AAM of Article Published in The VLDB Journal (2019)
`https://doi.org/10.1007/s00778-019-00544-1`

Comparing heuristics for graph edit distance computation

David B. Blumenthal · Nicolas Boria · Johann Gamper · Sébastien Bougleux · Luc Brun

Received: 31 December 2018 / Revised: 16 May 2019 / Accepted: 27 May 2019

Abstract Because of its flexibility, intuitiveness, and expressivity, the graph edit distance (GED) is one of the most widely used distance measures for labeled graphs. Since exactly computing GED is *NP*-hard, over the past years, various heuristics have been proposed. They use techniques such as transformations to the linear sum assignment problem with error-correction, local search, and linear programming to approximate GED via upper or lower bounds. In this paper, we provide a systematic overview of the most important heuristics. Moreover, we empirically evaluate all compared heuristics within an integrated implementation.

Keywords Graph edit Distance · Graph databases · Similarity search · Empirical evaluation

Mathematics Subject Classification (2010) 68R10 · 68T10 · 68P15 · 92E10

Contents

David B. Blumenthal, Johann Gamper
Free University of Bozen-Bolzano, Bolzano, Italy
E-mail: {david.blumenthal, gamper}@inf.unibz.it

Nicolas Boria, Sébastien Bougleux, Luc Brun
Normandie Univ., GREYC, ENSICAEN, UNICAEN, Caen, France
E-mail: {boria, brun}@ensicaen.fr, bougleux@unicaen.fr

1 Introduction

Labeled graphs can be used for modeling various kinds of objects, such as chemical compounds, images, molecular structures, and many more. Because of this flexibility, labeled graphs have received increasing attention over the past years. One task researchers have focused on is the following: Given a database $\mathscr{G}$ that contains labeled graphs, find all graphs $G \in \mathscr{G}$ that are sufficiently similar to a query graph H or find the k graphs from $\mathscr{G}$ that are most similar to H [23, 29, 66]. Being able to quickly answer graph similarity queries of this kind is crucial for the development of performant pattern recognition techniques in various application domains [62], such as keyword spotting in handwritten documents [61] and cancer detection [48].

For answering graph similarity queries, a distance measure between two labeled graphs G and H has to be defined. A very flexible, sensitive and therefore widely used measure is the graph edit distance (GED), which is defined as the minimum cost of an edit path between G and H [20, 59]. An edit path is a sequence of graphs starting at G and ending at a graph that is isomorphic to H such that each graph on the path can be obtained from its predecessor by applying one of the following edit operations: adding or deleting an isolated node or an edge, and relabelling an existing node or edge. Each edit operation comes with an associated non-negative edit cost, and the cost of an edit path is defined as the sum of the costs of its edit operations. GED inherits metric properties from the underlying edit costs [36]. For instance, if $\mathbb{G}$ is the domain of graphs with real-valued node and edge labels and the edit costs are defined as the Euclidean distances between the labels, then GED is a metric on $\mathbb{G}$.

Of course, computing GED is not the only way for assessing whether or not two graphs are similar. In particular, one popular approach is to embed the graphs into multidimensional vector spaces and then to compare their vector representations [19,23,29,66]. The main advantage of this paradigm is that it allows for fast computations. On the other hand, a substantial part of the information encoded in the original graphs is lost when embedding them into vector spaces. If the graphs are large (e. g., social networks or street networks), this information loss is tolerable. However, there are application domains such as keyword spotting in handwritten documents, cancer detection, and drug discovery, where the graphs are quite small and where local information that would be lost by embedding them into vector spaces is crucial [62]. GED is mainly used for these application domains, i. e., in settings where we have to answer fine-grained similarity queries for (possibly very many) rather small graphs.

Computing GED is a very difficult problem. In fact, it has been shown that the problem of computing GED is *NP*-hard even for uniform edit costs [68], and *APX*-hard for metric edit costs [45]. Even worse: since, by definition of GED, it holds that $\mathrm{GED}(G,H) = 0$ just in case G and H are isomorphic, approximating GED within any approximation ratio is *GI*-hard. These theoretical complexities are mirrored by the fact that, in practice, no available exact algorithm can reliably compute GED on graphs with more than 16 nodes [10].

Because of the hardness of exactly computing GED or approximating it within provable approximation ratios, during the past years, a huge variety of heuristics have been proposed that approximate GED via lower or upper bounds, using techniques such as transformations to the linear sum assignment problem with error-correction, linear programming, and local search. Since no theoretical guarantees can be provided for the produced bounds, the heuristics are always evaluated empirically. The most frequently used evaluation criteria are the following:

C1 Runtime behavior of the heuristics.
C2 Tightness of the produced lower or upper bounds.
C3 Performance of pattern recognition frameworks that use the bounds produced by the heuristics as underlying distance measures.

In this paper, we provide a systematic overview of the most important heuristics for the computation of GED. Figure 1 shows the suggested taxonomy. Whenever possible, we model the compared heuristics as instantiations of one of the following three paradigms: `LSAPE-GED`, `LP-GED`, and `LS-GED`. Instantiations of `LSAPE-GED` use transformations to the linear sum assignment problem with error-correction (LSAPE) for heuristically computing GED. All instantiation of `LSAPE-GED` produce upper bounds, some also yield lower bounds. Instantiations of `LP-GED` compute lower and upper bounds for GED by employing linear programming (LP) relaxations of mixed integer programming (MIP) formulations of GED. And instantiations of the paradigm `LS-GED` improve initially computed or randomly generated upper bounds by using variants of local search.

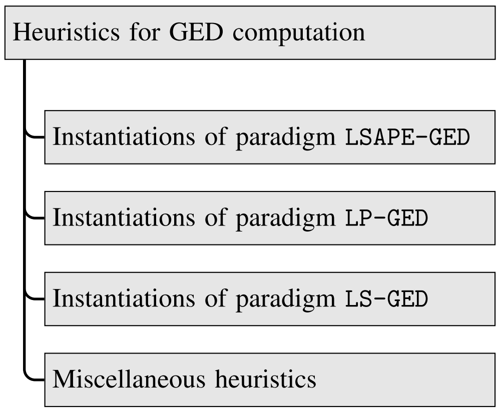

Heuristics for GED computation
Instantiations of paradigm LSAPE-GED
Instantiations of paradigm LP-GED
Instantiations of paradigm LS-GED
Miscellaneous heuristics

Fig. 1 Suggested taxonomy for heuristics for GED computation.

Locating the presented heuristics for GED within the suggested taxonomy has two main advantages: Firstly, it allows to clearly highlight differences and similarities between the presented heuristics. Secondly, the suggested taxonomy provides a guidance for implementing the heuristics in a clean, code-efficient, and comparable way, as common constituents of all instantiations of one of the paradigms can be implemented within an interface representing the paradigm. For instance, all instantiations of `LSAPE-GED` must have access to a solver for LSAPE. This solver can be implemented or called from an interface that represents the paradigm `LSAPE-GED`.

We carried out extensive experiments in order to test how the compared heuristics perform w. r. t. the evaluation criteria C1 to C3. For enhancing comparability, we reimplemented all heuristics within the C++ library GEDLIB and ensured that they use the same data structures and subroutines whenever possible. GEDLIB mirrors the taxonomy displayed in Figure 1, i. e., we implemented the paradigms `LSAPE-GED`, `LP-GED`, and `LS-GED` as abstract classes and their instantiations as derived classes. GEDLIB is freely available on GitHub: `https://github.com/dbblumenthal/gedlib/`.

An alternative view of the upper and lower bounds produced by heuristic algorithms for GED is to not regard them as proxies for GED, but rather as independent distance measures for labeled graphs whose design is guided by GED. With this interpretation, two meta-questions naturally arise:

Q1 Is it indeed beneficial to use GED as a guidance for the design of graph distance measures, if these distance measures are to be used within pattern recognition frameworks?
Q2 Do graph distance measures defined by upper bounds for GED or graph distance measures defined by lower bounds for GED perform better when used within pattern recognition frameworks?

To the best of our knowledge, these questions have not been explicitly discussed in the literature. In this paper, we intend to fill this gap. For addressing Q1, we evaluate if performing well w. r. t. the evaluation criterion C3 is positively correlated to performing well w. r. t. C2. For addressing Q2, we check if, globally, heuristics producing lower bounds or heuristics producing upper bounds perform better w. r. t. C3. In sum, our paper contains the following contributions:

– We suggest a taxonomy for algorithms that heuristically compute GED.
– We present the most important existing heuristics within this taxonomy.
– We present the results of an extensive empirical evaluation of all compared heuristics. For carrying out the experiments, all heuristics were reimplemented in C++.
– We empirically address the question whether or not GED constitutes a good guidance for the design of graph distance measures to be used for pattern recognition tasks.
– We empirically address the question whether lower or upper bounds for GED perform better when used for pattern recognition tasks.

The remainder of this paper is organized as follows: In Section 2, related work is discussed. In Section 3, important concepts and notations that are used throughout the paper are introduced. In Section 4, a first overview of the compared heuristics is provided and their most important properties are summarized in a comparative way. In the Sections 5 to 7, heuristics that instantiate the paradigms `LSAPE-GED`, `LP-GED`, and `LS-GED` are presented. In Section 8, miscellaneous heuristics that cannot be modeled as instantiations of one of the paradigms are presented. In Section 9, the outcomes of the experimental evaluation are presented. Section 10 concludes the paper. Appendix A contains short descriptions of the test datasets, and Appendix B contains further figures for visualizing the results of the experiments.

2 Related work

To the best of our knowledge, the present paper offers the first comprehensive comparative evaluation of algorithms for heuristically computing GED available in the literature. Nonetheless, there are similar works. In [10], some of the most important algorithms for exactly computing GED are evaluated. Structurally, this paper is very similar to the present one, but its scope is different, as heuristics are not considered.

In [1], the results of a graph edit distance contest are reported. Authors of exact or heuristic algorithms for GED could submit binaries of their algorithms, which were run and compared w. r. t. a set of evaluation criteria. As only the submitted algorithms are described in [1], many important GED heuristics are not covered. Moreover, [1] differs from the present paper in that it empirically evaluates implementations rather than algorithms.

In another survey [31], the main focus is on methods that learn good edit costs for given datasets. Only a few algorithms are presented that aim at computing GED for fixed edit costs, and those that are, are mostly designed for special graphs without node or edge labels. Moreover, the presented algorithms are not compared empirically.

The surveys [19,23,29,66] provide broader overviews of graph based methods for pattern recognition. In addition to GED, related topics such as exact graph matching and graph kernels are discussed. Like in [31], the presented methods are not compared experimentally in [19,23,29,66].

Methods for GED are also discussed in the books [49, 52]. In [52], some algorithms for exactly and heuristically computing GED are described and empirically evaluated, but the main focus is on how to use GED for defining graph kernels and vector space embeddings that can be employed for clustering and classification. The book [49] exclusively treats GED heuristics, but has a very narrow scope: While a few heuristics are described and tested in great detail, many others are not covered.

Finally, in [67,69,71], it is discussed how to index graph databases for efficiently answering graph similarity queries when GED is used as the underlying distance measure. However, as these papers focus on indexing techniques rather than on the design of heuristics, we do not present their findings in this survey.

3 Preliminaries

Since the graphs for which GED based methods are applied are mostly undirected [2,50,62], most heuristics for GED are presented for undirected labeled graphs, although they can usually be easily generalized to directed graphs. For the ease of presentation, we restrict to undirected graphs also in this paper. For the generalizations of the presented heuristics to directed graphs, we refer to the original publications.

An *undirected labeled graph* G is a 4-tuple $G = (V^G, E^G, \ell_V^G, \ell_E^G)$, where V^G and E^G are sets of nodes and edges, Σ_V and Σ_E are label alphabets, and $\ell_V^G : V^G \to \Sigma_V$, $\ell_E^G : E^G \to \Sigma_E$ are labeling functions. The *dummy symbol* ε denotes dummy nodes and edges as well as their labels. Throughout the paper, we denote the nodes of a graph G by $V^G := \{u_i \mid i \in [|V^G|]\}$ and the nodes of a graph H by $V^H := \{v_k \mid k \in [|V^H|]\}$. Furthermore, we use the notation $(u_i, u_j) := (u_j, u_i) := \{u_i, u_j\} \in E^G$ to denote that there is an undirected edge in G that connects the nodes u_i and u_j.

Let $\mathbb{G} := \{G \mid \mathrm{img}(\ell_V^G) \subseteq \Sigma_V \wedge \mathrm{img}(\ell_E^G) \subseteq \Sigma_E\}$ be the domain of graphs with node labels from Σ_V and edge labels from Σ_E. A function $c_V : \Sigma_V \cup \{\varepsilon\} \times \Sigma_V \cup \{\varepsilon\} \to \mathbb{R}_{\geq 0}$ is a *node edit cost functions* for $\mathbb{G}$ just in case, for all $(\alpha, \alpha') \in (\Sigma_V \cup \{\varepsilon\}) \times (\Sigma_V \cup \{\varepsilon\})$, $c_V(\alpha, \alpha') = 0$ holds if and only if $\alpha = \alpha'$. Similarly, $c_E : \Sigma_E \cup \{\varepsilon\} \times \Sigma_E \cup \{\varepsilon\} \to$

Table 1 Edit operations and edit costs.

edit operations	edit costs
node edit operations	
substitute α-labeled node by α'-labeled node	$c_V(\alpha,\alpha')$
delete isolated α-labeled node	$c_V(\alpha,\varepsilon)$
insert isolated α-labeled node	$c_V(\varepsilon,\alpha)$
edge edit operations	
substitute β-labeled edge by β'-labeled edge	$c_E(\beta,\beta')$
delete β-labeled edge	$c_E(\beta,\varepsilon)$
insert β-labeled edge between existing nodes	$c_E(\varepsilon,\beta)$

$\mathbb{R}_{\geq 0}$ is an *edge edit cost functions* for $\mathbb{G}$ just in case, for all $(\beta,\beta') \in (\Sigma_E \cup \{\varepsilon\}) \times (\Sigma_E \cup \{\varepsilon\})$, $c_E(\beta,\beta) = 0$ holds if and only if $\beta = \beta'$. We say that a node edit cost function c_V is *constant* just in case there are constants $c_V^{\text{sub}}, c_V^{\text{del}}, c_V^{\text{ins}} \in \mathbb{R}$ such that $c_V(\alpha,\alpha') = c_V^{\text{sub}}$, $c_V(\alpha,\varepsilon) = c_V^{\text{del}}$, and $c_V(\varepsilon,\alpha') = c_V^{\text{ins}}$ holds for all $(\alpha,\alpha') \in \Sigma_V \times \Sigma_V$ with $\alpha \neq \alpha'$. Constant edge edit costs are defined analogously. We say that the edit cost functions c_V and c_E are *uniform* if and only if both of them are constant and, additionally, we have $c_V^{\text{sub}} = c_V^{\text{del}} = c_V^{\text{ins}} = c_E^{\text{sub}} = c_E^{\text{del}} = c_E^{\text{ins}}$.

Given fixed edit cost functions c_V and c_E, GED is defined in terms of the six *edit operations* and their associated *edit costs*, which are detailed in Table 1. An *edit path P* between two graphs $G,H \in \mathbb{G}$ is a sequence $P := (o_i)_{i=1}^r$ of edit operations that satisfies $(o_r \circ \ldots \circ o_1)(G) \simeq H$, i. e., transforms G into a graph H' which is isomorphic to H. Its edit cost $c(P)$ is defined as the sum $c(P) := \sum_{i=1}^r c(o_i)$ of the contained edit operations. We are now in the position to give a first intuitive definition of GED.

Definition 1 (GED — first definition [10,36]) The *graph edit distance* (GED) between two graphs $G,H \in \mathbb{G}$ is defined as $\mathrm{GED}(G,H) := \min\{c(P) \mid P \in \Psi(G,H)\}$, where $\Psi(G,H)$ is the set of all edit paths between G and H.

Definition 1 is very intuitive but useless for algorithmic purposes: Since the graph isomorphism problem currently cannot be solved in polynomial time [3], it is not even possible to polynomially recognize an edit path as such, let alone to optimize over the set of all edit paths. For this reason, all exact or approximate GED algorithms work with an alternative definition. This definition crucially uses the concept of a node map between two graphs G and H.

Definition 2 (Node map [10]) Let $G,H \in \mathbb{G}$ be two graphs. A relation $\pi \subseteq (V^G \cup \{\varepsilon\}) \times (V^H \cup \{\varepsilon\})$ is called *node map* between G and H if and only if $|\{v \mid v \in (V^H \cup \{\varepsilon\}) \wedge (u,v) \in \pi\}| = 1$ holds for all $u \in V^G$ and $|\{u \mid u \in (V^G \cup \{\varepsilon\}) \wedge (u,v) \in \pi\}| = 1$ holds for all $v \in V^H$. We write $\pi(u) = v$ just in case $(u,v) \in \pi$ and $u \neq \varepsilon$, and $\pi^{-1}(v) = u$ just in case $(u,v) \in \pi$ and $v \neq \varepsilon$. $\Pi(G,H)$ denotes the set of all node maps between G and H. For edges $e = (u,u') \in E^G$ and $f = (v,v') \in E^H$, we introduce the short-hand notations $\pi(e) := (\pi(u),\pi(u'))$ and $\pi^{-1}(f) := (\pi^{-1}(v),\pi^{-1}(v'))$.

Table 2 Edit operations and edit costs induced by node map $\pi \in \Pi(G,H)$; $u \in V^G$ and $v \in V^H$ are nodes, $e \in E^G$ and $f \in E^H$ are edges.

case	edit operations	edit costs
node edit operations		
$\pi(u) = v$	substitute u by v	$c_V(u,v) := c_V(\ell_V^G(u),\ell_V^H(v))$
$\pi(u) = \varepsilon$	delete u	$c_V(u,\varepsilon) := c_V(\ell_V^G(u),\varepsilon)$
$\pi^{-1}(v) = \varepsilon$	insert v	$c_V(\varepsilon,v) := c_V(\varepsilon,\ell_V^H(v))$
edge edit operations		
$\pi(e) = f$	substitute e by f	$c_E(e,f) := c_E(\ell_E^G(e),\ell_E^H(f))$
$\pi(e) \notin E^H$	delete e	$c_E(e,\varepsilon) := c_E(\ell_E^G(e),\varepsilon)$
$\pi^{-1}(f) \notin E^G$	insert f	$c_E(\varepsilon,f) := c_E(\varepsilon,\ell_E^H(f))$

A node map $\pi \in \Pi(G,H)$ specifies for all nodes and edges of G and H whether they are substituted, deleted, or inserted. Table 2 details these edit operations.

Definition 3 (Induced edit path) Let $G,H \in \mathbb{G}$ be graphs, $\pi \in \Pi(G,H)$ be a node map between them, and O be the set of π's induced edit operations as specified in Table 2. Then an ordering $P_\pi := (o_r)_{r=1}^{|O|}$ of O is called *induced edit path* of the node map π if and only if edge deletions come first and edge insertions come last, i. e., if there are indices r_1 and r_2 such that o_r is an edge deletion just in case $1 \leq r < r_1$ and o_i is an edge insertion just in case $r_2 < r \leq |O|$.

It has been shown that induced edit paths are indeed edit paths, i. e., that $P_\pi \in \Psi(G,H)$ holds for all $\pi \in \Pi(G,H)$ [14]. The cost $c(P_\pi)$ of an edit path P_π induced by a node map $\pi \in \Pi(G,H)$ is given as follows:

$$c(P_\pi) = \underbrace{\sum_{\substack{u \in V^G \\ \pi(u) \in V^H}} c_V(u,\pi(u))}_{\text{node substitutions}} + \underbrace{\sum_{\substack{e \in E^G \\ \pi(e) \in E^H}} c_V(e,\pi(e))}_{\text{edge substitutions}} + \underbrace{\sum_{\substack{u \in V^G \\ \pi(u) \notin V^H}} c_V(u,\varepsilon)}_{\text{node deletions}} + \underbrace{\sum_{\substack{e \in E^G \\ \pi(e) \notin E^H}} c_E(e,\varepsilon)}_{\text{edge deletions}} + \underbrace{\sum_{\substack{v \in V^H \\ \pi^{-1}(v) \notin V^G}} c_V(\varepsilon,v)}_{\text{node insertions}} + \underbrace{\sum_{\substack{f \in E^H \\ \pi^{-1}(f) \notin E^G}} c_E(\varepsilon,f)}_{\text{edge insertions}} \quad (1)$$

Note that, by Definition 3, a node map π generally induces more than one edit path. However, all of these edit paths are equivalent, as they differ only w. r. t. the ordering of the contained edit operations. In the following, we will therefore identify all edit paths induced by π. We can now give an alternative definition of GED.

Table 3 Edit operations and edit costs induced by node map π shown in Figure 2, given the edit cost functions defined in Example 1.

edit operations	edit costs
node edit operations	
substitute u_1 by v_1	$c_V(u_1,v_1) = 0.75 \left\| \begin{bmatrix}0.69\\0.27\end{bmatrix} - \begin{bmatrix}0.92\\0.32\end{bmatrix} \right\|$
substitute u_2 by v_2	$c_V(u_2,v_2) = 0.75 \left\| \begin{bmatrix}1.40\\1.85\end{bmatrix} - \begin{bmatrix}1.76\\1.81\end{bmatrix} \right\|$
substitute u_3 by v_3	$c_V(u_3,v_3) = 0.75 \left\| \begin{bmatrix}2.55\\0.45\end{bmatrix} - \begin{bmatrix}2.30\\0.21\end{bmatrix} \right\|$
substitute u_4 by v_4	$c_V(u_4,v_4) = 0.75 \left\| \begin{bmatrix}0.93\\1.37\end{bmatrix} - \begin{bmatrix}0.92\\0.85\end{bmatrix} \right\|$
delete node u_5	$c_V(u_5,\varepsilon) = 0.675$
edge edit operations	
substitute (u_1,u_2) by (v_1,v_2)	$c_E((u_1,u_2),(v_1,v_2)) = 0$
substitute (u_2,u_3) by (v_2,v_3)	$c_E((u_2,u_3),(v_2,v_3)) = 0$
delete (u_4,u_5)	$c_E((u_4,u_5),\varepsilon) = 0.425$
insert (v_3,v_4)	$c_E(\varepsilon,(v_3,v_4)) = 0.425$

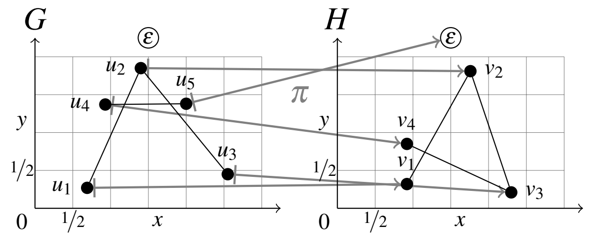


Fig. 2 Two graphs G and H from the LETTER (H) dataset and a node map π between them.

