star| &= \text{GED}(G^\star, G_3)\\ &= \text{SOD}(G^\star) - \big(\text{GED}(G^\star, G_1) + \text{GED}(G^\star, G_2)\big)\\ &\le \text{SOD}(\overline{G}) - \text{GED}(G_1, G_2)\\ &= |G_1| + |G_2| - |\overline{G}| - \big(|G_1| + |G_2| - 2|\overline{G}|\big) = |\overline{G}|,\end{aligned} \tag{C.2}$$

where the inequality again follows from Lemma 3.

Let $\overline{G}_1$ be a maximum common edge subgraph of $G_1$ and $G^\star$, and $\overline{G}_2$ be a maximum common edge subgraph of $G_2$ and $G^\star$. Since $\text{GED}(G^\star, \overline{G}_1) = |G^\star| + |G_1| - 2|\overline{G}_1|$, and $\text{GED}(G^\star, G_2) = |G^\star| + |G_2| - 2|\overline{G}_2|$, we have

$$\begin{aligned}\text{SOD}(G^\star) &= |G_1| + |G_2| + 3|G^*| - 2(|\overline{G}_1| + |\overline{G}_2|)\\ &\ge |G_1| + |G_2| - |\overline{G}| + 2\left(2|G^\star| - (|\overline{G}_1| + |\overline{G}_2|)\right)\\ &\ge \text{SOD}(\overline{G}) + 2\left(2|G^\star| - (|\overline{G}_1| + |\overline{G}_2|)\right),\end{aligned}$$

which, by plugging in the equation $\text{SOD}(\bar{G}) = \text{SOD}(G^\star)$, eventually leads to

$$2|G^\star| \le |\overline{G}_1| + |\overline{G}_2|. \tag{C.3}$$

Since both $\overline{G}_1$ and $\overline{G}_2$ are subgraphs of $G^\star$, we have $|\overline{G}_1| \le |G^\star|$ and $|\overline{G}_2| \le |G^\star|$. By combining these upper bounds on $|\overline{G}_1|$ and $|\overline{G}_2|$ with Eq. (C.3), we obtain $|G^\star| = |\overline{G}_1| = |\overline{G}_2|$ and hence $G^\star = \overline{G}_1 = \overline{G}_2$. Therefore, $G^\star$ is a subgraph of both $G_1$ and $G_2$, which implies $|G^\star| \le |\overline{G}|$ by choice of $\overline{G}$. Together with Eq. (C.2), this concludes the proof. □

## Appendix D. Proofs of propositions

**Proof of Proposition 1.** Let $\mathcal{L}_V = \mathbb{R}\setminus\{0\}$ be the set of node labels, $\mathcal{L}_E = \{1\}$ be the set of edge labels, $c_{\text{ns}}(x, y) := (x-y)^2$ be the node substitution costs, all other edit costs be equal to a constant $c > 2$, and $\mathcal{G} := \{G_x, G_y\}$ be the collection of graphs, where $G_x$ and $G_y$ each contain just one isolated node with labels $x = -1$ and $y = 1$, respectively. Assume that there is a generalized median graph $G^\star$. Then $G^\star$ contains just one isolated node, because otherwise we have $\text{SOD}(G^\star, \mathcal{G}) \ge 2 \cdot c > 4 = \text{SOD}(G_x, \mathcal{G})$. Let $z \in \mathcal{L}_V$ be the label of $G^\star$'s unique node. Since $0 \notin \mathcal{L}_V$, we know that $z \ne 0$. Let $G'$ be a graph with just one isolated node with label $z/2$. Then easy algebra shows that we have $\text{SOD}(G', \mathcal{G}) < \text{SOD}(G^\star, \mathcal{G})$. This contradicts the assumption that $G^\star$ is a generalized median graph. □

**Proof of Proposition 2.** Consider any order $n$ and any vector of pairwise different node labels $\varphi = (\varphi_i)_{i=1}^n$. Now consider the set $\mathcal{G}_{n,\varphi} := \{(A, \varphi, 1_{n\times n}) \mid A \in \mathcal{A}_n\}$ of graphs with unlabeled edges generated from $\varphi$. Since $\varphi$ is fixed and contains different labels, $\mathcal{G}_{n,\varphi}$ contains exactly $2^{(n^2-n)/2} = |\mathcal{A}_n|$ graphs. Furthermore, assume that, for all $x \ne y \in \varphi$, we have $c_{\text{ns}}(x, y) = c_{\text{nd}} = c_{\text{ni}} \gg c_{\text{ed}} = c_{\text{ei}} = c > 0$.