Definition 4 (GED — alternative definition [14, 20]) The *graph edit distance* (GED) between two graphs $G, H \in \mathbb{G}$ is defined as $\mathrm{GED}(G,H) := \min\{c(P_\pi) \mid \pi \in \Pi(G,H)\}$, where $P_\pi \in \Psi(G,H)$ is the edit path induced by the node map π.

In [36], it is shown that Definition 1 and Definition 4 are equivalent if the underlying edit cost functions c_V and c_E are metrics. In [10], this result is extended to general edit cost functions. There main advantage of using Definition 4 instead of Definition 1 is that, unlike edit paths, node maps can be constructed easily. Since each node map $\pi \in \Pi(G,H)$ induces an upper bound $c(P_\pi) \geq \mathrm{GED}(G,H)$, one can hence straightforwardly generate upper bounds.

Example 1 Consider the graphs G and H shown in Figure 2. G and H are taken from the LETTER (H) dataset and represent distorted letter drawings [50]. Their nodes are labeled with two-dimensional, non-negative Euclidean coordinates. Edges are unlabeled. Hence, we have $\Sigma_V = \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0}$ and $\Sigma_E = \{1\}$. In [52], it is suggested that the edit cost functions c_V and c_E for LETTER (H) should be defined as follows: $c_E(1,\varepsilon) := c_E(\varepsilon,1) := 0.425$, $c_V(\alpha,\alpha') := 0.75\,\|\alpha - \alpha'\|$, and $c_V(\alpha,\varepsilon) := c_V(\varepsilon,\alpha) := 0.675$ for all node labels $\alpha, \alpha' \in \Sigma_V$, where $\|\cdot\|$ is the Euclidean norm. Now consider the node map $\pi \in \Pi(G,H)$ shown in Figure 2. Its induced edit operations and edit costs are detailed in Table 3. By summing the induced edit costs, we obtain that π's induced edit path $P_\pi \in \Psi(G,H)$ has cost $c(P_\pi) = 2.623179$, which implies $\mathrm{GED}(G,H) \leq 2.623179$.

We conclude this section by recalling mathematical concepts and notations that are used throughout the paper.

Definition 5 (Miscellaneous definitions) In the remainder of this paper, we use the following definitions:

- For all $N \in \mathbb{N}$, we define $[N] := \{n \in \mathbb{N} \mid 1 \leq n \wedge n \leq N\}$.
- Let $G \in \mathbb{G}$ be a graph. A edge sequence $((u_{i_1}, u_{i_2}))_{i=1}^{k}$, $(u_{i_1}, u_{i_2}) \in E^G$ for all $i \in [k]$, is called *walk of length k* between the nodes $u_{i_1}, u_{k_2} \in V^G$, if and only if $u_{i_2} = u_{i+1_1}$ holds for all $i \in [k-1]$.
- Let $G \in \mathbb{G}$ be a graph. A walk between two nodes $u, u' \in V^G$ is called *path between u and u'*, if and only if no node is encountered more than once.
- Let $G \in \mathbb{G}$ be a graph. The *distance between two nodes* $u, u' \in V^G$ in G, is defined as $d^G(u,u') := 0$, if $u = u'$, as $d^G(u,u') := \min\{|P| \mid P \text{ is path between } u \text{ and } u'\}$, if $u \neq u'$ and u and u' are in the same connected component of G, as $d^G(u,u') := \infty$, if u and u' are in different connected components of G.
- The *diameter of a graph* $G \in \mathbb{G}$ is defined as $\mathrm{diam}(G) := \max_{u \in V^G} \max_{u' \in V^G} d^G(u,u')$.
- Let $G \in \mathbb{G}$ be a graph. The k^{th} *neighborhood of a node* $u \in V^G$ in G is defined as $N_k^G(u) := \{u' \in V^G \mid d^G(u,u') = k\}$. The 1st neighborhood is called *neighborhood of node u* and abbreviated as $N^G(u) := N_1^G(u)$.
- Let $G \in \mathbb{G}$ be a graph. The *set of edges that are incident with a node* $u \in V^G$ is denoted by $E^G(u)$.
- Let $G \in \mathbb{G}$ be a graph. The *degree of a node* $u \in V^G$ in G is defined as $\deg^G(u) := |N^G(u)|$.
- The *maximum degree of a graph* $G \in \mathbb{G}$ is defined as $\max\deg(G) := \max_{u \in V^G} \deg^G(u)$.
- Let $f : X \to Y$ be a function and $A \subseteq X$ be a subset of its domain. The *image of A under f* is denoted by $f[A]$, and the *multiset image of A under f* is denoted by $f[\![A]\!]$.
- Let $G \in \mathbb{G}$ be a graph and $V \subseteq V^G$ be a subset of its nodes. Then $G[V] := (V, E^G \cap (V \times V), \ell_V^G, \ell_E^G)$ denotes the subgraph of G which is induced by the node set V.
- The expression $\delta_{\texttt{true}|\texttt{false}}$ is defined as $\delta_{\texttt{true}} := 1$ and $\delta_{\texttt{false}} := 0$.

4 Overview of compared heuristics

Table 4 gives an overview of the heuristics that are compared in this survey. Each heuristic is denoted by a name written in typewriter font. Whenever possible, this name is taken from the original publication. If no original name is available, we invented a name which reflects the main technical ingredient of the heuristic.

Some heuristics are categorized as extensions rather than instantiations of the paradigms `LSAPE-GED` and `LS-GED`,

Table 4 Overview of compared heuristics.

heuristic	publications	upper bound	lower bound	limitations	presented in
instantiations of the paradigm `LSAPE-GED`					
`NODE`	[36]	yes	yes	ignores edges	Section 5.2.1
`BP`	[51]	yes	no	none	Section 5.2.2
`BRANCH`	[8,9,57]	yes	yes	none	Section 5.2.3
`BRANCH-FAST`	[8,9]	yes	yes	none	Section 5.2.4
`BRANCH-CONST`	[70,71]	yes	yes	constant c_E	Section 5.2.5
`STAR`	[68]	yes	yes	ignores edge labels, uniform c_V and c_E	Section 5.2.6
`SUBGRAPH`	[21]	yes	no	none	Section 5.2.7
`WALKS`	[32]	yes	no	constant and symmetric c_V and c_E	Section 5.2.8
`RING`	[4,6]	yes	no	none	Section 5.2.9
`RING-ML`	[6]	yes	no	none	Section 5.2.10
`PREDICT`	[6,54]	yes	no	none	Section 5.2.11
extensions of the paradigm `LSAPE-GED`					
`CENTRALITIES`	[25,53]	yes	no	none	Section 5.3.1
`MULTI-SOL`	[6,26]	yes	no	none	Section 5.3.2
instantiations of the paradigm `LP-GED`					
`F1`	[43,44]	yes	yes	none	Section 6.2.1
`F2`	[44]	yes	yes	none	Section 6.2.2
`COMPACT-MIP`	[10]	yes	yes	none	Section 6.2.3
`ADJ-IP`	[36]	yes	yes	ignores edge labels, constant and symmetric c_E	Section 6.2.4
instantiations of the paradigm `LS-GED`					
`REFINE`	[12,68]	yes	no	none	Section 7.2.1
`K-REFINE`	[12]	yes	no	none	Section 7.2.2
`BP-BEAM`	[56]	yes	no	none	Section 7.2.3
`IBP-BEAM`	[27]	yes	no	none	Section 7.2.4
`IPFP`	[7,14,16]	yes	no	none	Section 7.2.5
extensions of the paradigm `LS-GED`					
`MULTI-START`	[13,26]	yes	no	none	Section 7.3.1
`RANDPOST`	[12,13]	yes	no	none	Section 7.3.2
miscellaneous heuristics					
`HED`	[28]	no	yes	none	Section 8.1
`BRANCH-TIGHT`	[9]	yes	yes	none	Section 8.2
`SA`	[58]	yes	no	none	Section 8.3
`BRANCH-COMPACT`	[71]	no	yes	uniform c_V and c_E	Section 8.4
`PARTITION`	[71]	no	yes	uniform c_V and c_E	Section 8.5
`HYBRID`	[71]	no	yes	uniform c_V and c_E	Section 8.6

respectively. Although, in the original publications, these heuristics are usually presented as improvements of a specific instantiation of the respective paradigm, they can in fact be used to improve all instantiations. For instance, in [26], it is suggested that the local search algorithm `IPFP` should be run from several initial solutions in parallel. As this technique does not depend in the choice of the local search algorithm, it can be generalized to paradigm level and hence yields the extension `MULTI-START` of the paradigm `LS-GED`. In this survey, we generalize techniques to paradigm level whenever possible.

Moreover, some of the heuristics are designed for special edit cost functions. For instance, `BRANCH-CONST` requires constant edge edit costs, i. e., expects that there are constants $c^{\text{sub}}_E, c^{\text{del}}_E, c^{\text{ins}}_E \in \mathbb{R}$ such that $c_E(\beta,\beta') = c^{\text{sub}}_E$, $c_E(\beta,\varepsilon) = c^{\text{del}}_E$, and $c_V(\varepsilon,\beta') = c^{\text{ins}}_E$ holds for all $(\beta,\beta') \in \Sigma_E \times \Sigma_E$ with $\beta \neq \beta'$. In many datasets, this constraint is not satisfied. In our implementation of `BRANCH-CONST`, we therefore set $c^{\text{sub}}_E := \min\{c_E(\beta,\beta') \mid (\beta,\beta') \in \ell^G_E[E^G] \times \ell^H_E[E^H] \land \beta \neq \beta'\}$, $c^{\text{del}}_E := \min\{c_E(\beta,\varepsilon) \mid \beta \in \ell^G_E[E^G]\}$, and $c^{\text{ins}}_E := \min\{c_E(\varepsilon,\beta') \mid \beta' \in \ell^H_E[E^H]\}$ when running `BRANCH-CONST` on graphs G and H that come with non-constant edge edit costs. Since we use minima for defining the constants, this preprocessing leaves `BRANCH-CONST`'s lower bound valid. Similar techniques are used for enforcing the cost constraints of the other methods that are not designed for general edit costs.

5 Heuristics based on transformations to the linear sum assignment problem with error-correction

In this section, we first introduce the paradigm `LSAPE-GED`, which generalizes heuristics that use transformations to the linear sum assignment problem with error-correction (LSAPE) for upper and, possibly, lower bounding GED (Section 5.1). Subsequently, we present heuristics that can be modeled as instantiations of `LSAPE-GED` (Section 5.2), and summarize heuristics that can be modeled as extensions of `LSAPE-GED` (Section 5.3).

5.1 The paradigm `LSAPE-GED`

Definition 4 implies that each node map between G and H induces an upper bound for $\mathrm{GED}(G,H)$. Instantiations of the paradigm `LSAPE-GED` use transformations to LSAPE for heuristically finding a node map that induces a tight upper bound. LSAPE is defined as follows:

Definition 6 (LSAPE — first definition [15]) Given a matrix $\mathbf{C} \in \mathbb{R}^{(n+1)\times(m+1)}$ with $c_{n+1,m+1} = 0$, the *linear sum assignment problem with error-correction* (LSAPE) consists in the task to minimize $\mathbf{C}(\pi) := \sum_{(i,k)\in\pi} c_{i,k}$ over all relations $\pi \in \Pi(n,m)$, where

– $\Pi(n,m) \subseteq \mathscr{P}([n+1]\times[m+1])$ is defined as the set of all feasible LSAPE solutions for $\mathbf{C}$, and
– a relation $\pi \subseteq [n+1]\times[m+1]$ is called feasible LSAPE solution for $\mathbf{C}$ if and only if $|\{k \mid k\in[m+1] \wedge (i,k)\in\pi\}| = 1$ holds for all $i\in[n]$ and $|\{i \mid i\in[n+1] \wedge (i,k)\in\pi\}| = 1$ holds for all $k\in[m]$.

We write $\pi(i) = k$ if $(i,k)\in\pi$ and $i \neq n+1$; and $\pi^{-1}(k) = i$ if $(i,k)\in\pi$ and $k \neq m+1$. The set of all optimal LSAPE solutions for $\mathbf{C}$ is denoted as $\Pi^\star(\mathbf{C}) := \arg\min_{\pi\in\Pi(n,m)} \mathbf{C}(\pi)$. We write $\mathrm{LSAPE}(\mathbf{C}) := \min_{\pi\in\Pi(n,m)} \mathbf{C}(\pi)$ for the cost of an optimal solution for $\mathbf{C}$.

Given a matrix $\mathbf{C} \in \mathbb{R}^{(n+1)\times(m+1)}$, an optimal solution $\pi \in \Pi^\star(\mathbf{C})$ can be computed in $O(\min\{n,m\}^2 \max\{n,m\})$ time [15], using variants of the famous Hungarian Algorithm [38,46]. Once one optimal solution has been found, for each $s \in [|\Pi^\star(\mathbf{C})|]$, a solution set $\Pi^\star_s(\mathbf{C}) \subseteq \Pi^\star(\mathbf{C})$ of size s can be enumerated in $O(nm\sqrt{n+m} + s\log(n+m))$ time [64,65]. Greedy suboptimal solutions can be computed in $O(nm)$ time [55].

Node maps and feasible solutions for LSAPE are closely related. Assume that we are given graphs G and H and an LSAPE instance $\mathbf{C} \in \mathbb{R}^{(|V^G|+1)\times(|V^H|+1)}$. Then it immediately follows from Definition 2 and Definition 6 that we can identify the set $\Pi(G,H)$ of all node maps between G and H with the set $\Pi(|V^G|,|V^H|)$ of all feasible LSAPE solutions for $\mathbf{C}$: For all $i \in [|V^G|]$ and all $k \in [|V^H|]$, we associate $\mathbf{C}$'s i^{th} row with the node $u_i \in V^G$ and $\mathbf{C}$'s k^{th} column with the node $v_k \in V^H$. The last row and the last column of $\mathbf{C}$ are associated with the dummy node ε. Therefore, each feasible LSAPE solution π for $\mathbf{C}$ yields an upper bound for $\mathrm{GED}(G,H)$, namely, the cost $c(\pi)$ of the edit path induced by π's interpretation as a node map.

Sometimes, it is useful to view LSAPE not as an optimization problem over relations, but rather as an optimization problem over binary matrices. With this view, LSAPE can be equivalently defined as follows.

$$\mathbf{X} = \begin{array}{c} \\ 1\\2\\3\\4\\5\\6 \end{array}\!\!\begin{array}{cccc|c} 1&2&3&4&5\\ \hline \mathbf{1}&0&0&0&0\\ 0&\mathbf{1}&0&0&0\\ 0&0&\mathbf{1}&0&0\\ 0&0&0&\mathbf{1}&0\\ 0&0&0&0&\mathbf{1}\\ \hline 0&0&0&0&0 \end{array} \qquad \mathbf{X}' = \begin{array}{c} \\ 1\\2\\3\\4\\5\\6 \end{array}\!\!\begin{array}{cccc|c} 1&2&3&4&5\\ \hline \mathbf{1}&0&0&0&0\\ 0&\mathbf{1}&0&0&0\\ 0&0&0&0&\mathbf{1}\\ 0&0&\mathbf{1}&0&0\\ 0&0&0&0&\mathbf{1}\\ \hline 0&0&0&\mathbf{1}&0 \end{array}$$

Fig. 3 Two LSAPE solutions of size 6×5 in matrix representation. If viewed as node maps between graphs G and H with $|V^G| = 5$ and $|V^H| = 6$, the last row corresponds to the dummy node ε in G and the last column corresponds to the dummy node ε in H.

Definition 7 (LSAPE — alternative definition [15]) Given a matrix $\mathbf{C} \in \mathbb{R}^{(n+1)\times(m+1)}$ with $c_{n+1,m+1} = 0$, the *linear sum assignment problem with error-correction* (LSAPE) consists in the task to minimize $\mathbf{C}(\mathbf{X}) := \sum_{i=1}^{n+1}\sum_{k=1}^{m+1} c_{i,k}x_{i,k}$ over all binary matrices $\mathbf{X} \in \Pi(n,m)$, where

– $\Pi(n,m) \subseteq \{0,1\}^{(n+1)\times(m+1)}$ is defined as the set of all feasible LSAPE solutions for $\mathbf{C}$, and
– a binary matrix $\mathbf{X} \in \{0,1\}^{(n+1)\times(m+1)}$ is called feasible LSAPE solution for $\mathbf{C}$ if and only if $\sum_{k=1}^{m+1} x_{i,k} = 1$ holds for all $i \in [n]$ and $\sum_{i=1}^{n+1} x_{i,k} = 1$ holds for all $k \in [m]$.

Example 2 Let $\mathbf{C} \in \mathbb{R}^{6\times5}$ be a matrix and again consider the node map $\pi \in \Pi(G,H)$ visualized in Figure 2. The node map π's interpretation as a feasible LSAPE solution for $\mathbf{C}$ in relational form is given by $\{(i,i) \mid i \in [5]\}$. Its matrix representation $\mathbf{X}$ is shown in Figure 3, along with another feasible LSAPE solution $\mathbf{X}'$. Instead of substituting row 3 by column 3 and row 4 by column 4, $\mathbf{X}'$ substitutes row 4 by column 3, deletes row 3 and inserts column 4.

Instantiations of the paradigm `LSAPE-GED` now proceed as described in Algorithm 1: In a first step, the input graphs G and H and the edit cost functions are used to construct an LSAPE instance $\mathbf{C}$ of size $(|V^G|+1)\times(|V^H|+1)$ such that optimal LSAPE solutions for $\mathbf{C}$ induce cheap edit paths between G and H (line 1). This construction phase is where different instantiations of `LSAPE-GED` vary from each other. Subsequently, the LSAPE instance $\mathbf{C}$ is solved — either optimally or greedily — and the cost $c(P_\pi)$ of the edit path induced by the obtained LSAPE solution π is interpreted

Algorithm 1 The paradigm LSAPE-GED.

```
Input: Graphs $G$ and $H$, node edit costs $c_V$, edge edit costs $c_E$.
Output: An upper bound $UB$ and, possibly, a lower bound $LB$ for $\mathrm{GED}(G,H)$.
 1: use information encoded in $G$, $H$, $c_V$, and $c_E$ to construct LSAPE instance $\mathbf{C} \in \mathbb{R}^{(|V^G|+1)\times(|V^H|+1)}$;
 2: use optimal or greedy solver to compute cheap LSAPE solution $\pi \in \Pi(|V^G|,|V^H|)$;
 3: set upper bound to $UB := c(P_\pi)$;
 4: if line 1 ensures $\mathrm{LSAPE}(\mathbf{C}) \leq \xi(G,H,c_V,c_E)\cdot\mathrm{GED}(G,H)$ then
 5:     if optimal solver was used in line 1 then
 6:         set lower bound to $LB := \xi(G,H,c_V,c_E)\mathbf{C}(\pi)$;
 7:         return $LB$ and $UB$;
 8:     else
 9:         return $UB$;
10:     end if
11: end if
12: return $UB$;
```

as an upper bound for $\mathrm{GED}(G,H)$ (line 2). If the protocol for constructing the LSAPE instance $\mathbf{C}$ ensures that one can define a scaling function $\xi(G,H,c_V,c_E)$ such that

$$\xi(G,H,c_V,c_E)\mathrm{LSAPE}(\mathbf{C}) \leq \mathrm{GED}(G,H) \tag{2}$$

holds for all graphs $G,H \in \mathbb{G}$ and all edit cost functions c_V and c_E and an optimal LSAPE solver was used to compute π, $\xi(G,H,c_V,c_E)\mathbf{C}(\pi)$ is returned as a lower bound for GED along with the upper bound derived from the induced edit path (lines 4 to 7). Otherwise, only the upper bound is returned (lines 9 to 12).

Assume that an instantiation of LSAPE-GED constructs its LSAPE instance $\mathbf{C}$ in $O(\omega)$ time. Optimally solving $\mathbf{C}$ requires $O(\min\{|V^G|,|V^H|\}^2\max\{|V^G|,|V^H|\})$ time, while the complexity of greedily computing a cheap suboptimal solution is $O(|V^G||V^H|)$. The induced cost of the obtained node map π can be computed in $O(\max\{|E^G|,|E^H|\})$ time. The heuristic's overall runtime complexity is hence $O(\omega + \min\{|V^G|,|V^H|\}^2\max\{|V^G|,|V^H|\} + \max\{|E^G|,|E^H|\})$ if an optimal solver is used in line 2, and $O(\omega + |V^G||V^H| + \max\{|E^G|,|E^H|\})$ if $\mathbf{C}$ is solved greedily.

5.2 Instantiations of the paradigm LSAPE-GED

Next, we present eleven algorithms for heuristically computing GED that can be modeled as instantiations of the paradigm LSAPE-GED. All heuristics compute upper bounds for GED, and some of them also compute lower bounds. Some of the heuristics require special edit costs, while others can be used with general edit costs.