First, notice that any graph outside of $\mathcal{G}_{n,\varphi}$ has labeled nodes different from $\varphi$, and so its SOD to $\mathcal{G}_{n,\varphi}$ is very large (as node substitution, deletion and insertion have a very high cost). Then, remark that the SOD of any graph in $\mathcal{G}_{n,\varphi}$ always equals to $c(n^2 - n) \cdot 2^{((n^2-n)/2)-2}$. Indeed, for each pair $(G, G')$ of graphs of $\mathcal{G}_{n,\varphi}$, there is a unique node map that has a non-infinite cost, the one that maps each node label in $G$ to the same label in $G'$ (no node insertion nor deletion), *i.e.* the identity. Thus, for any graph in $\mathcal{G}_{n,\varphi}$, the contribution of the nodes to the sum of distances amounts to $0$ ($c_V = 0$ from $c_{\text{ns}} = 0$).

Regarding edges, note that for any pair of nodes, $\mathcal{G}_{n,\varphi}$ has $2^{((n^2-n)/2)-1}$ graphs where there is an edge between these two nodes, or $2^{((n^2-n)/2)-1}$ graphs where there is none. Hence, for any pair of nodes, its contribution to sum of distances amounts exactly to $c \cdot 2^{((n^2-n)/2)-1}$, whether this pair corresponds to an edge of $G$ or not. If this pair corresponds to an edge, then $2^{(n^2-n/2)-1}$ of the node maps that count in the sum of distances delete this edge, if on the other hand this pair does not correspond to an edge, then $2^{(n^2-n/2)-1}$ of the node maps insert an edge. □

**Proof of Proposition 3.** The proposition follows from the respective results for GED demonstrated in [85] and the fact that, as long as the edit costs are metric, the equation $\text{SOD}^\star :=$

$\min_{G\in\mathbb{G}} \mathrm{SOD}(G, \{G_1, G_2\}) = \mathrm{GED}(G_1, G_2)$ holds for all graphs $G_1, G_2 \in \mathbb{G}$. To show this equation, first note that we have $\mathrm{SOD}^\star \leq \mathrm{SOD}(G_1, \{G_1, G_2\}) = \mathrm{GED}(G_1, G_1) + \mathrm{GED}(G_1, G_2) = \mathrm{GED}(G_1, G_2)$. Assume now that $\mathrm{SOD}^\star = \mathrm{SOD}(G^\star, \{G_1, G_2\}) < \mathrm{GED}(G_1, G_2)$, and let $P_{\star\to 1}$ and $P_{\star\to 2}$ be optimal edit paths from $G^\star$ to $G_1$ and from $G^\star$ to $G_2$, respectively. Since the edit costs are metric, inverting the edit operations along $P_{\star\to 1}$ yields an edit path $P_{1\to\star}$ from $G_1$ to $G^\star$ with cost $c(P_{1\to\star}) = c(P_{\star\to 1})$. This implies the contradiction $\mathrm{GED}(G_1, G_2) \leq c(P_{1\to\star} \circ P_{\star\to 2}) = c(P_{1\to\star}) + c(P_{\star\to 2}) = \mathrm{SOD}^\star < \mathrm{GED}(G_1, G_2)$. □

**Proof of Proposition 4.** The proposition follows from the fact the, in the each iteration, the SOD is improved by at least $\varepsilon$. □

**Proof of Proposition 5.** The proposition follows from the fact that, in each iteration of BCU, we have to call $\lceil\mathrm{GED}\rceil$ exactly $|\mathcal{G}|$ times to compute upper bounds between the current median and all the graphs contained $\mathcal{G}$. □

**Proof of Proposition 6.** The proposition follows from the definition of a pseudometric space. □

**Proof of Proposition 7.** The proposition follows from (C3) and the path-based Definition 10 of GED. □

## References

[1] E. Ozdemir, C. Gunduz-Demir, A hybrid classification model for digital pathology using structural and statistical pattern recognition, IEEE Trans. Med. Imag. 32 (2) (2013) 474–483, http://dx.doi.org/10.1109/TMI.2012.2230186.