5.2.1 The algorithm NODE

The algorithm NODE [36] is a very simple instantiation of LSAPE: It completely ignores the edges of the input graphs G and H and just defines $\mathbf{C}$ as the node edit cost matrix between G and H. In other words, it sets

$$\begin{aligned} c_{i,k} &:= c_V(u_i,v_k) \\ c_{i,|V^H|+1} &:= c_V(u_i,\varepsilon) \\ c_{|V^G|+1,k} &:= c_V(\varepsilon,v_k) \end{aligned}$$

for all $(i,k) \in [|V^G|] \times [|V^H|]$.

The time complexity of constructing $\mathbf{C}$ is $O(|V^G||V^H|)$. As ineq. (2) with $\xi :\equiv 1$ holds for all graphs $G,H \in \mathbb{G}$ and all edit cost functions c_V and c_E, NODE computes both an upper and a lower bound for GED.

5.2.2 The algorithm BP

Unlike NODE the algorithm BP [51] also considers edges. Informally, this is done by adding to $c_{i,k}$ as defined by NODE the optimal cost of transforming the edges that are incident with u_i in G into the edges that are incident with v_k in H.

Formally, for each $(i,k) \in [|V^G|] \times [|V^H|]$, an auxiliary LSAPE instance $\mathbf{C}^{i,k} \in \mathbb{R}^{(\deg^G(u_i)+1)\times(\deg^H(v_k)+1)}$ is constructed. Let $(u_{i_j})_{j=1}^{\deg^G(u_i)}$ be an enumeration of u_i's neighborhood $N^G(u_i)$, and $(v_{k_l})_{l=1}^{\deg^H(v_k)}$ be an enumeration of v_k's neighborhood $N^H(v_k)$. BP sets

$$\begin{aligned} c^{i,k}_{j,l} &:= c_E((u_i,u_{i_j}),(v_k,v_{k_l})) \\ c^{i,k}_{j,\deg^H(v^k)+1} &:= c_E((u_i,u_{i_j}),\varepsilon) \\ c^{i,k}_{\deg^G(u_i),l} &:= c_E(\varepsilon,(v_k,v_{k_l})) \end{aligned}$$

for all $(j,l) \in [\deg^G(i)] \times [\deg^H(i)]$, and computes an optimal LSAPE solution $\pi^{i,k} \in \Pi(\deg^G(u_i),\deg^H(v_k))$. Once this has been done for all $(i,k) \in [|V^G|] \times [|V^H|]$, the final LSAPE instance $\mathbf{C}$ is constructed by setting

$$\begin{aligned} c_{i,k} &:= c_V(u_i,v_k) + \mathbf{C}^{i,k}(\pi^{i,k}) \\ c_{i,|V^H|+1} &:= c_V(u_i,\varepsilon) + \sum_{j=1}^{\deg^G(u_i)} c_E((u_i,u_{i_j}),\varepsilon) \\ c_{|V^G|+1,k} &:= c_V(\varepsilon,v_k) + \sum_{l=1}^{\deg^H(v_k)} c_E(\varepsilon,(v_k,v_{k_l})) \end{aligned}$$

for all $(i,k) \in [|V^G|] \times [|V^H|]$.

BP requires $O(|V^G||V^H|{\Delta^{G,H}_{\min}}^2\Delta^{G,H}_{\max})$ time for constructing $\mathbf{C}$, where $\Delta^{G,H}_{\min} := \min\{\max\deg(G),\max\deg(H)\}$ and $\Delta^{G,H}_{\max} := \max\{\max\deg(G),\max\deg(H)\}$. This construction does not guarantee that ineq. (2) holds, which implies that BP only returns an upper bound for GED.

5.2.3 The algorithm BRANCH

The algorithm BRANCH [8,9,57] is a slight modification of BP that also allows for the computation of a lower bound. The only modification is that the edge costs in the construction of the LSAPE instance $\mathbf{C}$ are divided by 2, i. e., $\mathbf{C}$ is defined by setting

$$\begin{aligned}
c_{i,k} &:= c_V(u_i,v_k) + 0.5 \cdot \mathbf{C}^{i,k}(\pi^{i,k}) \\
c_{i,|V^H|+1} &:= c_V(u_i,\varepsilon) + 0.5 \cdot \sum_{j=1}^{\deg^G(u_i)} c_E((u_i,u_{i_j}),\varepsilon) \\
c_{|V^G|+1,k} &:= c_V(\varepsilon,v_k) + 0.5 \cdot \sum_{l=1}^{\deg^H(v_k)} c_E(\varepsilon,(v_k,v_{k_l}))
\end{aligned}$$

for all $(i,k) \in [|V^G|] \times [|V^H|]$.

Like BP, BRANCH requires $O(|V^G||V^H|{\Delta_{\min}^{G,H}}^2 \Delta_{\max}^{G,H})$ time for constructing $\mathbf{C}$. Unlike BP, the construction carried out by BRANCH ensures that ineq. (2) with $\xi :\equiv 1$ holds for all graphs $G,H \in \mathbb{G}$ and all edit cost functions c_V and c_E, which implies that BRANCH also computes a lower bound for GED. Furthermore, it is shown that, given fixed metric edit cost functions c_V and c_E, the lower bound returned by BRANCH is a pseudo-metric on $\mathbb{G}$, i. e., is symmetric, non-negative, respects the triangle inequality, and equals 0 if two graphs $G,H \in \mathbb{G}$ are isomorphic.

5.2.4 The algorithm BRANCH-FAST

The algorithm BRANCH-FAST suggested in [8,9] speeds-up BRANCH at the cost of producing a looser lower bound. For all $(i,k) \in [|V^G|] \times [|V^H|]$, BRANCH-FAST computes $\Gamma_E^{i,k}$ as the size of the intersection of the multisets of edge labels that are incident to u_i in G and to v_k in H, i. e., sets $\Gamma_E^{i,k} := |\ell_E^G[\![E^G(u_i)]\!] \cap \ell_E^H[\![E^H(v_k)]\!]|$. Moreover, BRANCH-FAST computes the minimal deletion cost $c_{\min}^i := \min\{c_E(e,\varepsilon) \mid e \in E^G(u_i)\}$, the minimal insertion cost $c_{\min}^k := \min\{c_E(\varepsilon,f) \mid f \in E^H(v_k)\}$, as well as the minimal substitution cost $c_{\min}^{i,k} := \min\{c_E(e,f) \mid (e,f) \in E^G(u_i) \times E^H(v_k) \wedge \ell_E^G(e) \neq \ell_E^H(f)\}$ for the sets $E^G(u_i)$ and $E^H(v_k)$ of edges that are incident to u_i in G and to v_k in H, respectively. With these ingredients, BRANCH-FAST constructs its LSAPE instance $\mathbf{C}$ by setting

$$\begin{aligned}
c_{i,k} := c_V(u_i,v_k) + 0.5 \cdot [&(\Delta_{\min}^{i,k} - \Gamma_E^{i,k})c_{\min}^{i,k} \\
&+ \delta_{\deg^G(u_i) > \deg^H(v_k)}(\Delta_{\max}^{i,k} - \Delta_{\min}^{i,k})c_{\min}^i \\
&+ \delta_{\deg^G(u_i) < \deg^H(v_k)}(\Delta_{\max}^{i,k} - \Delta_{\min}^{i,k})c_{\min}^k] \\
c_{i,|V^H|+1} := c_V(u_i,\varepsilon) &+ 0.5 \cdot \deg^G(u_i) c_{\min}^i \\
c_{|V^G|+1,k} := c_V(\varepsilon,v_k) &+ 0.5 \cdot \deg^H(v_k) c_{\min}^k
\end{aligned}$$

for all $(i,k) \in [|V^G|] \times [|V^H|]$, where $\Delta_{\min}^{i,k} := \min\{\deg^G(u_i), \deg^H(v_i)\}$ and $\Delta_{\max}^{i,k} := \max\{\deg^G(u_i), \deg^H(v_i)\}$.

By sorting all sets of incident edge labels before populating $\mathbf{C}$, BRANCH-FAST can reduce the time complexity of constructing $\mathbf{C}$ to $O(\max\{|V^G|,|V^H|\}\Delta_{\max}^{G,H}\log(\Delta_{\max}^{G,H}) + |V^G||V^H|\Delta_{\min}^{G,H}\Delta_{\max}^{G,H})$. As ineq. (2) with $\xi :\equiv 1$ holds for each input, BRANCH-FAST returns an upper and a lower bound for GED. Like the lower bound produced by BRANCH, the lower bound yielded by BRANCH-FAST is a pseudo-metric if the underlying edit costs are metric. Furthermore, it can be shown that BRANCH-FAST's lower bound is never tighter than the one computed by BRANCH. For constant edge edit costs, BRANCH and BRANCH-FAST are equivalent.

5.2.5 The algorithm BRANCH-CONST

The algorithm BRANCH-CONST [70, 71] can be viewed as a speed-up of BRANCH and BRANCH-FAST for constant edge edit costs c_E.[1] It uses the fact that, if the edge edit costs are constant, the minimum edge deletion, insertion, and substitution costs employed by BRANCH-FAST do not have to be computed, as we have $c_{\min}^i = c_E^{\text{del}}$, $c_{\min}^k = c_E^{\text{ins}}$, and $c_{\min}^{i,k} = c_E^{\text{sub}}$. This implies that the LSAPE instance $\mathbf{C}$ can be constructed in $O(\max\{|V^G|,|V^H|\}\Delta_{\max}^{G,H}\log(\Delta_{\max}^{G,H}) + |V^G||V^H|\Delta_{\min}^{G,H})$ time. All other properties are inherited from BRANCH and BRANCH-FAST.

5.2.6 The algorithm STAR

The algorithm STAR [68] considers the neighbors of the nodes $u_i \in V^G$ and $v_k \in V^H$ when populating the cell $c_{i,k}$ of its LSAPE instance $\mathbf{C}$. It requires uniform edit cost functions c_V and c_E and ignores the edge labels of the input graphs. Let C be the constant such that $c_V(\alpha,\alpha') = c_E(\beta,\beta') = C$ holds for all $(\alpha,\alpha') \in (\Sigma_V \cup \{\varepsilon\}) \times (\Sigma_V \cup \{\varepsilon\})$ with $\alpha \neq \alpha'$ and all $(\beta,\beta') \in (\Sigma_E \cup \{\varepsilon\}) \times (\Sigma_E \cup \{\varepsilon\})$ with $\beta \neq \beta'$. In a first step, STAR computes $\Gamma_V^{i,k}$ as the size of the intersection of the multisets of node labels that are adjacent to u_i in G and to v_k in H, i. e., sets $\Gamma_V^{i,k} := |\ell_V^G[\![N^G(u_i)]\!] \cap \ell_V^H[\![N^H(v_k)]\!]|$. STAR then defines its LSAPE instance $\mathbf{C}$ by setting

$$\begin{aligned}
c_{i,k} &:= C \cdot [\delta_{\ell_V^G(u_i) \neq \ell_V^H(v_k)} + 2\Delta_{\max}^{i,k} - \Delta_{\min}^{i,k}) - \Gamma_V^{i,k}] \\
c_{i,|V^H|+1} &:= C \cdot [1 + 2\deg^G(u_i)] \\
c_{|V^G|+1,k} &:= C \cdot [1 + 2\deg^H(v_k)]
\end{aligned}$$

for all $(i,k) \in [|V^G|] \times [|V^H|]$, where $\Delta_{\min}^{i,k}$ and $\Delta_{\min}^{i,k}$ are defined as in Section 5.2.4.

STAR has the same time complexity as BRANCH-CONST, that is, STAR requires $O(\max\{|V^G|,|V^H|\}\Delta_{\max}^{G,H}\log(\Delta_{\max}^{G,H}) + |V^G||V^H|\Delta_{\min}^{G,H})$ time for constructing its LSAPE instance $\mathbf{C}$. Furthermore, ineq. (2) holds if the scaling function ξ

[1] As BRANCH-CONST was proposed before BRANCH and BRANCH-FAST, it is in fact more correct to say that BRANCH and BRANCH-FAST generalize BRANCH-CONST to arbitrary edit costs. For the sake of simplicity, we here change the order of presentation.

is defined as $\xi(G,H,c_V,c_E) := 1/\max\{4,\Delta^{G,H}_{\max}+1\}$. STAR hence returns both a lower and an upper bound for GED.

5.2.7 The algorithm SUBGRAPH

The algorithm SUBGRAPH [21] considers more global information than the previously presented heuristics for constructing its LSAPE instance $\mathbf{C}$. Given a constant $K \in \mathbb{N}_{\geq 1}$, SUBGRAPH constructs graphlets $G_i := G[\bigcup_{s=0}^{K} N^G_s(u_i)]$ and $H_k := H[\bigcup_{s=0}^{K} N^H_s(v_k)]$ for all $(i,k) \in [|V^G|] \times [|V^H|]$, i. e., associates all nodes in the input graphs to the subgraphs which are induced by the sets of all nodes that are at distance at most K. For graphlets G_i and H_k, SUBGRAPH defines $\mathrm{GED}_{i,k}(G_i,H_k) := \min\{c(P_\pi) \mid \pi \in \Pi(G_i,H_i) \wedge \pi(u_i) = v_k\}$ as the edit distance under the restriction that G^i's root node u_i be mapped to H^k's root node v_k. SUBGRAPH then constructs its LSAPE instance $\mathbf{C}$ by setting

$$\begin{aligned} c_{i,k} &:= \mathrm{GED}_{i,k}(G_i,H_k) \\ c_{i,|V^H|+1} &:= \mathrm{GED}(G_i,\mathscr{E}) \\ c_{|V^G|+1,k} &:= \mathrm{GED}(\mathscr{E},H_k) \end{aligned}$$

for all $(i,k) \in [|V^G|] \times [|V^H|]$, where $\mathscr{E}$ denotes the empty graph.

The time complexity of SUBGRAPH's construction phase of its LSAPE instance $\mathbf{C}$ is exponential in $\Delta^{G,H}_{\max}$. This implies that, unless $\max\deg(G)$ and $\max\deg(H)$ are constantly bounded, SUBGRAPH does not run in polynomial time. SUBGRAPH only computes an upper bound for GED.

5.2.8 The algorithm WALKS

The algorithm WALKS [32] requires constant and symmetric edit cost functions c_V and c_E and aims at computing a tight upper bound for GED by associating each node in the input graphs to the set of walks of size K that start at this node. Given constant $K \in \mathbb{N}_{\geq 1}$, a node $u_i \in V^G$, and a node $v_k \in V^H$, WALKS defines $\mathscr{W}^G_i$ and $\mathscr{W}^H_k$ as, respectively, the sets of walks of size K that start at u_i and v_k. Walks $W \in \mathscr{W}^G_i$ and $W' \in \mathscr{W}^H_k$ are called *similar* if they encode the same sequences of node and edge labels. Otherwise, W and W' are called *different*.

WALKS now computes the matrix products $\mathbf{W}^K_G$, $\mathbf{W}^K_H$, and $\mathbf{W}^K_\times$, where $\mathbf{W}_G$ is the adjacency matrix of G, $\mathbf{W}_H$ is the adjacency matrix of H, and $\mathbf{W}_\times$ is the adjacency matrix of the direct product graph $G \times H$ of G and H. $G \times H$ contains a node (u_i,v_k) for each $(i,k) \in [|V^G|] \times [|V^H|]$ such that $\ell^G_V(u_i) = \ell^H_V(v_k)$. Two nodes (u_i,v_k) and (u_j,v_l) of the product graph $G \times H$ are connected by an edge if and only if $(u_i,u_j) \in E^G$, $(v_k,v_l) \in E^H$, and $\ell^G_E(u_i,u_j) = \ell^H_E(v_k,v_l)$.

With the help of $\mathbf{W}^K_G$, $\mathbf{W}^K_H$, and $\mathbf{W}^K_\times$, for each node label $\alpha \in \Sigma_V$, WALKS computes an estimate $\widehat{h}_{i\setminus k}(\alpha)$ of the number of walks $W \in \mathscr{W}^G_i$ that end at a node with label α and must be substituted by a different walk $W' \in \mathscr{W}^H_k$. Analogously, $\widehat{h}_{k\setminus i}(\alpha)$ is computed as an estimate of the number of walks $W' \in \mathscr{W}^H_k$ that end at a node with label α and must be substituted by a different walk $W \in \mathscr{W}^G_i$. Moreover, WALKS computes an estimate $\widehat{r}_{i\setminus k} := \sum_{\alpha\in\Sigma_V} \widehat{h}_{i\setminus k}(\alpha) - \min\{\widehat{h}_{i\setminus k}(\alpha),\widehat{h}_{k\setminus i}(\alpha)\}$ of the number of walks in $W \in \mathscr{W}^G_i$ that must be substituted by a different walk $W' \in \mathscr{W}^H_i$ that does not end at the same node label, and an estimate $\widehat{r}_{k\setminus i} := \sum_{\alpha\in\Sigma_V} \widehat{h}_{k\setminus i}(\alpha) - \min\{\widehat{h}_{i\setminus k}(\alpha),\widehat{h}_{k\setminus i}(\alpha)\}$ of the number of walks in $W' \in \mathscr{W}^H_k$ that must be substituted by a different walk $W \in \mathscr{W}^G_i$ that does not end at the same node label. With these ingredients, WALKS constructs its LSAPE instance $\mathbf{C}$ by setting

$$\begin{aligned} c_{i,k} &:= [(\delta_{\ell^G_V(u_i)\neq\ell^H_V(v_k)} + K - 1)c^{\mathrm{sub}}_V \\ &\quad + Kc^{\mathrm{sub}}_E] \cdot \textstyle\sum_{\alpha\in\Sigma_V} \min\{\widehat{h}_{i\setminus k}(\alpha),\widehat{h}_{k\setminus i}(\alpha)\} \\ &\quad + [(\delta_{\ell^G_V(u_i)\neq\ell^H_V(v_k)} + K)c^{\mathrm{sub}}_V \\ &\quad + Kc^{\mathrm{sub}}_E] \cdot \min\{\widehat{r}_{i\setminus k},\widehat{r}_{k\setminus i}\} \\ &\quad + [(\delta_{\ell^G_V(u_i)\neq\ell^H_V(v_k)} + K)c^{\mathrm{del}}_V + Kc^{\mathrm{del}}_E] \cdot |\widehat{r}_{i\setminus k} - \widehat{r}_{k\setminus i}| \\ c_{i,|V^H|+1} &:= [(\delta_{\ell^G_V(u_i)\neq\ell^H_V(v_k)} + K)c^{\mathrm{del}}_V + Kc^{\mathrm{del}}_E] \cdot |\mathscr{W}^G_i| \\ c_{|V^G|+1,k} &:= [(\delta_{\ell^G_V(u_i)\neq\ell^H_V(v_k)} + K)c^{\mathrm{del}}_V + Kc^{\mathrm{del}}_E] \cdot |\mathscr{W}^H_k| \end{aligned}$$

for all $(i,k) \in [|V^G|] \times [|V^H|]$.

WALKS requires $O((|V^G||V^H|)^\omega)$ time for computing its LSAPE instance $\mathbf{C}$, where $O(n^\omega)$ is the complexity of multiplying two matrices with n rows and n columns. The asymptotically fastest matrix multiplication algorithms achieve $\omega < 2.38$ [40]; the fastest practically useful matrix multiplication algorithm runs in $O(n^{\log_2(7)}) \approx O(n^{2.81})$ time [63]. WALKS only computes an upper bound for GED.

5.2.9 The algorithm RING

Like SUBGRAPH and WALKS, the algorithm RING [4, 6] aims at computing a tight upper bound for GED by considering enlarged local structures. Given constant $K \in \mathbb{N}_{\geq 1}$, a node $u_i \in V^G$, and a node $v_k \in V^H$, RING uses breadth-first search to construct rings $\mathscr{R}^G_i := (\mathscr{L}^G_l(u_i))^{K-1}_{l=0}$ and $\mathscr{R}^H_k := (\mathscr{L}^H_l(v_k))^{K-1}_{l=0}$ whose lth layers are defined as the triplets $\mathscr{L}^G_l(u_i) := (N^G_l(u_i),OE^G_l(u_i),IE^G_l(u_i))$ and $\mathscr{L}^H_l(v_k) := (N^H_l(v_k),OE^H_l(v_k),IE^H_l(v_k))$, respectively. $OE^G_l(u_i) := E^G \cap (N^G_l(u_i) \times N^G_{l+1}(u_i))$ is defined the set of edges from G that connect nodes at distance l from u_i to nodes at distance $l+1$, while the set $IE^G_l(u_i) := E^G \cap (N^G_l(u_i) \times N^G_l(u_i))$ contains all edges that connect two nodes at distance l. The edge sets $OE^H_l(v_k)$ and $IE^H_l(v_k)$ are defined analogously.

In the next step, RING defined ring distances $d_{\mathscr{R}}(\mathscr{R}^G_i,\mathscr{R}^H_k) := \sum_{l=0}^{K-1} \lambda_l d_{\mathscr{L}}(\mathscr{L}^G_l(u_i),\mathscr{L}^H_l(v_k)$, where the layer distance $d_{\mathscr{L}}$ is defined as $d_{\mathscr{L}}(\mathscr{L}^G_l(u_i),\mathscr{L}^H_l(v_k) := \alpha_0 d_V(N^G_l(u_i),N^H_l(v_k)) + \alpha_1 d_E(OE^G_l(u_i),OE^H_l(v_k)) + \alpha_2 d_E(IE^G_l(u_i),IE^H_l(v_k))$, $(\lambda_l)_l \in \mathbb{R}^K_{\geq 0}$ and $(\alpha_s)_s \in \mathbb{R}^3_{\geq 0}$ are simplex vectors, d_V is a distance measure between node sets, and d_E is a distance measure between edge sets. RING suggests a strategy that uses a blackbox optimization for intelligently choosing the meta-parameters λ_l, α_s, and K.