[2] M. Stauffer, A. Fischer, K. Riesen, A novel graph database for handwritten word images, in: A. Robles-Kelly, M. Loog, B. Biggio, F. Escolano, R. Wilson (Eds.), S+SSPR 2016, in: LNCS, vol. 10029, Springer, Cham, 2016, pp. 553–563, http://dx.doi.org/10.1007/978-3-319-49055-7_49.

[3] M. Stauffer, T. Tschachtli, A. Fischer, K. Riesen, A survey on applications of bipartite graph edit distance, in: P. Foggia, C. Liu, M. Vento (Eds.), GbRPR 2017, in: LNCS, vol. 10310, Springer, Cham, 2017, pp. 242–252, http://dx.doi.org/10.1007/978-3-319-58961-9_22.

[4] H. Bunke, G. Allermann, Inexact graph matching for structural pattern recognition, Pattern Recognit. Lett. 1 (4) (1983) 245–253, http://dx.doi.org/10.1016/0167-8655(83)90033-8.

[5] K. Riesen, Structural Pattern Recognition with Graph Edit Distance, in: Advances in Computer Vision and Pattern Recognition, Springer, Cham, 2015, http://dx.doi.org/10.1007/978-3-319-27252-8.

[6] Z. Zeng, A.K.H. Tung, J. Wang, J. Feng, L. Zhou, Comparing stars: On approximating graph edit distance, Proc. VLDB Endow. 2 (1) (2009) 25–36, http://dx.doi.org/10.14778/1687627.1687631.

[7] D.B. Blumenthal, J. Gamper, On the exact computation of the graph edit distance, Pattern Recognit. Lett. 134 (2020) 46–57, http://dx.doi.org/10.1016/j.patrec.2018.05.002.

[8] D.B. Blumenthal, J. Gamper, Improved lower bounds for graph edit distance, IEEE Trans. Knowl. Data Eng. 30 (3) (2018) 503–516, http://dx.doi.org/10.1109/TKDE.2017.2772243.

[9] N. Boria, D.B. Blumenthal, S. Bougleux, L. Brun, Improved local search for graph edit distance, Pattern Recognit. Lett. 129 (2020) 19–25, http://dx.doi.org/10.1016/j.patrec.2019.10.028.

[10] D.B. Blumenthal, N. Boria, J. Gamper, S. Bougleux, L. Brun, Comparing heuristics for graph edit distance computation, VLDB J. 29 (1) (2020) 419–458, http://dx.doi.org/10.1007/s00778-019-00544-1.

[11] Y. Li, C. Gu, T. Dullien, O. Vinyals, P. Kohli, Graph matching networks for learning the similarity of graph structured objects, in: K. Chaudhuri, R. Salakhutdinov (Eds.), ICML 2019, in: Proceedings of Machine Learning Research, vol. 97, PMLR, Long Beach, 2019, pp. 3835–3845, URL http://proceedings.mlr.press/v97/li19d.html.

[12] Y. Bai, H. Ding, S. Bian, T. Chen, Y. Sun, W. Wang, SimGNN: A neural network approach to fast graph similarity computation, in: J.S. Culpepper, A. Moffat, P.N. Bennett, K. Lerman (Eds.), WSDM 2019, ACM, New York, 2019, pp. 384–392, http://dx.doi.org/10.1145/3289600.3290967.

[13] D.B. Blumenthal, J. Gamper, S. Bougleux, L. Brun, Upper bounding the graph edit distance based on rings and machine learning, Int. J. Pattern Recognit. Artif. Intell. (2021) http://dx.doi.org/10.1142/S0218001421510083, in press.

[14] X. Jiang, A. Munger, H. Bunke, On median graphs: properties, algorithms, and applications, IEEE Trans. Pattern Anal. Mach. Intell. 23 (10) (2001) 1144–1151, http://dx.doi.org/10.1109/34.954604.