Three different definitions of the node set distance d_V are suggested. The first proposal is to use the node edit cost function c_V to construct an auxiliary LSAPE instance $\mathbf{C}^{i,k}$ for the node sets $N_l^G(u_i)$ and $N_l^H(v_k)$, and then to define $d_V(N_l^G(u_i), N_l^H(v_k)) := \text{LSAPE}(\mathbf{C}^{i,k})$. Alternatively, it is proposed to define $d_V(N_l^G(u_i), N_l^H(v_k)) := \overline{\text{LSAPE}(\mathbf{C}^{i,k})}$, where $\overline{\text{LSAPE}(\mathbf{C}^{i,k})}$ is a proxy for $\text{LSAPE}(\mathbf{C}^{i,k})$ which is efficiently computed via greedy LSAPE solvers or fast heuristics based on multiset intersection. The edge set distance measure d_E can be defined analogously. RING then constructs its LSAPE instance $\mathbf{C}$ by setting

$$\begin{aligned} c_{i,k} &:= d_{\mathscr{R}}(\mathscr{R}_i^G, \mathscr{R}_k^H) \\ c_{i,|V^H|+1} &:= d_{\mathscr{R}}(\mathscr{R}_i^G, \mathscr{E}) \\ c_{|V^G|+1,k} &:= d_{\mathscr{R}}(\mathscr{E}, \mathscr{R}_k^H) \end{aligned}$$

for all $(i,k) \in [|V^G|] \times [|V^H|]$, where $\mathscr{E} := (\emptyset, \emptyset, \emptyset)_{l=0}^{K-1}$ is the empty ring of size K.

Constructing all rings for a graph G requires $O(|V^G|(|V^G| + |E^G|))$ time. Let Ω be the maximum size of a node or edge set that appears in one of the rings rooted at the nodes of G and H. Then, once all rings for G and H have been constructed, RING populates its LSAPE instance $\mathbf{C}$ in $O(|V^G||V^H|\Omega^3)$ time if an optimal LSAPE solver is used for computing d_V and d_E, and in $O(|V^G||V^H|\Omega^2)$ time if greedy or multiset intersection based heuristics are employed. RING only computes an upper bound for GED.

5.2.10 The algorithm RING-ML

The algorithm RING-ML [6] is similar to RING in that it also decomposes the input graphs into rings rooted at their nodes. However, instead of computing distances between the rings, RING-ML constructs a feature vectors $\mathbf{x}^{i,k} := (\mathbf{x}^{G,H}, \mathbf{x}^0, \ldots, \mathbf{x}^l, \ldots, \mathbf{x}^{K-1}) \in \mathbb{R}^{6K+10}$ for all $(i,k) \in [|V^G|+1] \times [|V^H|+1]$. The features contained in $\mathbf{x}^l \in \mathbb{R}^6$ express the dissimilarity of the lth layers of the rings $\mathscr{R}_i^G$ and $\mathscr{R}_k^H$. They are defined as $\mathbf{x}_0^l := |N_l^G(u_i)| - |N_l^H(v_k)|$, $\mathbf{x}_1^l := |OE_l^G(u_i)| - |OE_l^H(v_k)|$, $\mathbf{x}_2^l := |IE_l^G(u_i)| - |IE_l^H(v_k)|$, $\mathbf{x}_3^l := d_V(N_l^G(u_i), N_l^H(v_k))$, $\mathbf{x}_4^l := d_E(OE_l^G(u_i), OE_l^H(v_k))$, and $\mathbf{x}_5^l := d_E(IE_l^G(u_i), IE_l^H(v_k))$, where $u_{|V^G|+1} := v_{|V^H|+1} := \varepsilon$. The vector $\mathbf{x}^{G,H}$ contains global features that are the same for all $(i,k) \in [|V^G|] \times [|V^H|]$: the number of nodes and edges of G and H, the average costs for deleting nodes and edges from G, the average costs for inserting nodes and edges into H, and the average costs for substituting nodes and edges in G by nodes and edges in H.

Given a training set, RING-ML defines a node assignment $(u,v) \in (V^G \cup \{\varepsilon\}) \times (V^H \cup \{\varepsilon\})$ as *good* if and only if there is a node map $\pi \in \Pi(G,H)$ with $c(P_\pi) = \text{GED}(G,H)$ and $(u,v) \in \pi$. Next, RING-ML learns a function $p^\star$ that estimates the probability that a node assignment is good. This is done by computing optimal or close-to-optimal node maps between all graphs contained in a training set and then training a support vector classifier with probability estimates and RBF kernel, a one-class support vector machine with RBF kernel, or a fully connected, feedforward neural network on the generated training data. Once the probability estimate $p^\star$ has been learned, RING-ML constructs its LSAPE instance $\mathbf{C}$ by setting

$$\begin{aligned} c_{i,k} &:= 1 - p^\star(\mathbf{x}^{i,k}) \\ c_{i,|V^H|+1} &:= 1 - p^\star(\mathbf{x}^{i,|V^H|+1}) \\ c_{|V^G|+1,k} &:= 1 - p^\star(\mathbf{x}^{|V^G|+1,k}) \end{aligned}$$

for all $(i,k) \in [|V^G|] \times [|V^H|]$.

Depending on the choice of the node and edge set distances d_V and d_E, RING-ML requires $O(|V^G||V^H|\Omega^3)$ or $O(|V^G||V^H|\Omega^2)$ time for constructing all feature vectors for the graphs G and H. The time complexity of populating $\mathbf{C}$ once the feature vectors have been constructed depends on the employed machine learning technique. RING-ML only computes an upper bound for GED.

5.2.11 The algorithm PREDICT

The algorithm PREDICT [6, 54] differs from RING-ML only in that it uses different feature vectors $\mathbf{x}^{i,k}$. For computing its feature vectors, PREDICT first constructs an auxiliary LSAPE instance $\mathbf{C}^{\text{BP}}$ as done by the algorithm BP presented in Section 5.2.2. Subsequently, for all $(i,k) \in [|V^G|+1] \times [|V^H|+1]$, PREDICT defines $\mathbf{x}^{i,k} := (\mathbf{x}^{G,H}, \mathbf{x}^i, \mathbf{x}^k, c_V(u_i,v_k), c_{i,k}^{\text{BP}} - c_V(u_i,v_k)) \in \mathbb{R}^{24}$, where $u_{|V^G|+1} := v_{|V^H|+1} := \varepsilon$. The vector $\mathbf{x}^{G,H} \in \mathbb{R}^4$ contains four global features: the maximum, the minimum, the average, and the deviation of $\mathbf{C}^{\text{BP}}$. The vector $\mathbf{x}^i \in \mathbb{R}^9$ contains nine features associated to the ith row of $\mathbf{C}^{\text{BP}}$: its maximum, its minimum, its average, its deviation, its uniqueness, its divergence, its leader, its interval, and its outlierness. Analogously, $\mathbf{x}^k \in \mathbb{R}^9$ contains nine features associated to the kth row of $\mathbf{C}^{\text{BP}}$. Finally, $\mathbf{x}^{i,k}$ contains features for the node and the edge edit costs which are induced by assigning u_i to v_k. Once all feature vectors have been constructed, PREDICT proceeds exactly like RING-ML.

5.3 Extensions of the paradigm LSAPE-GED

Next, we present two extensions of the paradigm LSAPE-GED. Both of them can be used to improve all of the heuristics described in Section 5.2 above.

5.3.1 The extension CENTRALITIES

Assume that an LSAPE instance $\mathbf{C} \in \mathbb{R}^{(|V^G|+1)\times(|V^H|+1)}$ has been constructed by one of the instantiations of LSAPE-GED presented in Section 5.2 above. In [25,53], it is suggested to define a node centrality measure ϕ that maps central nodes to large and non-central nodes to small non-negative reals. Suggested centrality measures are, for instance, the degrees, the eigenvector centralities [11], and the pagerank centralities [18] of the nodes of the input graphs.

With the help of ϕ, the upper bound for GED induced by $\mathbf{C}$ can be improved. To this purpose, a second LSAPE instance $\mathbf{C}' \in \mathbb{R}^{(|V^G|+1)\times(|V^H|+1)}$ is constructed by setting

$$\begin{aligned}
c'_{i,k} &:= (1-\gamma)\cdot c_{i,k} + \gamma\cdot|\phi(u_i)-\phi(v_k)| \\
c'_{i,|V^H|+1} &:= (1-\gamma)\cdot c_{i,|V^H|+1} + \gamma\cdot\phi(u_i) \\
c'_{|V^G|+1,k} &:= (1-\gamma)\cdot c_{|V^G|+1,k} + \gamma\cdot\phi(v_k)
\end{aligned}$$

for all $(i,k)\in[|V^G|]\times[|V^H|]$, where $0\leq\gamma\leq 1$ is a meta-parameter. Subsequently, two cheap or optimal LSAPE solutions $\pi,\pi'\in\Pi(|V^G|,|V^H|)$ for $\mathbf{C}$ and $\mathbf{C}'$ are computed, and the returned upper bound for GED is improved from $UB := c(P_\pi)$ to $UB := \min\{c(P_\pi), c(P_{\pi'})\}$.[2]

5.3.2 The extension MULTI-SOL

Given a constant $K\in\mathbb{N}_{\geq 1}$, the extension `MULTI-SOL` of the paradigm `LSAPE-GED` suggested in [6,26] improves the upper bound returned by instantiations of `LSAPE-GED` by considering not only one, but rather up to K optimal LSAPE solutions. `MULTI-SOL` cannot be used in combination with greedy LSAPE solvers, as it requires that the LSAPE instance $\mathbf{C}$ is solved optimally in line 2 of Algorithm 1. Once the first optimal LSAPE solution $\pi_0^\star\in\Pi^\star(\mathbf{C})$ has been computed, `MULTI-SOL` uses a variant of the algorithm suggested in [64] for enumerating $K' := \min\{K, |\Pi^\star(\mathbf{C})|\}-1$ optimal LSAPE solutions $\{\pi_l^\star\}_{l=1}^{K'}$, all of which are pairwise different and different from $\pi_0^\star$. Since K is a constant, this enumeration requires only $O(|V^G|+|V^H|)$ additional time. Subsequently, the upper bound for GED is improved from $UB := c(P_{\pi_0^\star})$ to $UB := \min_{l=0}^{K'} c(P_{\pi_l^\star})$.

6 Heuristics based on linear programming

In this section, we first introduce the paradigm `LP-GED`, which generalizes heuristics that use linear programming for lower and upper bounding GED (Section 6.1). Subsequently, we present heuristics that can be modeled as instantiations of `LP-GED` (Section 6.2).

[2] In the original publications, this technique is suggested for the LSAPE instance produced by `BP` (cf. Section 5.2.2). It can, however, be employed in combination with the LSAPE instances produced by any instantiation of `LSAPE-GED`.

$$\begin{aligned}
\min\ & \sum_{u_i\in V^G}\sum_{v_k\in V^H} c_V(u_i,v_k)x^{\text{sub}}_{i,k} \\
& + \sum_{u_i\in V^G} c_V(u_i,\varepsilon)x^{\text{del}}_i + \sum_{v_k\in V^H} c_V(\varepsilon,v_k)x^{\text{ins}}_k \\
& + \sum_{(u_i,u_j)\in E^G}\sum_{(v_k,v_l)\in E^H} c_E((u_i,u_j),(v_k,v_l))y^{\text{sub}}_{i,j,k,l} \\
& + \sum_{(u_i,u_j)\in E^G} c_E((u_i,u_j),\varepsilon)y^{\text{del}}_{i,j} + \sum_{(v_k,v_l)\in E^H} c_E(\varepsilon,(v_k,v_l))y^{\text{ins}}_{k,l} \\
\text{s.t.}\ & x^{\text{del}}_i + \sum_{v_k\in V^H} x^{\text{sub}}_{i,k} = 1 && \forall u_i\in V^G \\
& x^{\text{ins}}_k + \sum_{u_i\in V^G} x^{\text{sub}}_{i,k} = 1 && \forall v_k\in V^H \\
& y^{\text{del}}_{i,j} + \sum_{(v_k,v_l)\in E^H} y^{\text{sub}}_{i,j,k,l} = 1 && \forall (u_i,u_j)\in E^G \\
& y^{\text{ins}}_{k,l} + \sum_{(u_i,u_j)\in E^G} y^{\text{sub}}_{i,j,k,l} = 1 && \forall (v_k,v_l)\in E^H \\
& y^{\text{sub}}_{i,j,k,l} - x^{\text{sub}}_{i,k}x^{\text{sub}}_{j,l} - x^{\text{sub}}_{i,l}x^{\text{sub}}_{j,k} = 0\ \forall((v_k,v_l),(v_k,v_l))\in E^G\times E^H \\
& \mathbf{x}^{\text{sub}}\in\{0,1\}^{|V^G|\times|V^H|},\ \mathbf{x}^{\text{del}}\in\{0,1\}^{|V^G|},\ \mathbf{x}^{\text{ins}}\in\{0,1\}^{|V^H|} \\
& \mathbf{y}^{\text{sub}}\in\{0,1\}^{|E^G|\times|E^H|},\ \mathbf{y}^{\text{del}}\in\{0,1\}^{|E^G|},\ \mathbf{y}^{\text{ins}}\in\{0,1\}^{|E^H|}
\end{aligned}$$

Fig. 4 A quadratic programming formulation of GED.

6.1 The paradigm LP-GED

Recall the alternative Definition 4 of GED, which defines the problem of computing GED as a minimization problem over the set of all node maps between two graphs G and H. This definition can straightforwardly be transformed into the quadratic programming formulation of GED detailed in Figure 4 [16,49]. The binary decision variables $x^{\text{sub}}_{i,k}$, x^{del}_i, and x^{ins}_k indicate, respectively, whether the node $u_i\in V^G$ is to be substituted by the node $v_k\in V^H$, whether u_i is to be deleted, and whether v_k is to be inserted. Analogously, the binary decision variables $y^{\text{sub}}_{i,j,k,l}$, $y^{\text{del}}_{i,j}$, and $y^{\text{ins}}_{k,l}$ indicate, respectively, whether the edge $(u_i,u_j)\in E^G$ is to be substituted by the edge $(v_k,v_l)\in E^H$, whether (u_i,u_j) is to be deleted, and whether (v_k,v_l) is to be inserted. The quadratic constraint $y^{\text{sub}}_{i,j,k,l} - x^{\text{sub}}_{i,k}x^{\text{sub}}_{j,l} - x^{\text{sub}}_{i,l}x^{\text{sub}}_{j,k} = 0$ ensures that (u_i,u_j) is substituted by (v_k,v_l) if and only if the node map π encoded by $\mathbf{x}^{\text{sub}}$, $\mathbf{x}^{\text{del}}$, and $\mathbf{x}^{\text{ins}}$ satisfies $\pi(u_i,u_j) = (v_k,v_l)$.

Heuristics that use linear programs (LP) for upper and lower bounding GED proceed as described in Algorithm 2: In a first step, the quadratic program shown in Figure 4 is linearized to obtain a (mixed) integer linear programming (MIP) formulation $\widehat{F}$ (line 1). This linearization phase is where different instantiations of `LP-GED` vary from each other. Next, all integrality constraints contained in $\widehat{F}$ are continuously relaxed, which yields in an LP F (line 2). Subsequently, F is solved and the lower bound LB is set to the optimal solution of F.

Algorithm 2 The paradigm LP-GED.

```
Input: Graphs $G$ and $H$, node edit costs $c_V$, edge edit costs $c_E$.
Output: An upper bound $UB$ and a lower bound $LB$ for $\mathrm{GED}(G,H)$.
1: construct linearization of the quadratic programming formulation
   detailed in Figure 4;
2: relax all integrality constraints of the obtained MIP;
3: solve the resulting LP and set $LB$ to the obtained minimum;
4: use optimal continuous solution to construct projection problem
   $\mathbf{C} \in [0,1]^{(|V^G|+1)\times(|V^H|+1)}$;
5: compute $\pi^\star \in \arg\min_{\pi\in\Pi(|V^G|,|V^H|)} \mathbf{C}(\pi)$;
6: set $UB := c(P_{\pi^\star})$;
7: return $LB$ and $UB$;
```

In the literature, LP based heuristics for GED are usually described as algorithms that only yield lower bounds. However, they can straightforwardly be extended to also compute upper bounds. To that purpose, after solving the LP F, an LSAPE instance $\mathbf{C} \in \mathbb{R}^{(|V^G|+1)\times(|V^H|+1)}$ is constructed, whose optimal solutions $\pi^\star \in \arg\min_{\pi\in\Pi(|V^G|,|V^H|)} \mathbf{C}(\pi)$ can be viewed as projections of the previously computed optimal and possibly continuous solution for F to the discrete domain (line 4). Subsequently, an optimal solution $\pi^\star$ for $\mathbf{C}$ is computed (line 5), the upper bound UB is set to its induced edit cost (line 6), and LB and UB are returned (line 7).

In theory, the LP F can be solved in $O(\mathrm{var}(F)^{3.5}\,\mathrm{enc}(F))$ time, where $\mathrm{var}(F)$ is the number of variables contained in F and $\mathrm{enc}(F)$ is the number of bits needed to encode F [37]. However, popular LP solvers such as IBM CPLEX or Gurobi Optimization often use asymptotically slower but practically faster algorithms. Solving the projection problem $\mathbf{C}$ requires $O(\min\{|V^G|,|V^H|\}^2 \max\{|V^G|,|V^H|\})$ time.

6.2 Instantiations of the paradigm LP-GED

We present four different linearizations of the quadratic program shown in Figure 4. The first three linearizations presented in Sections 6.2.1 to 6.2.3 are designed for general graphs and edit costs. They hence not only yield upper and lower bounds for GED as detailed in Algorithm 2, but also exact algorithms if fed into exact MIP solvers. The last linearization presented in Section 6.2.4 requires the edge edit costs to be constant and symmetric. Furthermore, it is designed for graphs without edge labels. If used with graphs whose edges are labeled, it hence does not yield an exact algorithm for GED, even if fed into an exact MIP solver.

6.2.1 The linearization F1

The integer linear program F1 [43, 44] and displayed in Figure 5 is a straightforward linearization of the quadratic programming formulation shown in Figure 4. F1 has $|V^G||V^H| + |V^G| + |V^H| + |E^G||E^H| + |E^G| + |E^H| = O(|E^G||E^H|)$ binary variables and $|V^G| + |V^H| + |E^G| + |E^H| + 2|E^G||E^H| = O(|E^G||E^H|)$ constraints. Given an optimal solution $(\mathbf{x}^{\mathrm{sub}}, \mathbf{x}^{\mathrm{del}}, \mathbf{x}^{\mathrm{ins}}, \mathbf{y}^{\mathrm{sub}}, \mathbf{y}^{\mathrm{del}}, \mathbf{y}^{\mathrm{ins}})$ for the continuous relaxation of F1, the projection problem $\mathbf{C}$ is defined as

$$
\begin{aligned}
c_{i,k} &:= 1 - x^{\mathrm{sub}}_{i,k}\\
c_{i,|V^H|+1} &:= 1 - x^{\mathrm{del}}_{i}\\
c_{|V^G|+1,k} &:= 1 - x^{\mathrm{ins}}_{k}
\end{aligned}
$$

for all $(i,k) \in [|V^G|] \times [|V^H|]$.

$$
\begin{aligned}
\min\ & \sum_{u_i\in V^G}\sum_{v_k\in V^H} c_V(u_i,v_k)x^{\mathrm{sub}}_{i,k}\\
&+ \sum_{u_i\in V^G} c_V(u_i,\varepsilon)x^{\mathrm{del}}_{i} + \sum_{v_k\in V^H} c_V(\varepsilon,v_k)x^{\mathrm{ins}}_{k}\\
&+ \sum_{(u_i,u_j)\in E^G}\sum_{(v_k,v_l)\in E^H} c_E((u_i,u_j),(v_k,v_l))y^{\mathrm{sub}}_{i,j,k,l}\\
&+ \sum_{(u_i,u_j)\in E^G} c_E((u_i,u_j),\varepsilon)y^{\mathrm{del}}_{i,j} + \sum_{(v_k,v_l)\in E^H} c_E(\varepsilon,(v_k,v_l))y^{\mathrm{ins}}_{k,l}\\
\text{s. t.}\quad & x^{\mathrm{del}}_{i} + \sum_{v_k\in V^H} x^{\mathrm{sub}}_{i,k} = 1 && \forall u_i \in V^G\\
& x^{\mathrm{ins}}_{k} + \sum_{u_i\in V^G} x^{\mathrm{sub}}_{i,k} = 1 && \forall v_k \in V^H\\
& y^{\mathrm{del}}_{i,j} + \sum_{(v_k,v_l)\in E^H} y^{\mathrm{sub}}_{i,j,k,l} = 1 && \forall (u_i,u_j) \in E^G\\
& y^{\mathrm{ins}}_{k,l} + \sum_{(u_i,u_j)\in E^G} y^{\mathrm{sub}}_{i,j,k,l} = 1 && \forall (v_k,v_l) \in E^H\\
& y^{\mathrm{sub}}_{i,j,k,l} - x^{\mathrm{sub}}_{i,k} - x^{\mathrm{sub}}_{k,l} \leq 0 && \forall ((v_k,v_l),(v_k,v_l)) \in E^G \times E^H\\
& y^{\mathrm{sub}}_{i,j,k,l} - x^{\mathrm{sub}}_{j,l} - x^{\mathrm{sub}}_{j,k} \leq 0 && \forall ((v_k,v_l),(v_k,v_l)) \in E^G \times E^H\\
& \mathbf{x}^{\mathrm{sub}} \in \{0,1\}^{|V^G|\times|V^H|},\ \mathbf{x}^{\mathrm{del}} \in \{0,1\}^{|V^G|},\ \mathbf{x}^{\mathrm{ins}} \in \{0,1\}^{|V^H|}\\
& \mathbf{y}^{\mathrm{sub}} \in \{0,1\}^{|E^G|\times|E^H|},\ \mathbf{y}^{\mathrm{del}} \in \{0,1\}^{|E^G|},\ \mathbf{y}^{\mathrm{ins}} \in \{0,1\}^{|E^H|}
\end{aligned}
$$

Fig. 5 The linearization F1 suggested in [43, 44].

6.2.2 The linearization F2

The linearization F2 [44] displayed in Figure 6 improves F1 by reducing the number of variables and constraints. It uses the fact that the node and edge substitution variables $\mathbf{x}^{\mathrm{sub}}$ and $\mathbf{y}^{\mathrm{sub}}$ implicitly encode the node and edge deletion and insertion variables $\mathbf{x}^{\mathrm{del}}$, $\mathbf{x}^{\mathrm{ins}}$, $\mathbf{y}^{\mathrm{del}}$, and $\mathbf{y}^{\mathrm{ins}}$. F2 has $|V^G| + |V^H| + |V^H||E^G| = O(|V^H||E^G|)$ constraints and $|V^G||V^H| + |E^G||E^H| = O(|E^G||E^H|)$ binary variables. Given an optimal solution $(\mathbf{x}^{\mathrm{sub}}, \mathbf{y}^{\mathrm{sub}})$ for the continuous relaxation of F2, the projection problem $\mathbf{C}$ is defined as

$$
\begin{aligned}
c_{i,k} &:= 1 - x^{\mathrm{sub}}_{i,k}\\
c_{i,|V^H|+1} &:= \sum_{v_k\in V^H} x^{\mathrm{sub}}_{i,k}\\
c_{|V^G|+1,k} &:= \sum_{u_i\in V^G} x^{\mathrm{sub}}_{i,k}
\end{aligned}
$$

for all $(i,k) \in [|V^G|] \times [|V^H|]$.

$$
\begin{aligned}
\min \sum_{u_i V^G} \sum_{v_k \in V^H} & c'_V(u_i, v_k) x^{\text{sub}}_{i,k} \\
& + \sum_{(u_i,u_j) \in E^G} \sum_{(v_k,v_l) \in E^H} c'_E((u_i,u_j),(v_k,v_l)) y^{\text{sub}}_{i,j,k,l} + C \\
\text{s.t.} \quad & \sum_{v_k \in V^H} x^{\text{sub}}_{i,k} \leq 1 \qquad \forall u_i \in V^G \\
& \sum_{u_i \in V^G} x^{\text{sub}}_{i,k} \leq 1 \qquad \forall v_k \in V^H \\
& \sum_{(v_k,v_l) \in E^H} y^{\text{sub}}_{i,j,k,l} - x^{\text{sub}}_{i,k} - x^{\text{sub}}_{j,k} \leq 0 \; \forall (v_k,(u_i,u_j)) \in V^H \times E^G \\
& \mathbf{x}^{\text{sub}} \in \{0,1\}^{|V^G| \times |V^H|}, \mathbf{y}^{\text{sub}} \in \{0,1\}^{|E^G| \times |E^H|}
\end{aligned}
$$

Fig. 6 The linearization F2 suggested in [44]. The modified edit costs c'_V and c'_E and the constant C are defined as $c'_V(u_i,v_k) := c_V(u_i,v_k) - c_V(u_i,\varepsilon) - c_V(\varepsilon,v_k)$, $c'_E((u_i,u_j),(v_k,v_l)) := c_E((u_i,u_j),(v_k,v_l)) - c_E((u_i,u_j),\varepsilon) - c_E(\varepsilon,(v_k,v_l))$, and $C := \sum_{u_i \in V^G} c_V(u_i,\varepsilon) + \sum_{v_k \in V^H} c_V(\varepsilon,v_k) + \sum_{(u_i,u_j) \in E^G} c_E((u_i,u_j),\varepsilon) + \sum_{(v_k,v_l) \in E^H} c_E(\varepsilon,(v_k,v_l))$.

$$
\begin{aligned}
\min \sum_{u_i \in V^G} \sum_{v_k \in V^H} z^{\text{sub}}_{i,k} + \sum_{u_i \in V^G} z^{\text{del}}_i + \sum_{v_k \in V^H} z^{\text{ins}}_k & \\
\text{s.t.} \quad x^{\text{del}}_i + \sum_{v_k \in V^H} x^{\text{sub}}_{i,k} = 1 & \qquad \forall u_i \in V^G \\
x^{\text{ins}}_k + \sum_{u_i \in V^G} x^{\text{sub}}_{i,k} = 1 & \qquad \forall v_k \in V^H \\
\sum_{u_j \in N^G(u_i)} \sum_{v_l \in N^H(v_k)} \frac{c_E((u_i,u_j),(v_k,v_l))}{2} x^{\text{sub}}_{j,l} & \\
+ \sum_{u_j \in N^G(u_i)} \sum_{v_l \notin N^H(v_k)} \frac{c_E((u_i,u_j),\varepsilon)}{2} x^{\text{sub}}_{j,l} & \\
+ \sum_{u_j \notin N^G(u_i)} \sum_{v_l \in N^H(v_k)} \frac{c_E(\varepsilon,(v_k,v_l))}{2} x^{\text{sub}}_{j,l} & \\
+ \sum_{u_j \in N^G(u_i)} \frac{c_E((u_i,u_j),\varepsilon)}{2} x^{\text{del}}_j & \\
+ \sum_{v_l \in N^H(v_k)} \frac{c_E(\varepsilon,(v_k,v_l))}{2} x^{\text{ins}}_l & \\
+ c_V(u_i,v_k) - a^{\text{sub}}_{i,k}(1 - x^{\text{sub}}_{i,k}) - z^{\text{sub}}_{i,k} \leq 0 \; \forall (u_i,v_k) \in V^G \times V^H & \\
\sum_{u_j \in N^G(u_i)} \sum_{v_l \in V^H} \frac{c_E((u_i,u_j),\varepsilon)}{2} x^{\text{sub}}_{j,l} & \\
+ \sum_{u_j \in N^G(u_i)} \frac{c_E((u_i,u_j),\varepsilon)}{2} x^{\text{del}}_j & \\
+ c_V(u_i,\varepsilon) - a^{\text{del}}_i(1 - x^{\text{del}}_i) - z^{\text{del}}_i \leq 0 & \qquad \forall u_i \in V^G \\
\sum_{u_j \in V^G} \sum_{v_l \in N^H(v_k)} \frac{c_E(\varepsilon,(v_k,v_l))}{2} x^{\text{sub}}_{j,l} & \\
+ \sum_{v_l \in N^H(v_k)} \frac{c_E(\varepsilon,(v_k,v_l))}{2} x^{\text{ins}}_l & \\
+ c_V(\varepsilon,v_k) - a^{\text{ins}}_k(1 - x^{\text{ins}}_k) - z^{\text{ins}}_k \leq 0 & \qquad \forall v_k \in V^H \\
\mathbf{x}^{\text{sub}} \in \{0,1\}^{|V^G| \times |V^H|}, \mathbf{x}^{\text{del}} \in \{0,1\}^{|V^G|}, \mathbf{x}^{\text{ins}} \in \{0,1\}^{|V^H|} & \\
\mathbf{z}^{\text{sub}} \in \mathbb{R}^{|V^G| \times |V^H|}_{\geq 0}, \mathbf{z}^{\text{del}} \in \mathbb{R}^{|V^G|}_{\geq 0}, \mathbf{z}^{\text{ins}} \in \mathbb{R}^{|E^H|}_{\geq 0} &
\end{aligned}
$$

Fig. 7 The linearization COMPACT-MIP suggested in [10]. For all $(u_i,v_k) \in V^G \times V^H$, the constants $a^{\text{sub}}_{i,k}$, a^{del}_i, and a^{ins}_k are defined as $a^{\text{sub}}_{i,k} := 0.5 \cdot [\sum_{u_j \in N^G(u_i)} \sum_{v_l \in N^H(v_k)} c_E((u_i,u_j),(v_k,v_l)) + \sum_{u_j \in N^G(u_i)} (|V^H| - \deg^H(v_k) + 1) c_E((u_i,u_j),\varepsilon) + \sum_{v_l \in N^H(v_k)} (|V^G| - \deg^G(u_i) + 1) c_E(\varepsilon,(v_k,v_l))]$, $a^{\text{del}}_i := 0.5 \cdot \sum_{u_j \in N^G(u_i)} (|V^H| + 1) c_E((u_i,u_j),\varepsilon)$, and $a^{\text{ins}}_k := 0.5 \cdot \sum_{v_l \in N^H(v_k)} (|V^G| + 1) c_E(\varepsilon,(v_k,v_l))$.

6.2.3 The linearization COMPACT-MIP

The linearization COMPACT-MIP [10] and displayed in Figure 7 makes do without edge variables. Instead, it contains continuous variables $\mathbf{z}^{\text{sub}}$, $\mathbf{z}^{\text{del}}$, $\mathbf{z}^{\text{ins}}$, which, at the optimum, contain the edit costs which are induced by the node assignment π encoded by optimal binary node variables $\mathbf{x}^{\text{sub}}$, $\mathbf{x}^{\text{del}}$, and $\mathbf{x}^{\text{ins}}$. COMPACT-MIP has $|V^G||V^H| + |V^G| + |V^H| = O(|V^G||V^H|)$ binary variables, $|V^G||V^H| + |V^G| + |V^H| = O(|V^G||V^H|)$ continuous variables, and $|V^G||V^H| + |V^G| + |V^H| = O(|V^G||V^H|)$ constraints. It is hence smaller than both F1 and F2. Given an optimal solution $(\mathbf{x}^{\text{sub}}, \mathbf{x}^{\text{del}}, \mathbf{x}^{\text{ins}}, \mathbf{z}^{\text{sub}}, \mathbf{z}^{\text{del}}, \mathbf{z}^{\text{ins}})$ for the continuous relaxation of COMPACT-MIP, the projection problem $\mathbf{C}$ is defined as

$$
\begin{aligned}
c_{i,k} &:= 1 - x^{\text{sub}}_{i,k} \\
c_{i,|V^H|+1} &:= 1 - x^{\text{del}}_i \\
c_{|V^G|+1,k} &:= 1 - x^{\text{ins}}_k
\end{aligned}
$$

for all $(i,k) \in [|V^G|] \times [|V^H|]$.

6.2.4 The linearization ADJ-IP

The linearization ADJ-IP [36] displayed in Figure 8 requires the edge edit costs c_E to be constant and symmetric. Furthermore, it is designed for graphs without edge labels. ADJ-IP has $3(|V^G| + |V^H|)^2 = O((|V^G| + |V^H|)^2)$ binary variables and $2|V^G| + 2|V^H| + |V^G||V^H| + |V^G|^2 + |V^H|^2 = O((|V^G| + |V^H|)^2)$ constraints. Note that ADJ-IP sets all edge substitution costs to 0. If used with graphs whose edges are labeled, it hence ignores all edit costs induced by substituting an edge $(u_i,u_j) \in E^G$ by an edge $(v_k,v_l) \in E^H$ with $\ell^G_E(u_i,u_j) \neq \ell^H_E(v_k,v_l)$. Given an optimal solution $(\mathbf{x},\mathbf{s},\mathbf{t})$ for the continuous relaxation of ADJ-IP, the projection problem $\mathbf{C}$ is defined as

$$
\begin{aligned}
c_{i,k} &:= 1 - x_{i,k} \\
c_{i,|V^H|+1} &:= \sum_{v_k \in V^H} x_{i,k} \\
c_{|V^G|+1,k} &:= \sum_{u_i \in V^G} x_{i,k}
\end{aligned}
$$

for all $(i,k) \in [|V^G|] \times [|V^H|]$.

$$\begin{aligned}
\min \sum_{u_i\in V^G}\sum_{v_k\in V^H} & [c_V(u_i,v_k)x_{i,k}+\frac{c_E^{\text{del}}}{2}(s_{i,k}+t_{i,k})]\\
+\sum_{u_i\in V^G}\sum_{u_j\in V^G} & [c_V(u_i,\varepsilon)x_{i,|V^H|+j}+\frac{c_E^{\text{del}}}{2}(s_{i,|V^H|+j}+t_{i,|V^H|+j})]\\
+\sum_{v_l\in V^H}\sum_{v_k\in V^H} & [c_V(\varepsilon,v_k)x_{|V^G|+l,k}+\frac{c_E^{\text{del}}}{2}(s_{|V^G|+l,k}+t_{|V^G|+l,k})]\\
\text{s.t.}\quad & \sum_{u_j\in V^G}x_{i,|V^H|+j}+\sum_{v_k\in V^H}x_{i,k}=1 && \forall u_i\in V^G\\
& \sum_{v_k\in V^H}x_{|V^G|+k,|V^H|+i}+\sum_{u_j\in V^G}x_{j,|V^H|+i}=1 && \forall u_i\in V^G\\
& \sum_{v_l\in V^H}x_{|V^G|+l,k}+\sum_{u_i\in V^G}x_{i,k}=1 && \forall v_k\in V^H\\
& \sum_{u_i\in V^G}x_{|V^G|+k,|V^H|+i}+\sum_{v_l\in V^H}x_{|V^G|+k,l}=1 && \forall v_k\in V^H\\
& s_{i,k}-t_{i,k}+\sum_{u_j\in N^G(u_i)}x_{j,k}-\sum_{v_l\in N^H(v_k)}x_{i,l}=0 && \forall(u_i,v_k)\in V^G\times V^H\\
& s_{i,|V^H|+j}-t_{i,|V^H|+j}+\sum_{u_r\in N^G(u_i)}x_{r,|V^H|+j}=0 && \forall(u_i,u_j)\in V^G\times V^G\\
& s_{|V^G|+l,k}-t_{|V^G|+l,k}-\sum_{v_s\in N^H(v_k)}x_{|V^G|+l,s}=0 && \forall(v_l,v_k)\in V^H\times V^H\\
& \mathbf{x},\mathbf{s},\mathbf{t}\in\{0,1\}^{(|V^G|+|V^H|)\times(|V^H|+|V^G|)}
\end{aligned}$$

Fig. 8 The linearization ADJ-IP suggested in [36].

7 Heuristics based on local search

In this section, we first introduce the paradigm LS-GED, which generalizes heuristics that use variants of local search for upper bounding GED (Section 7.1). Subsequently, we present heuristics that can be modeled as instantiations (Section 7.2) and extensions (Section 7.2) of LS-GED.

7.1 The paradigm LS-GED

Algorithm 3 shows how to compute an upper bound for GED via a variant of local search. In a first step, an initial node map $\pi \in \Pi(G,H)$ is generated randomly or constructed, for instance, by calling an instantiation of LSAPE-GED (line 1). Subsequently, a variant of local search is run, which produces an improved node map $\pi' \in \Pi(G,H)$ with $c(P_{\pi'}) \leq c(P_\pi)$ (line 2). This refinement phase is where different instantiations of LS-GED vary from each other. Once π' has been computed, $UB := c(P_{\pi'})$ is returned (lines 3 to 4).

7.2 Instantiations of the paradigm LS-GED

In the sequel, we present five algorithms for heuristically computing GED that can be modeled as instantiations of the paradigm LS-GED. All of them yield upper but no lower bounds for GED and can be used with general edit costs.

Algorithm 3 The paradigm LS-GED.

```
Input: Graphs $G$ and $H$, node edit costs $c_V$, edge edit costs $c_E$.
Output: An upper bound $UB$ for GED$(G,H)$.
1: compute or randomly construct initial node mao $\pi \in \Pi(G,H)$;
2: use information encoded in $G$, $H$, $c_V$, and $c_E$ to construct node map
   $\pi' \in \Pi(G,H)$ with $c(P_{\pi'}) \leq c(P_\pi)$ via local search starting at $\pi$;
3: set upper bound to $UB := c(P_{\pi'})$;
4: return $UB$;
```

7.2.1 The algorithm REFINE

Given an initial node map $\pi \in \Pi(G,H)$, the algorithm REFINE [12, 68] proceeds as follows: Let $((u_s,v_s))_{s=1}^{|\pi|}$ be an arbitrary ordering of the node assignments contained in π, let $u_{|\pi|+1} := v_{|\pi|+1} := \varepsilon$, and let $G_\pi := (V_\pi^G \cup V_\pi^H, A_\pi)$ be an auxiliary directed bipartite graph, where $V_\pi^G := \{u_s \mid s \in [|\pi|+1]\}$, $V_\pi^H := \{v_s \mid s \in [|\pi|+1]\}$, and $A_\pi := \pi \cup \{(u_{|\pi|+1}, v_{|\pi|+1})\} \cup \{(v_s, u_{s'}) \mid (s,s') \in [|\pi|+1] \times [|\pi|+1] \wedge s \neq s'\}$. In other words, G_π contains a forward arc for each assignment contained in π, a forward arc between the additionally added dummy nodes $u_{|\pi|+1}$ and $v_{|\pi|+1}$, and backward arcs between nodes in V_π^G and V_π^H that are not assigned to each other by π. Note that, by definition of a node map, V_π^G contains each node $u \in V^G$ exactly once, V_π^H contains each node $v \in V^H$ exactly once, but both V_π^G and V_π^H might contain multiple copies of the dummy node ε. A directed cycle $C \subseteq A_\pi$ in G_π with $|C| = 4$ is called swap. There are exactly $\binom{|\pi|+1}{2} = O((|V^G|+|V^H|)^2)$ swaps.

For each swap $C = \{(u_s,v_s),(v_s,u_{s'}),(u_{s'},v_{s'}),(v_{s'},u_s)\}$, REFINE checks if the swapped node map $\pi' := (\pi \setminus \{(u_s,v_s),(u_{s'},v_{s'})\}) \cup \{(u_s,v_{s'}),(u_{s'},v_s)\}$ induces a smaller upper bound than π. If, at the end of the for-loop, a node map π' has been found that improves the upper bound, π is updated to the node map that yields the largest improvement and the process iterates. Otherwise, the output node map π' is set to π and REFINE terminates.

For checking if a swap C improves the induced upper bound, it suffices to consider the edges that are incident with the nodes involved in the swap. Therefore, one iteration of REFINE runs in $O((|V^G|+|V^H|)^2\Delta_{\max}^{G,H})$ time, where $\Delta_{\max}^{G,H} := \max\{\max\deg(G), \max\deg(H)\}$. Since the induced upper bound improves in each iteration, this gives an overall runtime complexity of $O(UB(|V^G|+|V^H|)^2\Delta_{\max}^{G,H})$ for integral edit costs, where UB is the initial upper bound.

7.2.2 The algorithm K-REFINE

The algorithm K-REFINE [12] is a straightforward extension of the algorithm REFINE presented in the previous section. Let $\pi \in \Pi(G,H)$ be an initial node map and $K \in \mathbb{N}_{\geq 2}$ be a constant. Furthermore, let the auxiliary directed bipartite graph G_π be defined as in the previous section. A directed cycle $C \subseteq A_\pi$ in G_π with $|C| = 2K'$ is called swap of size K', where $K' \in \{2,\ldots,K\}$. There are exactly $\binom{|\pi|+1}{K'}(K'-1)! = O((|V^G|+|V^H|)^{K'})$ swaps of size K'.

Starting with $K' := 2$, K-REFINE checks if there is a swap of size K' that improves the induced upper bound. If so, π is updated to the node map obtained by the swap of size K' that yields the largest improvement, K' is reset to 2, and the process iterates. If no swap of size K' yields an improvement, K-REFINE checks whether K' equals the maximal swap size K. If this is the case, the output node map π' is set to π and K-REFINE terminates. Otherwise, K-REFINE increments K' and continues the search.

One iteration of K-REFINE runs in $O((|V^G|+|V^H|)^K \Delta^{G,H}_{\max})$ time. For integral edit costs, K-REFINE's overall runtime complexity is hence $O(UB(|V^G|+|V^H|)^K \Delta^{G,H}_{\max})$, where UB is the initial upper bound.

7.2.3 The algorithm BP-BEAM

Given an initial node map $\pi \in \Pi(G,H)$ and a constant $K \in \mathbb{N}_{\geq 1}$, the algorithm BP-BEAM [56] starts by producing a random ordering $((u_s,v_s))_{s=1}^{|\pi|}$ of the node assignments contained in π. BP-BEAM now constructs an output node map π' with $c(P_{\pi'}) \leq c(P_{\pi'})$ by partially traversing an implicitly constructed tree T via beam search with beam size K. The nodes of T are tuples $(\pi'', c(P_{\pi''}), s)$, where $\pi'' \in \Pi(G,H)$ is an ordered node map, $c(P_{\pi''})$ is its induced edit cost, and $s \in [|\pi|]$ is the depth of the tree node in T. Tree nodes $(\pi'', c(P_{\pi''}), s)$ with $s = |\pi|$ are leafs, and the children of an inner node $(\pi'', c(P_{\pi''}), s)$ are $\{(\mathrm{swap}(\pi'',s,s'), c(P_{\mathrm{swap}(\pi'',s,s')}), s+1) \mid s' \in \{s,\ldots,|\pi|\}\}$. Here, $\mathrm{swap}(\pi'',s,s')$ is the ordered node map obtained from π'' by swapping the assignments (u_s,v_s) and $(u_{s'},v_{s'})$, i. e., setting $v_s := v_{s'}$ and $v_{s'} := v_s$.

At initialization, BP-BEAM sets the output node map π' to the initial node map π. Furthermore, BP-BEAM maintains a priority queue q of tree nodes which is initialized as $q := \{(\pi, c(P_\pi), 1)\}$ and sorted w. r. t. non-decreasing induced edit cost of the contained node maps. As long as q is non-empty, BP-BEAM extracts the top node $(\pi'', c(P_{\pi''}), s)$ from q and updates the output node map π' to π'' if $c(P_{\pi''}) < c(P_{\pi'})$. If $s < |\pi|$, BP-BEAM adds all of its children to the priority queue q and subsequently discards all but the first K tree nodes contained in q. Once q is empty, the cheapest encountered node map π' is returned.