[15] C. de la Higuera, F. Casacuberta, Topology of strings: Median string is NP-complete, Theoret. Comput. Sci. 230 (1–2) (2000) 39–48, http://dx.doi.org/10.1016/S0304-3975(97)00240-5.

[16] F. Nicolas, E. Rivals, Hardness results for the center and median string problems under the weighted and unweighted edit distances, J. Discrete Algorithms 3 (2005) 390–415, http://dx.doi.org/10.1016/j.jda.2004.08.015.

[17] A. Münger, Synthesis of Prototype Graphs from Sample Graphs (Master's thesis), University of Bern, 1998 (in German).

[18] M. Ferrer, Theory and Algorithms on the Median Graph. Application to Graph-Based Classification and Clustering (Ph.D. thesis), Universitat Autònoma de Barcelona, 2008, URL http://hdl.handle.net/10803/5788.

[19] M. Ferrer, E. Valveny, F. Serratosa, Median graph: A new exact algorithm using a distance based on the maximum common subgraph, Pattern Recognit. Lett. 30 (5) (2009) 579–588, http://dx.doi.org/10.1016/j.patrec.2008.12.014.

[20] M. Ferrer, E. Valveny, F. Serratosa, Median graphs: A genetic approach based on new theoretical properties, Pattern Recognit. 42 (9) (2009) 2003–2012, http://dx.doi.org/10.1016/j.patcog.2009.01.034.

[21] A. Hlaoui, S. Wang, Median graph computation for graph clustering, Soft Comput. 10 (1) (2006) 47–53, http://dx.doi.org/10.1007/s00500-005-0464-1.

[22] L. Musmanno, C.C. Ribeiro, Heuristics for the generalized median graph problem, European J. Oper. Res. 254 (2) (2016) 371–384, http://dx.doi.org/10.1016/j.ejor.2016.03.048.

[23] L. Mukherjee, V. Singh, J. Peng, J. Xu, M.J. Zeitz, R. Berezney, Generalized median graphs and applications, J. Comb. Optim. 17 (1) (2009) 21–44, http://dx.doi.org/10.1007/s10878-008-9184-7.

[24] M. Ferrer, F. Serratosa, A. Sanfeliu, Synthesis of median spectral graph, in: J.S. Marques, N.P. de la Blanca, P. Pina (Eds.), IbPRIA 2005, in: LNCS, vol. 3523, Springer, Berlin, Heidelberg, 2005, pp. 139–146, http://dx.doi.org/10.1007/11492542_18.

[25] D. White, R.C. Wilson, Mixing spectral representations of graphs, in: ICPR 2006, IEEE Computer Society, Los Alamitos, 2006, pp. 140–144, http://dx.doi.org/10.1109/ICPR.2006.803.

[26] S. Umeyama, An eigendecomposition approach to weighted graph matching problems, IEEE Trans. Pattern Anal. Mach. Intell. 10 (5) (1988) 695–703, http://dx.doi.org/10.1109/34.6778.

[27] P. Goyal, E. Ferrara, Graph embedding techniques, applications, and performance: A survey, Knowl.-Based Syst. 151 (2018) 78–94, http://dx.doi.org/10.1016/j.knosys.2018.03.022.

[28] K. Riesen, H. Bunke, Graph Classification and Clustering Based on Vector Space Embedding, in: Series in Machine Perception and Artificial Intelligence, vol. 77, World Scientific, Singapore, 2010, http://dx.doi.org/10.1142/7731.

[29] M. Ferrer, E. Valveny, F. Serratosa, K. Riesen, H. Bunke, Generalized median graph computation by means of graph embedding in vector spaces, Pattern Recognit. 43 (4) (2010) 1642–1655, http://dx.doi.org/10.1016/j.patcog.2009.10.013.

[30] M. Ferrer, D. Karatzas, E. Valveny, I. Bardaji, H. Bunke, A generic framework for median graph computation based on a recursive embedding approach, Comput. Vis. Image Underst. 115 (7) (2011) 919–928, http://dx.doi.org/10.1016/j.cviu.2010.12.010.