By construction of T, we know that at most $1 + K(|\pi|-1) = O(|V^G|+|V^H|)$ tree nodes are extracted from q. For each extracted inner node, BP-BEAM has to constructed all children, which requires $O((|V^G|+|V^H|)\Delta^{G,H}_{\max})$ time, and subsequently sort q, which requires $O((|V^G|+|V^H|)\log(|V^G|+|V^H|))$ time. BP-BEAM hence runs in $O((|V^G|+|V^H|)^2(\Delta^{G,H}_{\max}+\log(|V^G|+|V^H|)))$ time.

7.2.4 The algorithm IBP-BEAM

Since the size of the priority queue q is restricted to K, which parts of the search tree T are visited by BP-BEAM crucially depends on the ordering of the initial node map π. Therefore, BP-BEAM can be improved by considering not one but several initial orderings. The algorithm IBP-BEAM suggested in [27] does exactly this. That is, given a constant number of iterations $I \in \mathbb{N}_{\geq 1}$, IBP-BEAM runs BP-BEAM with I different randomly created orderings of the initial node map π and then returns the cheapest node map π' encountered in one of the iterations. Therefore, IBP-BEAM runs in $O(I(|V^G|+|V^H|)^2(\Delta^{G,H}_{\max}+\log(|V^G|+|V^H|)))$ time.

7.2.5 The algorithm IPFP

The algorithm IPFP [42] can be seen as an adaptation of the seminal Frank-Wolfe algorithm [30] to cases where an integer solution is required. Its adaptation to the case of GED, first suggested in [16], implicitly constructs a matrix $\mathbf{Q} \in \mathbb{R}^{((|V^G|+1)\cdot(|V^H|+1))\times((|V^G|+1)\cdot(|V^H|+1))}$ such that $\min_{\mathbf{X}\in\Pi(|V^G|,|V^H|)} \mathrm{vec}(\mathbf{X})^{\mathsf{T}}\mathbf{Q}\,\mathrm{vec}(\mathbf{X}) = \mathrm{GED}$. Recall that $\Pi(|V^G|,|V^H|) \in \{0,1\}^{(|V^G|+1)\times(|V^H|+1)}$ is the set of feasible LSAPE solutions of size $(|V^G|+1)\times(|V^H|+1)$, and that LSAPE solutions of size $(|V^G|+1)\times(|V^H|+1)$ are equivalent to node maps between G and H. $\mathbf{Q}$ can hence be viewed as a matrix representation of the quadratic program shown in Figure 4. In this context, we define the cost function $c : [0,1]^{(|V^G|+1)\cdot(|V^H|+1)} \to \mathbb{R}$ as $c(\mathbf{X}) := \mathrm{vec}(\mathbf{X})^{\mathsf{T}}\mathbf{Q}\,\mathrm{vec}(\mathbf{X})$.

Starting from an initial node map $\mathbf{X}_0 \in \Pi(|V^G|,|V^H|)$ with induced upper bound $UB := c(\mathbf{X}_0)$, the algorithm converges to a, possibly fractional, local minimum for GED by repeating the five following steps:

1. Populate LSAPE instance $\mathbf{C}_k := \mathbf{Q}\,\mathrm{vec}(\mathbf{X}_k)$.
2. Compute $\mathbf{B}_{k+1} \in \arg\min_{\mathbf{B}\in\Pi(|V^G|,|V^H|)} \mathbf{C}_k(\mathbf{B})$.
3. Set $UB := \min\{UB, c(\mathbf{B}_{k+1})\}$.
4. Compute $\alpha_{k+1} := \min_{\alpha\in[0,1]} c(\mathbf{X}_k + \alpha\cdot(\mathbf{B}_{k+1}-\mathbf{X}_k))$.
5. Set $\mathbf{X}_{k+1} := \mathbf{X}_k + \alpha_{k+1}(\mathbf{B}_{k+1}-\mathbf{X}_k)$.

The algorithm iterates until $|c(\mathbf{X}_k) - \mathbf{C}_k(\mathbf{B}_{k+1})|/c(\mathbf{X}_k)$ is smaller than a convergence threshold ε or a maximal number of iterations I has been reached. Subsequently, the possibly fractional local optimum $\mathbf{X}_{k+1}$ is projected to the closest integral solution $\widehat{\mathbf{X}}$, and the upper bound $UB := \min\{UB, c(\widehat{\mathbf{X}})\}$ is returned.

Populating the LSAPE instance $\mathbf{C}_k$ in step 1 requires $O(k|V^G||V^H|\max\{|V^G|,|V^H|\})$ time. Solving the LSAPE instance in step 2 requires $O(\min\{|V^G|,|V^H|\}^2\max\{|V^G|,|V^H|\})$ time. Updating the upper bound in step 3 requires $O(\max\{|V^G|,|V^H|\}^2)$ time. Determining the optimal step width α_{k+1} in step 4 can be done analytically in $O(|V^G||V^H|)$ time. And projecting the final fractional solution $\mathbf{X}_{k+1}$ to the integral solution $\widehat{\mathbf{X}}$ requires $O(\min\{|V^G|,|V^H|\}^2 \max\{|V^G|,|V^H|\})$ time. IPFP's overall runtime complexity is hence $O(I^2|V^G||V^H|\max\{|V^G|,|V^H|\})$.

Slightly different versions of IPFP that use LSAP instead of LSAPE as a linear model have been presented in [14] and [7]. The main advantage of these versions w.r.t. the one presented in this survey is that they are easier to implement: Unlike LSAPE, LSAP is a standard combinatorial optimization problem and solvers are available for all major programming languages. The drawback of the version presented in [14] is that is uses a significantly larger quadratic matrix $\mathbf{Q}$, while the drawback of the version presented in [7] is that it can be used only for quasimetric edit cost functions.

7.3 Extensions of the paradigm LS-GED

Next, we present two extensions of the paradigm LS-GED. Both of them can be used to improve all of the heuristics described in Section 7.2 above.

7.3.1 The extension MULTI-START

MULTI-START was suggested in [26] as an extension to the IPFP algorithm. While the general LS-GED framework computes a local optimum, the quality of the local optimum highly depends on the initialization of the method, which is a general drawback of local search methods. Hence, the MULTI-START extension to the framework simply proposes to use K different initial solutions, run the LS-GED framework on each of them (possibly in parallel), and return the best among the K computed local optima.

In order to further reduce the computing time of MULTI-START when parallelization is available, it was suggested in [13] to run in parallel more local searches than the number of desired local optima and to stop the whole process when the number local searches that have converged has reached the number of desired local optima. In this context, the framework runs with two parameters: K is the number of initial solutions, and $0 < \rho \leq 1$ is defined such that $\lceil \rho \cdot K \rceil$ is the number of desired computed local optima.

7.3.2 The extension RANDPOST

The RANDPOST framework initially proposed in [13] and refined in [12] aims at extending the the MULTI-START framework by running it several times in a row, and using the information contained in the computed local optima computed so far in order to produce better initializations. In addition to the two parameters K and ρ of MULTI-START, RANDPOST requires two parameters: the number of iterations $L \in \mathbb{N}$ and a penalty parameter $\eta \in [0,1]$. RANDPOST maintains a score matrix $\mathbf{M} \in \mathbb{R}_{\geq 0}^{(|V^G|+1)\times(|V^H|+1)}$ for all possible node assignments, which is initialized as $\mathbf{0}_{(|V^G|+1)\times(|V^H|+1)}$. The score for each substitution $(u_i, v_k) \in V^G \times V^H$ is represented by the value $m_{i,k}$ in the score matrix $\mathbf{M}$, while the scores for the deletion (u_i, ε) and the insertion (ε, v_k) are represented by the values $m_{i,|V^H|+1}$ and $m_{|V^G|+1,k}$, respectively. When the penalty parameter η equals 0, $m_{i,k}$ always represents the number of converged local optima that contain the corresponding assignment. When $\eta > 0$, the score of each assignment depends on both the number and the cost of converged local optima that contain it (assignments that appear in node maps with lower costs receive higher scores).

RANDPOST starts by running one iteration of MULTI-START. Next, RANDPOST carries out L iterations of its main for-loop. Inside this for-loop, RANDPOST starts by updating the scores matrix $\mathbf{M}$ as $\mathbf{M} := \mathbf{M} + \sum_{\pi\in S} \mathbf{X}^\pi[(1-\eta) + \eta \frac{UB-LB}{c(P_\pi)-LB}]$, where S is the set of converged node maps computed by the previous iteration of MULTI-START and, for each node map $\pi \in S$, $\mathbf{X}^\pi$ is the binary matrix that encodes π. Subsequently, RANDPOST randomly generates new initial node maps such that assignments with higher scores are more likely to be part of the generated node maps: For each of the first $|V^G|$ rows $\mathbf{M}_i$ of the score matrix $\mathbf{M}$, RANDPOST draws a column $k \in [|V^H|+1]$ from the distribution encoded my $\mathbf{M}_i$. If $k = |V^H|+1$, the node deletion (u_i, ε) is added to the node map π that is being constructed. Otherwise, the substitution (u_i, v_k) is added to π, the score $m_{j,k}$ is temporarily set to 0 for all $j \in [|V^G|] \setminus [i]$, and the column k is marked as covered. Once all nodes of G have been processed, node insertions (ε, v_k) are added to π for all uncovered columns $k \in [|V^H|]$. This process is repeated until K different node maps have been created. Once all new initial node maps have been generated, RANDPOST carries out another iteration of MULTI-START and updates the upper bound if one of the newly computed node maps yields an improvement.

8 Miscellaneous heuristics

In this section, we describe six algorithms that do not instantiate any of the paradigms LSAPE-GED, LS-GED, and LP-GED discussed in Sections 5 to 6. The first three algorithms presented in Sections 8.1 to 8.3 accept arbitrary edit cost functions, whereas the remaining three algorithms presented in Sections 8.4 to 8.6 are designed for uniform edit costs.

8.1 The algorithm HED

Given two input graphs G and H, the algorithm HED [28] starts by constructing the same LSAPE instance $\mathbf{C} \in \mathbb{R}^{(|V^G|+1)\times(|V^H|+1)}$ as the algorithm BRANCH presented in Section 5.2.3 above. However, instead of feeding $\mathbf{C}$ into an

LSAPE solver for obtaining upper and lower bounds for GED, HED computes a lower bound

$$LB := 0.5 \cdot \sum_{i=1}^{|V^G|} \min_{k \in [|V^H|+1]} c_{i,k} + 0.5 \cdot \sum_{k=1}^{|V^H|} \min_{i \in [|V^G|+1]} c_{i,k}$$

for GED by summing the minima of $\mathbf{C}$'s rows and columns. Note that, in general, LB does not correspond to a feasible LSAPE solution, because of which HED does not compute an upper bound for GED. Furthermore, it holds that $LB \leq \text{LSAPE}(\mathbf{C})$, which implies that HED's lower bound is never tighter than the lower bound computed by BRANCH.

As detailed in Section 5.2.3, the LSAPE instance $\mathbf{C}$ can be constructed in $O(|V^G||V^H|\Delta_{\min}^{G,H^2}\Delta_{\max}^{G,H})$ time, where $\Delta_{\min}^{G,H} := \min\{\max\deg(G), \max\deg(H)\}$ and $\Delta_{\max}^{G,H} := \max\{\max\deg(G), \max\deg(H)\}$. This implies that the overall runtime complexity of HED is $O(|V^G||V^H|\Delta_{\min}^{G,H^2}\Delta_{\max}^{G,H})$.

8.2 The algorithm BRANCH-TIGHT

Given two input graphs G and H, the algorithm BRANCH-TIGHT [9] starts by enforcing $|V^G| = |V^H| =: N$. If the edit cost functions c_V and c_E are metric, this is done by adding $\max\{|V^G|, |V^H|\} - |V^G|$ isolated dummy nodes to G and adding $\max\{|V^G|, |V^H|\} - |V^H|$ isolated dummy nodes to H. Otherwise, $|V^H|$ isolated dummy nodes are added to G and $|V^G|$ isolated dummy nodes are added to H. Next, dummy edges are added to G and H to render G and H d-regular with $d = O(\Delta_{\max}^{G,H})$. Both of these preprocessing operations leave $\text{GED}(G,H)$ invariant.

After preprocessing the input graphs, BRANCH-TIGHT runs an anytime algorithm that, given a maximal number of iterations I, computes lower bounds $(LB_s)_{r=1}^{I}$ and upper bounds $(UB_s)_{r=1}^{I}$ for GED such that LB_1 equals the lower bound computed by the algorithm BRANCH presented in Section 5.2.3 above and $LB_{r+1} \geq LB_s$ holds for all $r \in [I-1]$. Once I or a given time limit has been reached or the lower bound has converged, BRANCH-TIGHT returns the last lower bound $LB := LB_{I'}$ and the best encountered upper bound.

BRANCH-TIGHT repeatedly solves instances of the linear sum assignment or minimum cost perfect bipartite matching problem (LSAP). LSAP is similar to LSAPE but does not allow deletions and insertions of rows and columns.

Definition 8 (LSAP) Given a square matrix $\mathbf{C} \in \mathbb{R}^{n \times n}$, the *linear sum assignment problem* (LSAP) asks to minimize $\mathbf{C}(\mathbf{X}) := \sum_{i=1}^{n}\sum_{k=1}^{m} c_{i,k}x_{i,k}$ over all permutation matrices $\mathbf{X} \in \widehat{\Pi}(\mathbf{C})$, where $\widehat{\Pi}(n,m) := \{\mathbf{X} \in \{0,1\}^{n \times n} \mid \mathbf{1}^{\mathsf{T}}\mathbf{X} = \mathbf{1}^{\mathsf{T}} \wedge \mathbf{X}\mathbf{1} = \mathbf{1}\}$ and $\mathbf{1}$ is the n-sized vector of ones. $\text{LSAP}(\mathbf{C}) := \mathbf{C}(\mathbf{X})$ denotes the cost of an optimal solution $\mathbf{X}$.

For each $(u_i, v_k) \in V^G \times V^H$ and each iteration $r \in [I]$, BRANCH-TIGHT constructs and solves LSAP instances $\mathbf{C}^{i,k,r} \in \mathbb{R}^{d \times d}$ defined as

$$c_{j,l}^{i,k,r} := \begin{cases} 0.5 \cdot c_E((u_i,u_j),(v_k,v_l)) & \text{if } r = 1 \\ c_{j,l}^{i,k,r-1} - s_{j,l}^{i,k,r-1} - \frac{s_{i,k}^{r-1}}{d} + s_{i,k}^{j,l,r-1} + \frac{s_{j,l}^{r-1}}{d} & \text{else} \end{cases}$$

for all $(u_j, v_l) \in N^G(u_i) \times N^H(v_k)$. Here, $s_{j,l}^{r,i,k}$ is the slack of the variable $x_{j,l}$ in an optimal LSAP solution of the LSAP instance $\mathbf{C}^{i,k,r}$, and $s_{i,k}^{r}$ is the slack of the variable $x_{i,k}$ in an optimal solution of the LSAP instance $\mathbf{C}^r \in \mathbb{R}^{N \times N}$, which, in turn, is constructed by setting

$$c_{i,k}^{r} := c_V(u_i, v_k) + \text{LSAP}(\mathbf{C}^{i,k,r})$$

for all $(u_i, v_k) \in V^G \times V^H$. After constructing $\mathbf{C}^r$, an optimal solution $\mathbf{X}^r$ for $\mathbf{C}^r$ is computed, LB_r is set to $\text{LSAP}(\mathbf{C}^r)$, and UB_r is set to the cost of the edit path induced by $\mathbf{X}^r$. Subsequently, r is incremented and the process iterates.

Preprocessing the input graphs G and H requires $O(N^3\Delta_{\max}^{G,H^2})$ and one iteration of the anytime algorithm runs in $O(N^2\Delta_{\max}^{G,H^3} + N^3)$. This implies that BRANCH-TIGHT's overall runtime complexity is $O(N^3\Delta_{\max}^{G,H^2} + I(N^2\Delta_{\max}^{G,H^3} + N^3))$. Recall that we have $N = \max\{|V^G|, |V^H|\}$, if the edit cost functions are metric, and $N = |V^G| + |V^H|$, otherwise.

8.3 The algorithm SA

The algorithm SA [58] uses simulated annealing to improve the upper bound computed by an instantiation of the paradigm LSAPE-GED discussed in Section 5 above.[3] SA is hence similar to the local search based heuristics presented in Section 7. However, instead of varying an initial node map, SA varies the processing order for greedily computing a cheap solution for an initially computed LSAPE instance.

Assume w. l. o. g. that G and H are two input graphs with $|V^G| \geq |V^H|$. SA starts by running an instantiation of LSAPE-GED to obtain an initial node map $\pi \in \Pi(G,H)$, an LSAPE instance $\mathbf{C} \in \mathbb{R}^{(|V^G|+1)\times(|V^H|+1)}$, and, possibly, a lower bound LB. If the employed LSAPE-GED instantiation does not yield a lower bound, LB can be computed with any other method that produces a lower bound.

Given a maximal number of iterations I and start and end probabilities p_1 and p_I with $1 > p_1 \geq p_I > 0$ for accepting an unimproved node map, SA initializes an ordering $\sigma : [|V^G|] \to [|V^G|]$ of the first $|V^G|$ rows of $\mathbf{C}$ by setting $\sigma(i) := i$ for all $i \in [|V^G|]$, computes a cooling factor $a := (\log(p_1)/\log(p_I))^{1/(I-1)}$ such that $p_1^{a^{-(I-1)}} = p_I$, sets the current acceptance probability to $p := p_1$, initializes the best encountered node map π' and the current node map

[3] In [58], SA is presented as a technique for improving the upper bound computed by the LSAPE-GED instantiation BP. Since SA can be used with any instantiation of LSAPE-GED, we here present a more general version.

π'' as $\pi' := \pi'' := \pi$, and sets the number r of consecutive iterations without improvement of the upper bound to $r := 0$.

While the maximal number of iterations I has not been reached and the best upper bound $c(P_{\pi'})$ is greater than LB, `SA` does the following: First, a candidate row ordering σ' is obtained from the current ordering σ by setting $\sigma'(1) := \sigma(i)$, $\sigma'(j) := \sigma(j-1)$ for all $j \in [i] \setminus \{1\}$, and $\sigma'(j) := \sigma(j)$ for all $j \in [|V^G|] \setminus [i]$, where $i \in [|V^G|]$ is a randomly selected row of $\mathbf{C}$. Next, a candidate node map π''' is computed by greedily assigning the σ'-ordered rows of $\mathbf{C}$ to the cheapest unassigned columns. If π''''s induced upper bound is cheaper than the upper bound of the current node map π'', π'' and σ are updated to π''' and σ', respectively. Otherwise, they are updated with a probability that is proportional to the current acceptance probability p and inversely proportional to the deterioration $c(P_{\pi'''}) - c(P_{\pi''})$ of the induced upper bound.

After updating π'' and σ, `SA` checks if π'''s induced upper bound is tighter than the upper bound induced by the best encountered node map π'. If this is the case, π' is updated to π'' and the number r of consecutive iterations without improvement is reset to 0. Otherwise, r is incremented and the current ordering σ is reshuffled randomly with probability r/I. Finally, the current acceptance probability p is set to $p_1^{\alpha^{-s}}$, where s is the number of the current iteration, and `SA` iterates. After exiting the main loop, `SA` returns $UB := c(P_{\pi'})$.

The dominant operations in one iteration of `SA` are the greedy computation of the candidate node map π''' and the computation of its induced upper bound. One iteration of `SA` hence runs in $O(|V^G||V^H| + \max\{E^G, E^H\})$ time. This implies that `SA`'s overall runtime complexity is $O(\omega + I(|V^G||V^H| + \max\{E^G, E^H\}))$, where $O(\omega)$ is the runtime required for computing the initial upper and lower bounds as well as the LSAPE instance $\mathbf{C}$.

8.4 The algorithm `BRANCH-COMPACT`

The algorithm `BRANCH-COMPACT` [71] yields a lower bound for GED with uniform edit cost functions c_V and c_E. Recall that c_V and c_E are uniform if there is a constant $c \in \mathbb{R}_{>0}$ such that $c_V(\alpha, \alpha') = c_E(\beta, \beta') = c$ holds for all node labels $(\alpha, \alpha') \in (\Sigma_V \cup \{\varepsilon\}) \times (\Sigma_V \cup \{\varepsilon\})$ with $\alpha \neq \alpha'$ and all edge labels $(\beta, \beta') \in (\Sigma_E \cup \{\varepsilon\}) \times (\Sigma_E \cup \{\varepsilon\})$ with $\beta \neq \beta'$.

Given input graphs G and H, `BRANCH-COMPACT` starts by constructing branches $\mathscr{B}^G_i := (\ell^G_V(u_i), \ell^G_E[\![E^G(u_i)]\!])$ and $\mathscr{B}^H_k := (\ell^H_V(v_k), \ell^H_E[\![E^H(v_k)]\!])$ for all $u_i \in V^G$ and all $v_k \in V^H$. Subsequently, `BRANCH-COMPACT` sorts the branches in non-decreasing lexicographical order, i. e., computes orderings $\sigma^G : [|V^G|] \to [|V^G|]$ and $\sigma^H : [|V^H|] \to [|V^H|]$ such that $\mathscr{B}^G_{\sigma^G(i)} \preceq_L \mathscr{B}^G_{\sigma^G(i+1)}$ holds for all $i \in [|V^G| - 1]$ and $\mathscr{B}^H_{\sigma^H(k)} \preceq_L \mathscr{B}^H_{\sigma^H(k+1)}$ holds for all $k \in [|V^H| - 1]$.