[31] M. Ferrer, I. Bardají, E. Valveny, D. Karatzas, H. Bunke, Median graph computation by means of graph embedding into vector spaces, in: Y. Fu, Y. Ma (Eds.), Graph Embedding for Pattern Analysis, Springer, New York, NY, 2013, pp. 45–71, http://dx.doi.org/10.1007/978-1-4614-4457-2_3.

[32] A. Nienkötter, X. Jiang, Improved prototype embedding based generalized median computation by means of refined reconstruction methods, in: A. Robles-Kelly, M. Loog, B. Biggio, F. Escolano, R.C. Wilson (Eds.), S+SSPR 2016, in: LNCS, vol. 10029, Springer, Cham, 2016, pp. 107–117, http://dx.doi.org/10.1007/978-3-319-49055-7_10.

[33] N. Boria, S. Bougleux, B. Gaüzère, L. Brun, Generalized median graph via iterative alternate minimizations, in: D. Conte, J. Ramel, P. Foggia (Eds.), GbRPR 2019, in: LNCS, vol. 11510, Springer, Cham, 2019, pp. 99–109, http://dx.doi.org/10.1007/978-3-030-20081-7_10.

[34] M.B. Cohen, Y.T. Lee, G.L. Miller, J. Pachocki, A. Sidford, Geometric median in nearly linear time, in: D. Wichs, Y. Mansour (Eds.), STOC 2016, ACM, New York, 2016, pp. 9–21, http://dx.doi.org/10.1145/2897518.2897647.

[35] E. Pekalska, R.P.W. Duin, P. Paclík, Prototype selection for dissimilarity-based classifiers, Pattern Recognit. 39 (2) (2006) 189–208, http://dx.doi.org/10.1016/j.patcog.2005.06.012.

[36] H. Bunke, S. Günter, Weighted mean of a pair of graphs, Computing 67 (3) (2001) 209–224, http://dx.doi.org/10.1007/s006070170006.

[37] R. Chaieb, K. Kalti, M.M. Luqman, M. Coustaty, J.-M. Ogier, N.E.B. Amara, Fuzzy generalized median graphs computation: Application to content-based document retrieval, Pattern Recognit. 72 (2017) 266–284, http://dx.doi.org/10.1016/j.patcog.2017.07.030.

[38] A.K.C. Wong, M. You, Entropy and distance of random graphs with application to structural pattern recognition, IEEE Trans. Pattern Anal. Mach. Intell. 7 (5) (1985) 599–607, http://dx.doi.org/10.1109/TPAMI.1985.4767707.

[39] A. Solé-Ribalta, F. Serratosa, Models and algorithms for computing the common labelling of a set of attributed graphs, Comput. Vis. Image Underst. 115 (7) (2011) 929–945, http://dx.doi.org/10.1016/j.cviu.2010.12.007.

[40] A. Solé-Ribalta, Multiple Graph Matching and Applications (Ph.D. thesis), Universitat Rovira i Virgili, Tarragona, Spain, 2012.

[41] N. Rebagliati, A. Solé-Ribalta, M. Pelillo, F. Serratosa, On the relation between the common labelling and the median graph, in: G.L. Gimel'farb, E.R. Hancock, A. Imiya, A. Kuijper, M. Kudo, S. Omachi, T. Windeatt, K. Yamada (Eds.), S+SSPR 2012, in: LNCS, vol. 7626, Springer, Berlin, Heidelberg, 2012, pp. 107–115, http://dx.doi.org/10.1007/978-3-642-34166-3_12.

[42] B.J. Jain, Statistical graph space analysis, Pattern Recognit. 60 (2016) 802–812, http://dx.doi.org/10.1016/j.patcog.2016.06.023.

[43] G. Peyré, M. Cuturi, J. Solomon, Gromov–Wasserstein averaging of kernel and distance matrices, in: M. Balcan, K.Q. Weinberger (Eds.), ICML 2016, in: Proceedings of Machine Learning Research, vol. 48, PMLR, New York, 2016, pp. 2664–2672, URL http://proceedings.mlr.press/v48/peyre16.pdf.

[44] T. Vayer, N. Courty, R. Tavenard, L. Chapel, R. Flamary, Optimal transport for structured data with application on graphs, in: K. Chaudhuri, R. Salakhutdinov (Eds.), ICML 2019, in: Proceedings of Machine Learning Research, vol. 97, PMLR, Long Beach, 2019, pp. 6275–6284, URL http://proceedings.mlr.press/v97/titouan19a/titouan19a.pdf.

[45] D. Justice, A. Hero, A binary linear programming formulation of the graph edit distance, IEEE Trans. Pattern Anal. Mach. Intell. 28 (8) (2006) 1200–1214, http://dx.doi.org/10.1109/TPAMI.2006.152.

[46] S. Bougleux, L. Brun, V. Carletti, P. Foggia, B. Gaüzère, M. Vento, Graph edit distance as a quadratic assignment problem, Pattern Recognit. Lett. 87 (2017) 38–46, http://dx.doi.org/10.1016/j.patrec.2016.10.001.

[47] L. Babai, Graph isomorphism in quasipolynomial time [extended abstract], in: D. Wichs, Y. Mansour (Eds.), STOC 2016, ACM, New York, 2016, pp. 684–697, http://dx.doi.org/10.1145/2897518.2897542.

[48] D.B. Blumenthal, S. Bougleux, J. Gamper, L. Brun, GEDLIB: A C++ library for graph edit distance computation, in: D. Conte, J.-Y. Ramel, P. Foggia (Eds.), GbRPR 2019, in: LNCS, vol. 11510, Springer, Cham, 2019, pp. 14–24, http://dx.doi.org/10.1007/978-3-030-20081-7_2.

[49] E. Weiszfeld, F. Plastria, On the point for which the sum of the distances to n given points is minimum, Ann. Oper. Res. 167 (2009) 7–41, http://dx.doi.org/10.1007/s10479-008-0352-z.

[50] Y. Vardi, C.-H. Zhang, The multivariate L1-median and associated data depth, Proc. Natl. Acad. Sci. USA 97 (4) (2000) 1423–1426, http://dx.doi.org/10.1073/pnas.97.4.1423.

[51] F. Nicolas, E. Rivals, Hardness results for the center and median string problems under the weighted and unweighted edit distances, J. Discrete Algorithms 3 (2–4) (2005) 390–415, http://dx.doi.org/10.1016/j.jda.2004.08.015.

[52] M. Hayashida, H. Koyano, Finding median and center strings for a probability distribution on a set of strings under Levenshtein distance based on integer linear programming, in: A.L.N. Fred, H. Gamboa (Eds.), BIOSTEC 2016, in: CCIS, vol. 690, Springer, Cham, 2017, pp. 108–121, http://dx.doi.org/10.1007/978-3-319-54717-6_7.

[53] J.I. Abreu, J.R. Rico-Juan, A new iterative algorithm for computing a quality approximate median of strings based on edit operations, Pattern Recognit. Lett. 36 (2014) 74–80, http://dx.doi.org/10.1016/j.patrec.2013.09.014.

[54] S. Sharma, P. Tripathi, Gut microbiome and type 2 diabetes: where we are and where to go? J. Nutr. Biochem. 63 (2019) 101–108, http://dx.doi.org/10.1016/j.jnutbio.2018.10.003.

[55] E.A. Franzosa, A. Sirota-Madi, J. Avila-Pacheco, N. Fornelos, H.J. Haiser, S. Reinker, T. Vatanen, A.B. Hall, H. Mallick, L.J. McIver, J.S. Sauk, R.G. Wilson, B.W. Stevens, J.M. Scott, K. Pierce, A.A. Deik, K. Bullock, F. Imhann, J.A. Porter, A. Zhernakova, J. Fu, R.K. Weersma, C. Wijmenga, C.B. Clish, H. Vlamakis, C. Huttenhower, R.J. Xavier, Gut microbiome structure and metabolic activity in inflammatory bowel disease, Nat. Microbiol. 4 (2) (2019) 293–305, http://dx.doi.org/10.1038/s41564-018-0306-4.

[56] G.B. Gloor, J.M. Macklaim, V. Pawlowsky-Glahn, J.J. Egozcue, Microbiome datasets are compositional: And this is not optional, Front. Microbiol. 8 (2017) 2224:1–2224:6, http://dx.doi.org/10.3389/fmicb.2017.02224.