`BRANCH-COMPACT` now performs a first parallel linear scan over the sorted sequences of branches $(\mathscr{B}^G_{\sigma^G(i)})_{i=1}^{|V^G|}$ and $(\mathscr{B}^H_{\sigma^H(k)})_{k=1}^{|V^H|}$ to delete a maximal number of pairs of branches $(\mathscr{B}^G_{\sigma^G(i)}, \mathscr{B}^H_{\sigma^H(k)})$ with $\mathscr{B}^G_{\sigma^G(i)} = \mathscr{B}^H_{\sigma^H(k)}$. Subsequently, `BRANCH-COMPACT` initializes its lower bound as $LB := 0$ and performs a second parallel linear scan over the remaining branches. In this scan, a maximal number of pairs of branches $(\mathscr{B}^G_{\sigma^G(i)}, \mathscr{B}^H_{\sigma^H(k)})$ with $\ell^G_V(u_{\sigma^G(i)}) = \ell^H_V(v_{\sigma^H(k)})$ is deleted and LB is incremented by $c/2$ for each deleted pair of branches. Finally, LB is incremented by $c(\max\{|V^G|, |V^H|\} - D)$, where D is the number of pairs of branches that have been deleted during the two scans.

`BRANCH-COMPACT` first sorts the branches in $O(\max\{|V^G|, |V^H|\} \Delta^{G,H}_{\max} \log(\Delta^{G,H}_{\max}))$ time, and then computes its lower bound in $O(\max\{|V^G|, |V^H|\})$ time. `BRANCH-COMPACT`'s overall runtime complexity is hence $O(\max\{|V^G|, |V^H|\}(\Delta^{G,H}_{\max} \log(\Delta^{G,H}_{\max}) + \log(\max\{|V^G|, |V^H|\})))$.

8.5 The algorithm `PARTITION`

The algorithm `PARTITION` [71] computes a lower bound for GED with uniform edit costs. Given input graphs G and H and a constant $K \in \mathbb{R}_{\geq 1}$, `PARTITION` starts by initializing a collection $\mathscr{S} := \emptyset$ of K'-sized substructures of G that are not subgraph-isomorphic to H, where $K' \in [K]$ and a K'-sized substructure of G is a connected subgraph of G that is composed of K' elements (nodes or edges). For instance, 1-sized substructures are single nodes or edges, 2-sized substructures are nodes together with an incident edge, and 3-sized substructures are nodes together with two incident edges or edges together with their terminal nodes.

Starting with $K' := 1$, `PARTITION` now consecutively checks for each K'-sized substructure $S^G \subseteq G$ of G if there is a K'-sized substructure of H which is isomorphic to S^G. If this is not the case S^G is added to $\mathscr{S}$ and deleted from G. Once all K'-sized substructure have been considered, K' is incremented and the process iterates if $K' \leq K$. Otherwise, `PARTITION` returns the lower bound $LB := c|\mathscr{S}|$.

Since G and H have, respectively, $O(|E^G|)$ and $O(|E^H|)$ substructures of sizes 1, 2, and 3, `PARTITION` with $K \leq 3$ runs in $O(|E^G||E^H|)$ time. Determining non-isomorphic substructures of size $K > 3$ cannot be done naively but requires to call subgraph isomorphism verification algorithms such as the one proposed in [24]. These algorithms run in superpolynomial time but are often fast in practice for small K.

8.6 The algorithm `HYBRID`

The algorithm `HYBRID` [71] improves the lower bounds of the algorithms `BRANCH-CONST` and `PARTITION` presented in Section 5.2.5 and Section 8.5. Given input graphs G and H and a constant $K \in \mathbb{R}_{\geq 1}$, `HYBRID` first runs `PARTITION` with the maximal size of the considered substructures set to K, and hence obtains a collection $\mathscr{S}$ of substructures $S^G \subseteq G$ of G that are not subgraph-isomorphic to H.

Let $\mathscr{C}(\mathscr{S}) := \bigtimes_{S^G \in \mathscr{S}} S^G$ be the set of all configurations of nodes or edges that appear in the non-isomorphic substructures. For each configuration $a := (a_s)_{s=1}^{|\mathscr{S}|} \in \mathscr{C}(\mathscr{S})$, HYBRID creates a modified graph G_a, where all nodes or edges a_s contained in the configuration a get a special wildcard label γ, and runs a variant of BRANCH-CONST on the graphs G_a and H, which edits γ-labeled nodes and edges for free. Finally, HYBRID returns the lower bound $LB := |\mathscr{S}| + \min\{LB_a \mid a \in \mathscr{C}(\mathscr{S})\}$, where LB_a is the lower bound returned by the wildcard version of BRANCH-CONST if run on the graphs G_a and H. This lower bound is guaranteed to be as least as tight as the lower bounds computed by PARTITION and BRANCH-CONST.

Let $O(\omega_1)$ be the runtime complexity of PARTITION with the maximal size of the considered substructures set to K and $O(\omega_2)$ be the runtime complexity of BRANCH-CONST. Then HYBRID runs in $O(\omega_1 + \omega_2|\mathscr{C}(\mathscr{S})|)$ time. Note that $|\mathscr{C}(\mathscr{S})|$ can get huge. For instance, assume that PARTITION completely partitions G into substructures of size 2. Then it holds that $|\mathscr{C}(\mathscr{S})| = \prod_{S^G \in \mathscr{S}} |S^G| = 2^{|V^G|}$. HYBRID's runtime complexity is hence not polynomially bounded.

9 Experimental evaluation

We carried out extensive experiments to empirically evaluate the presented heuristics and to address the two meta-questions Q1 and Q2 introduced in Section 1. We first describe the setup of our experiments (Sections 9.1 to 9.4) and then report their results (Sections 9.5 to 9.9).

9.1 Datasets and edit cost functions

We tested on the widely used benchmark datasets AIDS, MUTA, PROTEIN, LETTER (H), GREC, and FP from the IAM Graph Database Repository [50,52], and used the edit cost functions suggested in [2,52]. Table 5 summarizes important statistics of the datasets. For details on the edit cost function, cf. Appendix A. In order to be able to compare all heuristics on all datasets, we used the technique described in Section 4 to extend heuristics with cost constraints to general edit costs.

9.2 Choice of options and parameters

For all instantiations of the paradigms LSAPE-GED, LS-GED, and LP-GED and for all miscellaneous heuristics, we followed the original publications to determine their meta-parameters. In the remainder of this section, we give detailed descriptions for each heuristic and describe how we tested the extensions MULTI-SOL and CENTRALITIES of the paradigm LSAPE-GED, as well as the extensions MULTI-START and RANDPOST of the paradigm LS-GED.

– *Meta-parameters for SUBGRAPH and WALKS:* As suggested in [21] and [32], for each dataset, we determined the parameters K of SUBGRAPH and WALKS as the $K \in [5]$ that yielded the tightest average upper bounds on a set of training graphs. To cope with SUBGRAPH's exponential runtime complexity, we set a time limit of 1 ms for the computation of each cell of its LSAPE instance $\mathbf{C}$.
– *Options and meta-parameters for RING:* As highlighted in [4, 6], RING performs best if the node and edge set distances are computed via optimal LSAPE solvers or multiset intersection based proxies. We included both options in our experiments; the resulting heuristics are denoted as $\texttt{RING}^{\texttt{OPT}}$ and $\texttt{RING}^{\texttt{MS}}$, respectively. For both variants and each dataset, the meta-parameters λ_l, α_s, and K were determined by running a blackbox optimizer on a set of training graphs, as suggested in [4, 6].
– *Options for RING-ML and PREDICT:* As highlighted in [4], the machine learning based heuristics RING-ML and PREDICT perform best if one-class support vector machines with RBF kernel or fully connected feedforward deep neural networks are used for training. We included both variants in our experiments; the resulting heuristics are denoted as $\texttt{RING-ML}^{\texttt{1-SVM}}$, $\texttt{RING-ML}^{\texttt{DNN}}$, $\texttt{PREDICT}^{\texttt{1-SVM}}$, and $\texttt{PREDICT}^{\texttt{DNN}}$, respectively.
– *Meta-parameters for K-REFINE:* We ran K-REFINE with swap size $K := 3$. We hence followed the suggestion in [12], where it is highlighted that $K > 3$ leads to an enormous blowup of K-REFINE's runtime on larger graphs.
– *Meta-parameters for BP-BEAM and IBP-BEAM:* As suggested in [56] and [27], we set the beam size employed by BP-BEAM and IBP-BEAM to $K := 5$ and the number of iterations employed by IBP-BEAM to $I := 20$.
– *Options and meta-parameters for IPFP:* As highlighted in [7], the best performing variant of IPFP that can cope with general edit costs is the one suggested in [16]. In our experiments, we therefore only included this variant. Like in the experiments of the original publications, we set the maximal number of iterations to $I := 100$ and the convergence threshold to $\varepsilon := 10^{-3}$.
– *Meta-parameters for BRANCH-TIGHT:* As suggested in [9], we set the number of iterations carried out by BRANCH-TIGHT to $I := 20$.
– *Meta-parameters for SA:* As suggested in [58], we set SA's number of iterations to $I := 100$ and used start and end probabilities $p_1 := 0.8$ and $p_I := 10^{-2}$. We used BRANCH for computing SA's initial LSAPE instance $\mathbf{C}$.
– *Meta-parameters for PARTITION and HYBRID:* In [71], it is suggested to set the maximal size of the substructures employed by PARTITION and HYBRID to $K := 8$. However, how to implement these heuristics with $K > 3$ is not well documented in [71] and the authors did not reply to

Table 5 Overview of test datasets.

dataset	# graphs	# classes	# nodes					# edges					labels	
			min	max	mean	std	median	min	max	mean	std	median	nodes	edges
AIDS	1500	2	2	95	15.7	13.8	11	1	103	16.2	15.1	11	yes	yes
MUTA	4337	2	4	417	30.3	20.1	27	3	112	30.8	16.8	28	yes	yes
PROTEIN	600	6	2	126	32.6	15.3	32	1	149	62.1	25.5	60	yes	yes
LETTER (H)	2250	15	1	9	4.7	1.3	5	0	9	4.5	1.6	5	yes	no
GREC	1100	22	4	24	11.5	4.9	11	2	29	11.9	6.0	10	yes	yes
FP	2800	4	0	26	5.4	5.0	4	0	24	4.4	4.5	3	no	yes

our request to share their implementation. We therefore used $K := 3$ for our experiments. To cope with HYBRID's exponential runtime complexity, we set a time limit of 1 s and set up HYBRID to return the maximum of the lower bounds computed by PARTITION and BRANCH-CONST if it did not terminate within the time limit.

– *Configurations for the extensions MULTI-SOL and CENTRALITIES of the paradigm LSAPE-GED:* In order to test MULTI-SOL and CENTRALITIES, we ran all instantiations of LSAPE-GED with all configurations $(K,\gamma) \in \{1,3,7,10\} \times \{0,0.7\}$, where K is the maximal number of solutions computed by MULTI-SOL and γ is the weight of the centralities used by CENTRALITIES. We used pagerank centralities with $\gamma = 0.7$, because in [53] this setup is reported to yield the best results among all variants of CENTRALITIES. MULTI-SOL is used just in case $K \neq 1$ and CENTRALITIES is used just in case $\gamma \neq 0$.
– *Configurations for the extensions MULTI-START and RANDPOST of the paradigm LS-GED:* For testing MULTI-START and RANDPOST, we ran each LS-GED instantiation with all $(K,\rho,L,\eta) \in (\{(40,1/2,1),(40,1/4,3),(40,1/8,7)\} \times \{0,1\}) \cup (\{1,10,20,30,40\} \times \{(1,0,0)\})$. K is the number of initial node maps constructed by MULTI-START, $\lceil \rho \cdot K \rceil$ is the number of completed runs from initial node maps, L is the number of RANDPOST loops, and η is the penalty for expensive converged node maps employed by RANDPOST. The initial node maps were constructed randomly under the constraint that they contain exactly $\min\{|V^G|,|V^H|\}$ node substitutions. Note that each configuration that uses RANDPOST (i. e., has $L > 0$) in total carries out exactly 40 runs from different initial node maps.

9.3 Test protocol and test metrics

For each test dataset $\mathscr{D}$, we randomly selected a training set $\mathscr{D}_{\text{train}} \subseteq \mathscr{D}$ and a testing set $\mathscr{D}_{\text{test}} \subseteq \mathscr{D} \setminus \mathscr{D}_{\text{train}}$. We ensured that both sets are balanced w. r. t. the classes of the contained graphs and set their sizes to the largest integers not greater than, respectively, 50 (for training) and 100 (for testing) that allowed balancing. All algorithms that require training were trained on $\mathscr{D}_{\text{train}}$. Subsequently, we ran all compared algorithms on all pairs of graphs $(G,H) \in \mathscr{D}_{\text{test}} \times \mathscr{D}_{\text{test}}$. Recall that we compared various configurations of the extensions MULTI-SOL and CENTRALITIES for the instantiation of LSAPE-GED; and that we tested various configurations of the extensions MULTI-START and RANDPOST for the instantiation of LS-GED. In the following, the expression "algorithm" denotes a heuristic together with its configuration.

Algorithms for GED computation are typically evaluated w. r. t. their runtime behaviour, the tightness of the produced bounds, and the performance of pattern recognition frameworks that use the produced bounds as underlying distance measures (cf. the criteria C1 to C3). For all compared algorithms ALG, we therefore recorded the average runtime $t(\texttt{ALG})$. Moreover, we recorded the average lower bound $d_{LB}(\texttt{ALG})$ and the classification coefficient $c_{LB}(\texttt{ALG})$ for all algorithms that yield lower bounds, and the average upper bound $d_{UB}(\texttt{ALG})$ and the classification coefficient $c_{UB}(\texttt{ALG})$ for all algorithms that yield upper bounds.

The coefficients c_{LB} and c_{UB} were computed as

$$c_{LB}(\texttt{ALG}) := (d_{LB}^{\text{inter}}(\texttt{ALG}) - d_{LB}^{\text{intra}}(\texttt{ALG}))/\max LB(\texttt{ALG})$$
$$c_{UB}(\texttt{ALG}) := (d_{UB}^{\text{inter}}(\texttt{ALG}) - d_{UB}^{\text{intra}}(\texttt{ALG}))/\max UB(\texttt{ALG}),$$

where $d_{LB}^{\text{inter}}(\texttt{ALG})$ and $d_{UB}^{\text{inter}}(\texttt{ALG})$ are the average lower and upper bounds between graphs with different classes, $d_{LB}^{\text{intra}}(\texttt{ALG})$ and $d_{UB}^{\text{intra}}(\texttt{ALG})$ are the average lower and upper bounds between graphs with the same class, and $\max LB(\texttt{ALG})$ and $\max UB(\texttt{ALG})$ denote the maximal lower and upper bounds computed by ALG. The reason for defining the classification coefficients in this way is that pattern recognition frameworks based on distance measures perform well just in case the intra-class distances are significantly smaller than the inter-class distances. Hence, large classification coefficients $c_{LB}(\texttt{ALG})$ and $c_{UB}(\texttt{ALG})$ imply that the respective lower or upper bounds are fit for use within distance based pattern recognition frameworks. We normalized by the maximal lower and upper bounds in order to ensure $c_{LB}(\texttt{ALG}), c_{UB}(\texttt{ALG}) \in [-1,1]$ and hence render the classification coefficients comparable across different datasets. We rounded $t(\texttt{ALG})$ to microseconds and $d_{LB|UB}(\texttt{ALG})$ as well as $c_{LB|UB}(\texttt{ALG})$ to two decimal places.

After running all algorithms, we computed a joint score $s_{LB}(\mathtt{ALG}) \in [0,1]$ for all algorithms that yield lower bounds and a joint score $s_{UB}(\mathtt{ALG}) \in [0,1]$ for all algorithms that yield upper bounds. The joint scores are defined as

$$s_{LB}(\mathtt{ALG}) := \frac{d_{LB}(\mathtt{ALG})}{3 \cdot d^\star_{LB}} + \frac{t^\star_{LB}}{3 \cdot t(\mathtt{ALG})} + \frac{c_{LB}(\mathtt{ALG})}{3 \cdot c^\star_{LB}}$$
$$s_{UB}(\mathtt{ALG}) := \frac{d^\star_{UB}}{3 \cdot d_{UB}(\mathtt{ALG})} + \frac{t^\star_{UB}}{3 \cdot t(\mathtt{ALG})} + \frac{c_{UB}(\mathtt{ALG})}{3 \cdot c^\star_{UB}},$$

where $d^\star_{LB}$, $t^\star_{LB}$, and $c^\star_{LB}$ denote the best (i. e., largest) average lower bound, the best average runtime, and the best classification coefficient yielded by any algorithm that computes a lower bound. Analogously, $d^\star_{UB}$, $t^\star_{UB}$, and $c^\star_{UB}$ denote the best (i. e., smallest) average upper bound, the best average runtime, and the best classification coefficient yielded by any algorithm that computes an upper bound. With this definition, each evaluation criterion contributes a quantity between 0 and $1/3$ to the joint score, and an algorithm has joint score 1 if it performs best w. r. t. all three criteria.

We partially ordered the compared algorithms w. r. t. the Pareto dominance relations $\succ_{LB}$ and $\succ_{UB}$. For two algorithms $\mathtt{ALG}_1$ and $\mathtt{ALG}_2$ that compute lower bounds, we say that the lower bound computed by $\mathtt{ALG}_1$ dominates the one produced by $\mathtt{ALG}_2$ on a given dataset (in symbols: $\mathtt{ALG}_1 \succ_{LB} \mathtt{ALG}_2$) just in case $\mathtt{ALG}_1$ performs at least as good as $\mathtt{ALG}_2$ w. r. t. to all three evaluation criteria d_{LB}, t, and c_{LB}, and strictly better than $\mathtt{ALG}_2$ w. r. t. at least one of them. The dominance relation $\succ_{UB}$ for the upper bounds is defined analogously. Note that, with these definitions, $\mathtt{ALG}_1 \succ_{LB} \mathtt{ALG}_2$ implies $s_{LB}(\mathtt{ALG}_1) > s_{LB}(\mathtt{ALG}_2)$ and $\mathtt{ALG}_1 \succ_{UB} \mathtt{ALG}_2$ implies $s_{UB}(\mathtt{ALG}_1) > s_{UB}(\mathtt{ALG}_2)$, but the inverse implications do not hold. The joint scores s_{LB} and s_{UB} hence allow to compare algorithms that are Pareto optimal.

Using the partial orders $\succ_{LB}$ and $\succ_{UB}$, we computed aggregated joint lower bound scores $\widehat{s_{LB}}(\mathtt{H})$ for all heuristics $\mathtt{H}$ that compute lower bounds, as well as aggregated joint upper bounds score $\widehat{s_{UB}}(\mathtt{H})$ and $\widehat{s_{UB}}(\mathtt{E})$ for all heuristics $\mathtt{H}$ that compute lower bounds and all extensions $\mathtt{E}$ of the paradigms LSAPE-GED and LS-GED. These scores were computed as

$$\widehat{s_{LB}}(\mathtt{H}) := \delta_{\mathscr{C}(\mathtt{H})\cap \mathrm{MAX}_{\succ_{LB}} \neq \emptyset} \max_{\mathtt{ALG}\in\mathscr{C}(\mathtt{H})\cap \mathrm{MAX}_{\succ_{LB}}} s_{LB}(\mathtt{ALG})$$
$$\widehat{s_{UB}}(\mathtt{H}) := \delta_{\mathscr{C}(\mathtt{H})\cap \mathrm{MAX}_{\succ_{UB}} \neq \emptyset} \max_{\mathtt{ALG}\in\mathscr{C}(\mathtt{H})\cap \mathrm{MAX}_{\succ_{UB}}} s_{UB}(\mathtt{ALG})$$
$$\widehat{s_{UB}}(\mathtt{E}) := \frac{\delta_{\mathscr{C}(\mathtt{P}(\mathtt{E}))\cap \mathrm{MAX}_{\succ_{UB}} \neq \emptyset} \sum_{\mathtt{ALG}\in\mathscr{C}(\mathtt{E})\cap \mathrm{MAX}_{\succ_{UB}}} s_{UB}(\mathtt{ALG})}{\sum_{\mathtt{ALG}\in\mathscr{C}(\mathtt{P}(\mathtt{E}))\cap \mathrm{MAX}_{\succ_{UB}}} s_{UB}(\mathtt{ALG})},$$

where $\mathscr{C}(\mathtt{H})$ is the set of compared algorithms that are configurations of the heuristic $\mathtt{H}$, $\mathscr{C}(\mathtt{E})$ is the set of compared algorithms that use the extension $\mathtt{E}$, $\mathscr{C}(\mathtt{P}(\mathtt{E}))$ is the set of compared algorithms that instantiate the paradigm extended by $\mathtt{E}$, and $\mathrm{MAX}_{\succ_{LB}}$ and $\mathrm{MAX}_{\succ_{UB}}$ are the set of maxima w. r. t. the partial orders $\succ_{LB}$ and $\succ_{UB}$, respectively. In other words, we set the aggregated joint scores $\widehat{s_{LB}}(\mathtt{H})$ and $\widehat{s_{UB}}(\mathtt{H})$ of a heuristic $\mathtt{H}$ to the maximal scores of Pareto optimal configurations of $\mathtt{H}$, and to 0 if no configurations of $\mathtt{H}$ were Pareto optimal. The aggregated joint upper bound score $\widehat{s_{UB}}(\mathtt{E})$ of an extension $\mathtt{E}$ of the paradigms LSAPE-GED and LS-GED was set to the sum of the joint upper bound scores of Pareto optimal algorithms that use $\mathtt{E}$ divided by the sum of the joint upper bound scores of Pareto optimal algorithms that instantiate the paradigm extended by $\mathtt{E}$, and to 0 if no algorithms that instantiate the paradigm extended by $\mathtt{E}$ were Pareto optimal. We also computed vectors $\chi_{LB}(\mathtt{H}) \in \{0,1\}^3$ for all heuristics that yield lower bounds and vectors $\chi_{UB}(\mathtt{H}), \chi_{UB}(\mathtt{E}) \in \{0,1\}^3$ for all heuristics that yield upper bounds and all extensions of the paradigms LSAPE-GED and LS-GED. These vectors indicate whether a heuristic or an extension has a configuration that performed best w. r. t. one or several of the observed metrics $f_{1_{LB|UB}} := d_{LB|UB}$, $f_{2_{LB|UB}} := t_{LB|UB}$, and $f_{3_{LB|UB}} := c_{LB|UB}$. That is, the indicator vectors were computed as follows:

$$\chi_{LB}(\mathtt{H}) := (\delta_{\exists \mathtt{ALG}\in\mathscr{C}(\mathtt{H}):f_{r_{LB}}(\mathtt{ALG})=f^\star_{r_{LB}}})^3_{r=1}$$
$$\chi_{UB}(\mathtt{H}) := (\delta_{\exists \mathtt{ALG}\in\mathscr{C}(\mathtt{H}):f_{r_{UB}}(\mathtt{ALG})=f^\star_{r_{UB}}})^3_{r=1}$$
$$\chi_{UB}(\mathtt{E}) := (\delta_{\exists \mathtt{ALG}\in\mathscr{C}(\mathtt{E}):f_{r_{UB}}(\mathtt{ALG})=f^\star_{r_{UB}}})^3_{r=1}$$

Finally, we trained linear regression models $c_{LB} \sim d_{LB} := (a_{LB}, m_{LB})$ and $c_{UB} \sim d_{UB} := (a_{UB}, m_{UB})$ defined as

$$(a_{LB}, m_{LB}) := \underset{(a,m)\in\mathbb{R}\times\mathbb{R}}{\arg\min} \sum_{\mathtt{ALG}} [c_{LB}(\mathtt{ALG}) - (a + m \cdot d_{LB}(\mathtt{ALG}))]^2$$
$$(a_{UB}, m_{UB}) := \underset{(a,m)\in\mathbb{R}\times\mathbb{R}}{\arg\min} \sum_{\mathtt{ALG}} [c_{UB}(\mathtt{ALG}) - (a + m \cdot d_{UB}(\mathtt{ALG}))]^2$$

for each dataset, which relate the tightnesses of the computed upper and lower bounds to the obtained classification coefficients: Tightness of lower bounds is positively correlated to high classification coefficients if the slope m_{LB} is positive; tightness of upper bounds is positively correlated to high classification coefficients if the slope m_{UB} is negative. We also computed the p-values p_{LB} and p_{UB} of the regression models, to assess whether the correlations between bounds and classification coefficients are statistically significant. Table 6 gives an overview of all test metrics.