[57] Y. Ban, L. An, H. Jiang, Investigating microbial co-occurrence patterns based on metagenomic compositional data, Bioinformatics 31 (20) (2015) 3322–3329, http://dx.doi.org/10.1093/bioinformatics/btv364.

[58] J. Friedman, E.J. Alm, Inferring correlation networks from genomic survey data, PLoS Comput. Biol. 8 (9) (2012) e1002687:1–e1002687:11, http://dx.doi.org/10.1371/journal.pcbi.1002687.

[59] H. Fang, C. Huang, H. Zhao, M. Deng, CCLasso: correlation inference for compositional data through Lasso, Bioinformatics 31 (19) (2015) 3172–3180, http://dx.doi.org/10.1093/bioinformatics/btv349.

[60] H. Hirano, K. Takemoto, Difficulty in inferring microbial community structure based on co-occurrence network approaches, BMC Bioinform. 20 (1) (2019) 329:1–329:14, http://dx.doi.org/10.1186/s12859-019-2915-1.

[61] L. Kaufman, P.J. Rousseeuw, Finding Groups in Data: An Introduction to Cluster Analysis, John Wiley & Sons, Hoboken, 1990, http://dx.doi.org/10.1002/9780470316801.

[62] E. Schubert, P.J. Rousseeuw, Faster k-medoids clustering: Improving the PAM, CLARA, and CLARANS algorithms, 2018, arXiv:1810.05691.

[63] H. Park, C. Jun, A simple and fast algorithm for K-medoids clustering, Expert Syst. Appl. 36 (2) (2009) 3336–3341.

[64] S.P. Lloyd, Least squares quantization in PCM, IEEE Trans. Inf. Theory 28 (2) (1982) 129–136, http://dx.doi.org/10.1109/TIT.1982.1056489.

[65] P.S. Bradley, O.L. Mangasarian, W.N. Street, Clustering via concave minimization, in: M. Mozer, M.I. Jordan, T. Petsche (Eds.), NIPS 1996, MIT Press, Cambridge, Massachusetts, 1996, pp. 368–374.

[66] D. Arthur, S. Vassilvitskii, k-means++: the advantages of careful seeding, in: N. Bansal, K. Pruhs, C. Stein (Eds.), SODA 2007, SIAM, Philadelphia, 2007, pp. 1027–1035.

[67] E. Chávez, G. Navarro, R.A. Baeza-Yates, J.L. Marroquín, Searching in metric spaces, ACM Comput. Surv. 33 (3) (2001) 273–321, http://dx.doi.org/10.1145/502807.502808.

[68] G. Wang, B. Wang, X. Yang, G. Yu, Efficiently indexing large sparse graphs for similarity search, IEEE Trans. Knowl. Data Eng. 24 (3) (2012) 440–451, http://dx.doi.org/10.1109/TKDE.2010.28.

[69] X. Wang, X. Ding, A.K.H. Tung, S. Ying, H. Jin, An efficient graph indexing method, in: A. Kementsietsidis, M.A.V. Salles (Eds.), ICDE 2012, IEEE Computer Society, Los Alamitos, 2012, pp. 210–221, http://dx.doi.org/10.1109/ICDE.2012.28.

[70] X. Zhao, C. Xiao, X. Lin, W. Wang, Y. Ishikawa, Efficient processing of graph similarity queries with edit distance constraints, VLDB J. 22 (6) (2013) 727–752, http://dx.doi.org/10.1007/s00778-013-0306-1.

[71] W. Zheng, L. Zou, X. Lian, D. Wang, D. Zhao, Efficient graph similarity search over large graph databases, IEEE Trans. Knowl. Data Eng. 27 (4) (2015) 964–978, http://dx.doi.org/10.1109/TKDE.2014.2349924.

[72] X. Zhao, C. Xiao, X. Lin, W. Zhang, Y. Wang, Efficient structure similarity searches: a partition-based approach, VLDB J. 27 (1) (2018) 53–78, http://dx.doi.org/10.1007/s00778-017-0487-0.

[73] I. Kalantari, G. McDonald, A data structure and an algorithm for the nearest point problem, IEEE Trans. Softw. Eng. 9 (5) (1983) 631–634, http://dx.doi.org/10.1109/TSE.1983.235263.

[74] H. Noltemeier, K. Verbarg, C. Zirkelbach, Monotonous bisector* trees – a tool for efficient partitioning of complex scenes of geometric objects, in: B. Monien, T. Ottmann (Eds.), Data Structures and Efficient Algorithms, Final Report on the DFG Special Joint Initiative, in: LNCS, vol. 594, Springer, Berlin, Heidelberg, 1992, pp. 186–203, http://dx.doi.org/10.1007/3-540-55488-2_27.

[75] K. Riesen, H. Bunke, IAM graph database repository for graph based pattern recognition and machine learning, in: N. da Vitoria Lobo, T. Kasparis, F. Roli, J.T. Kwok, M. Georgiopoulos, G.C. Anagnostopoulos, M. Loog (Eds.), S+SSPR 2008, in: LNCS, vol. 5342, Springer, Berlin, Heidelberg, 2008, pp. 287–297, http://dx.doi.org/10.1007/978-3-540-89689-0_33.

[76] Z. Abu-Aisheh, R. Raveaux, J.-Y. Ramel, A graph database repository and performance evaluation metrics for graph edit distance, in: C. Liu, B. Luo, W.G. Kropatsch, J. Cheng (Eds.), GbRPR 2015, in: LNCS, vol. 9069, Springer, Cham, 2015, pp. 138–147, http://dx.doi.org/10.1007/978-3-319-18224-7_14.

[77] A. Metwaly, A. Dunkel, N. Waldschmitt, A.C.D. Raj, I. Lagkouvardos, A.M. Corraliza, A. Mayorgas, M. Martinez-Medina, S. Reiter, M. Schloter, T. Hofmann, M. Allez, J. Panes, A. Salas, D. Haller, Integrated microbiota and metabolite profiles link Crohn's disease to sulfur metabolism, Nature Commun. 11 (1) (2020) 4322, http://dx.doi.org/10.1038/s41467-020-17956-1.

[78] S. Bougleux, B. Gaüzère, L. Brun, Graph edit distance as a quadratic program, in: ICPR 2016, IEEE Computer Society, Los Alamitos, 2016, pp. 1701–1706, http://dx.doi.org/10.1109/ICPR.2016.7899881.

[79] D.B. Blumenthal, É. Daller, S. Bougleux, L. Brun, J. Gamper, Quasimetric graph edit distance as a compact quadratic assignment problem, in: ICPR 2018, IEEE Computer Society, Los Alamitos, 2018, pp. 934–939, http://dx.doi.org/10.1109/ICPR.2018.8546055.

[80] M. Schirmer, A. Garner, H. Vlamakis, R.J. Xavier, Microbial genes and pathways in inflammatory bowel disease, Nat. Rev. Microbiol. 17 (8) (2019) 497–511, http://dx.doi.org/10.1038/s41579-019-0213-6.

[81] M.T. Alam, G.C.A. Amos, A.R.J. Murphy, S. Murch, E.M.H. Wellington, R.P. Arasaradnam, Microbial imbalance in inflammatory bowel disease patients at different taxonomic levels, Gut Pathog. 12 (2020) 1, http://dx.doi.org/10.1186/s13099-019-0341-6.

[82] L. Hubert, P. Arabie, Comparing partitions, J. Classification 2 (1) (1985) 193–218, http://dx.doi.org/10.1007/BF01908075.

[83] S. Bougleux, L. Brun, Linear sum assignment with edition, 2016, arXiv:1603.04380.

[84] L. Bahiense, G. Manic, B. Piva, C.C. de Souza, The maximum common edge subgraph problem: A polyhedral investigation, Discrete Appl. Math. 160 (18) (2012) 2523–2541, http://dx.doi.org/10.1016/j.dam.2012.01.026.

[85] D.B. Blumenthal, New Techniques for Graph Edit Distance Computation (Ph.D. thesis), Free University of Bozen-Bolzano, 2019, URL https://bia.unibz.it/handle/10863/10433.