9.4 Implementation and hardware specifications

To ensure comparability, we reimplemented all compared heuristics in C++. Our implementation builds upon the Boost Graph Library [41] and Eigen [33] for efficiently managing graphs and matrices. For solving LSAPE, we used the solver suggested in [17], which is efficiently implemented in the LSAPE toolbox available at `https://bougleux.users.greyc.fr/lsape/`. We used the blackbox optimizer NOMAD [39] for training $\mathtt{RING}^{\mathtt{OPT}}$ and

Table 6 Overview of test metrics.

syntax	semantic
observed metrics for compared algorithms	
$d_{LB\|UB}$	average lower and upper bounds
t	average runtime
$c_{LB\|UB}$	classification coefficients of lower and upper bounds
inferred metrics for compared algorithms	
$s_{LB\|UB}$	joint lower and upper bound scores
inferred metrics for compared heuristics and extensions	
$\widehat{s_{LB\|UB}}$	aggregated joint lower and upper bound scores
$\chi_{LB\|UB}$	indicate whether heuristics and extensions have configuration that are optimal w. r. t. observed metrics
inferred metrics for test datasets	
$d^{\star}_{LB\|UB}$	tightest average lower and upper bounds
$t^{\star}_{LB\|UB}$	average runtimes of fastest algorithms producing lower and upper bounds
$c^{\star}_{LB\|UB}$	best classification coefficients of lower and upper bounds
$m_{LB\|UB}$	slopes of linear regression models $c_{LB\|UB} \sim d_{LB\|UB}$
$p_{LB\|UB}$	p-values of linear regression models $c_{LB\|UB} \sim d_{LB\|UB}$

RING$^{\texttt{MS}}$, the support vector machine library LIBSVM [22] for training RING-ML$^{\texttt{1-SVM}}$ and PREDICT$^{\texttt{1-SVM}}$, the artificial neural network library FANN [47] for training RING-ML$^{\texttt{DNN}}$ and PREDICT$^{\texttt{DNN}}$, and the mathematical programming library Gurobi [34] for implementing the instantiations of LP-GED.

All heuristics were run in six threads: Instantiations of LSAPE-GED were set up to parallelly construct their LSAPE instance **C**, instantiations of LS-GED were implemented to parallelly carry out runs from several initial solutions, and instantiations of LP-GED were allowed to use multithreading when solving their LP via calls to Gurobi. For the miscellaneous heuristics, we used the following parallelization techniques: HED was set up to construct its LSAPE instance **C** in parallel, BRANCH-TIGHT was implemented to parallelize the construction phases of all of its LSAP instances $\mathbf{C}^r$, and SA and HYBRID were set up to use the parallelized versions of, respectively, BRANCH and BRANCH-CONST as subroutines. BRANCH-COMPACT and PARTITION do not allow straightforward parallelizations and where hence run in only one thread.

Source code and datasets are distributed with GEDLIB: `https://github.com/dbblumenthal/gedlib/` [5]. Tests were run on a machine with two Intel Xeon E5-2667 v3 processors with 8 cores each and 98 GB of main memory running GNU/Linux.

9.5 Lower bounds

Figure 9 shows the average lower bounds and runtimes of all heuristics that compute lower bounds. Globally, we see that instantiations of LP-GED (except COMPACT-MIP) yielded tight lower bound, but were also relatively slow. Instantiations

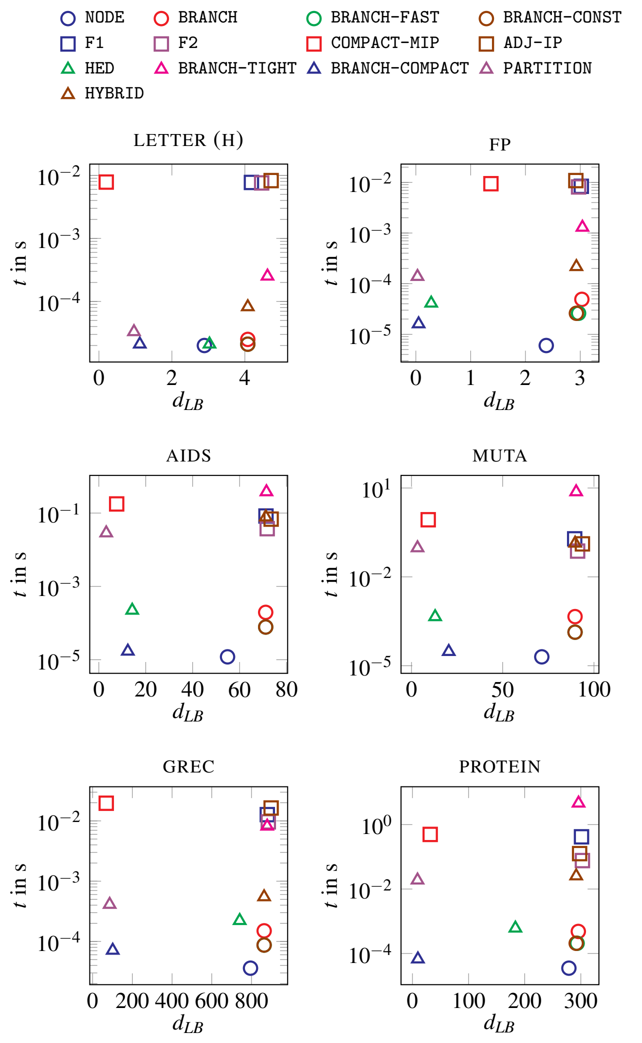


Fig. 9 Average lower bounds vs. average runtime. Instantiations of LSAPE-GED and LP-GED are displayed as circles and squares, respectively; miscellaneous methods are displayed as triangles.

of LSAPE-GED were faster and only slightly less accurate. Among the miscellaneous heuristics, only BRANCH-TIGHT and HYBRID produced competitive lower bounds.

Table 7 shows the aggregated joint lower bound scores $\widehat{s_{LB}}(\mathtt{H})$, as well as the indicator vectors $\chi_{LB}(\mathtt{H})$. The fast but relatively imprecise LSAPE-GED instantiations BRANCH-CONST and NODE were Pareto optimal on six (BRANCH-CONST) respectively five (NODE) out of six datasets. Among the more precise heuristics, BRANCH-TIGHT and the LP-GED instantiations ADJ-IP and F2 performed best. They were Pareto optimal on four (ADJ-IP) respectively three (F2 and BRANCH-TIGHT) out of six datasets.

In Figure 10, the results are further aggregated by averaging the scores and summing the indicator vectors over all datasets. Instantiations of LSAPE-GED are displayed white, instantiations of LP-GED are displayed light grey, and mis-

Table 7 Overview of results for lower bounds. For each heuristic `H` and each dataset, a non-zero aggregated joint lower bound score (displayed bold) implies that `H` was Pareto optimal on the dataset.

heuristic	LETTER (H)		MUTA		AIDS		PROTEIN		FP		GREC	
	χ_{LB}	$\widehat{s_{LB}}$	χ_{LB}	$\widehat{s_{LB}}$	χ_{LB}	$\widehat{s_{LB}}$	χ_{LB}	$\widehat{s_{LB}}$	χ_{LB}	$\widehat{s_{LB}}$	χ_{LB}	$\widehat{s_{LB}}$
instantiations of the paradigm `LSAPE-GED`												
`NODE`	(0,0,0)	0.00	(0,**1**,0)	**0.59**	(0,**1**,**1**)	**0.92**	(0,**1**,**1**)	**0.97**	(0,**1**,**1**)	**0.93**	(0,**1**,0)	**0.94**
`BRANCH`	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,**1**)	0.00	(0,0,**1**)	**0.68**	(0,0,0)	**0.68**	(0,0,**1**)	**0.73**
`BRANCH-FAST`	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,**1**)	0.00	(0,0,**1**)	**0.71**	(0,0,**1**)	**0.74**	(0,0,**1**)	**0.79**
`BRANCH-CONST`	(0,**1**,0)	**0.92**	(0,0,0)	**0.41**	(0,0,**1**)	**0.76**	(0,0,**1**)	**0.74**	(0,0,**1**)	**0.75**	(0,0,**1**)	**0.86**
`STAR`	(0,0,0)	0.00	(0,0,**1**)	0.00	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	0.00
instantiations of the paradigm `LP-GED`												
`F1`	(0,0,0)	0.00	(0,0,**1**)	0.00	(0,0,**1**)	0.00	(0,0,**1**)	0.00	(0,0,0)	0.00	(0,0,0)	0.00
`F2`	(0,0,0)	0.00	(0,0,0)	**0.32**	(0,0,**1**)	**0.66**	(**1**,0,**1**)	**0.67**	(0,0,0)	0.00	(0,0,0)	**0.65**
`COMPACT-MIP`	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	0.00
`ADJ-IP`	(**1**,0,**1**)	**0.67**	(**1**,0,**1**)	**0.67**	(**1**,0,**1**)	**0.67**	(0,0,**1**)	0.00	(0,0,**1**)	0.00	(**1**,0,0)	**0.66**
miscellaneous heuristics												
`HED`	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	0.00
`BRANCH-TIGHT`	(0,0,**1**)	**0.68**	(0,0,**1**)	0.00	(0,0,**1**)	0.00	(0,0,**1**)	0.00	(**1**,0,0)	**0.64**	(0,0,**1**)	**0.66**
`BRANCH-COMPACT`	(0,0,0)	0.00	(0,0,**1**)	**0.63**	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	0.00
`PARTITION`	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	0.00
`HYBRID`	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,**1**)	0.00	(0,0,**1**)	0.00	(0,0,**1**)	0.00	(0,0,**1**)	0.00

cellaneous heuristics are displayed black. We observe that, globally, `BRANCH-CONST` and `NODE` achieved the best aggregated joint lower bound scores, i. e., exhibited the best tradeoffs between tightness of the obtained lower bound, runtime, and classification coefficient (cf. Figure 10a). `NODE` and `BRANCH-CONST` also performed best in terms of runtime (cf. Figure 10c). In terms of tightness of the obtained lower bound, the LP based heuristics `ADJ-IP` performed best, followed by `BRANCH-TIGHT` and `F2` (cf. Figure 10b). `BRANCH-TIGHT` and `ADJ-IP` also were the best performing heuristics w. r. t. the classification coefficient (cf. Figure 10d).

The Figures 17 to 22 in Appendix B show the dominance graphs induced by the relation $\succ_{LB}$ for each dataset and hence visualize the results in more detail.

9.6 Upper bounds

Figure 11 shows the average upper bounds and runtimes of all heuristics that compute upper bounds. Instantiations of `LS-GED` usually provided the tightest upper bounds at the price of large execution times. Instantiations of `LP-GED` (except `COMPACT-MIP`) also yielded low upper bounds but were even slower. Finally, the paradigm `LSAPE-GED` represents the category with the largest variations. Its instantiations usually required low execution times (except `RING`, `SUBGRAPH`, and `WALKS`), but the provided upper bounds greatly depend on the intrinsic difficulty of the datasets.

Table 8 shows the aggregated joint upper bound scores $\widehat{s_{UB}}(\mathtt{H})$ and $\widehat{s_{UB}}(\mathtt{E})$, as well as the indicator vectors $\chi_{UB}(\mathtt{H})$ and $\chi_{UB}(\mathtt{E})$. The `LS-GED` instantiation `IPFP` was Pareto optimal on all datasets, as it always computed the tightest upper bound. The instantiation `NODE` of `LSAPE-GED` was Pareto optimal and the fastest heuristic on five out of six datasets and also achieved very high aggregated joint upper bound scores on these datasets. The instantiation `REFINE` of `LS-GED` performed well, too, as it was Pareto optimal on all datasets except for GREC. On the negative side, we see that the instantiations of `LP-GED` and the miscellaneous heuristics performed very poorly, as they were almost always dominated by other heuristics.

In Figure 12, the results are further aggregated by averaging the scores and summing the indicator vectors over all datasets. Instantiations and extensions of `LSAPE-GED` are displayed white, instantiations of `LP-GED` are displayed light grey, instantiations and extensions of `LS-GED` are displayed dark grey, and miscellaneous heuristics are displayed black. We observe that, globally, `NODE`, `IPFP`, and `REFINE` achieved the best aggregated joint upper bound scores, i. e., exhibited the best tradeoffs between tightness of the obtained upper bound, runtime, and classification coefficient (cf. Figure 12a). In terms of runtime, `NODE` and `BRANCH-CONST` performed best (cf. Figure 12c). In terms of classification coefficient and tightness, the instantiations of `LS-GED` performed best, with `IPFP` as the best performing heuristic among them (cf. Figure 12b and Figure 12d).

The average aggregated joint upper bound scores of both extensions `CENTRALITIES` and `MULTI-SOL` of the paradigm `LSAPE-GED` turned out to be smaller than 0.5 (cf. Figure 12a). That is, on average, instantiations of `LSAPE-GED` did not

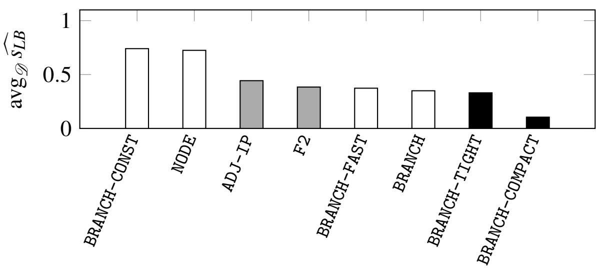
Table 8 Overview of results for upper bounds. For each heuristic H and each dataset, a non-zero aggregated joint upper bound score (displayed bold) implies that there was a Pareto optimal configuration of H. For each extension E and each dataset, a non-zero aggregated joint upper bound score (displayed bold) means that at least one algorithm using E was Pareto optimal. An aggregated joint upper bound score greater than 0.5 means that, on average, the heuristics instantiating the paradigm extended by E benefited from E.

heuristic	LETTER (H)		MUTA		AIDS		PROTEIN		FP		GREC	
	χ_{UB}	$\widehat{s_{UB}}$	χ_{UB}	$\widehat{s_{UB}}$	χ_{UB}	$\widehat{s_{UB}}$	χ_{UB}	$\widehat{s_{UB}}$	χ_{UB}	$\widehat{s_{UB}}$	χ_{UB}	$\widehat{s_{UB}}$
instantiations of the paradigm LSAPE-GED												
NODE	(0,0,0)	0.00	(0,**1**,**1**)	**0.94**	(0,**1**,0)	**0.89**	(0,**1**,**1**)	**0.99**	(0,**1**,0)	**0.92**	(0,**1**,0)	**0.98**
BP	(0,0,0)	0.00	(0,0,**1**)	0.00	(0,0,0)	0.00	(0,0,**1**)	0.00	(0,0,0)	0.00	(0,0,0)	0.00
BRANCH	(0,0,0)	0.00	(0,0,**1**)	0.00	(0,0,0)	0.00	(0,0,**1**)	**0.67**	(0,0,0)	0.00	(0,0,0)	**0.68**
BRANCH-FAST	(0,0,0)	**0.78**	(0,0,**1**)	0.00	(0,0,0)	0.00	(0,0,**1**)	0.00	(0,0,0)	**0.65**	(0,0,0)	**0.76**
BRANCH-CONST	(0,**1**,0)	**0.93**	(0,0,**1**)	**0.68**	(0,0,0)	0.00	(0,0,**1**)	**0.75**	(0,0,0)	0.00	(0,0,0)	**0.81**
STAR	(0,0,0)	0.00	(0,0,**1**)	0.00	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	**0.70**	(0,0,0)	0.00
SUBGRAPH	(0,0,0)	0.00	(0,0,**1**)	0.00	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	0.00
WALKS	(0,0,0)	0.00	(0,0,**1**)	0.00	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	0.00
RING$^{\text{OPT}}$	(0,0,0)	**0.63**	(0,0,**1**)	0.00	(0,0,0)	0.00	(0,0,**1**)	0.00	(0,0,0)	0.00	(0,0,0)	0.00
RING$^{\text{MS}}$	(0,0,0)	0.00	(0,0,**1**)	**0.63**	(0,0,0)	**0.58**	(0,0,**1**)	0.00	(0,0,0)	**0.64**	(0,0,0)	0.00
RING-ML$^{\text{1-SVM}}$	(0,0,**1**)	0.00	(0,0,**1**)	0.00	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	0.00
RING-ML$^{\text{DNN}}$	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	0.00
PREDICT$^{\text{1-SVM}}$	(0,0,**1**)	**0.55**	(0,0,**1**)	0.00	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	0.00
PREDICT$^{\text{DNN}}$	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	0.00
extensions of the paradigm LSAPE-GED												
MULTI-SOL	(0,0,0)	0.00	(0,0,**1**)	**0.46**	(0,0,0)	**0.59**	(0,0,**1**)	**0.62**	(0,0,0)	**0.38**	(0,0,0)	**0.62**
CENTRALITIES	(0,0,0)	**0.48**	(0,0,**1**)	**0.58**	(0,0,0)	**0.44**	(0,0,**1**)	**0.49**	(0,0,0)	**0.41**	(0,0,0)	**0.58**
instantiations of the paradigm LP-GED												
F1	(0,0,0)	0.00	(0,0,**1**)	0.00	(0,0,0)	0.00	(0,0,**1**)	0.00	(0,0,0)	0.00	(0,0,0)	0.00
F2	(0,0,0)	0.00	(0,0,**1**)	0.00	(0,0,0)	0.00	(0,0,**1**)	0.00	(0,0,0)	0.00	(0,0,0)	0.00
COMPACT-MIP	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	0.00	(0,0,0)	0.00
ADJ-IP	(0,0,0)	0.00	(0,0,**1**)	0.00	(0,0,0)	0.00	(0,0,**1**)	0.00	(0,0,0)	0.00	(0,0,0)	0.00
instantiations of the paradigm LS-GED												
REFINE	(**1**,0,0)	**0.64**	(0,0,**1**)	**0.66**	(0,0,0)	**0.64**	(0,0,**1**)	**0.66**	(0,0,**1**)	**0.67**	(0,0,**1**)	0.00
K-REFINE	(**1**,0,0)	**0.63**	(0,0,**1**)	0.00	(0,0,0)	0.00	(0,0,**1**)	0.00	(0,0,**1**)	0.00	(0,0,**1**)	0.00
BP-BEAM	(**1**,0,0)	**0.62**	(0,0,0)	**0.30**	(0,0,0)	**0.60**	(0,0,**1**)	0.00	(0,0,0)	**0.63**	(0,0,**1**)	0.00
IBP-BEAM	(**1**,0,0)	0.00	(0,0,**1**)	0.00	(0,0,0)	0.00	(0,0,**1**)	0.00	(0,0,**1**)	0.00	(0,0,**1**)	0.00
IPFP	(**1**,0,0)	**0.63**	(**1**,0,**1**)	**0.67**	(**1**,0,**1**)	**0.67**	(**1**,0,**1**)	**0.67**	(**1**,0,**1**)	**0.67**	(**1**,0,**1**)	**0.67**
extensions of the paradigm LS-GED												
MULTI-START	(**1**,0,0)	**0.40**	(**1**,0,**1**)	**0.91**	(**1**,0,**1**)	**0.85**	(**1**,0,**1**)	**0.80**	(**1**,0,**1**)	**0.68**	(**1**,0,**1**)	**0.83**
RANDPOST	(**1**,0,0)	0.00	(**1**,0,**1**)	**0.29**	(**1**,0,**1**)	**0.32**	(**1**,0,**1**)	**0.40**	(**1**,0,**1**)	0.00	(**1**,0,**1**)	**0.17**
miscellaneous heuristics												
BRANCH-TIGHT	(0,0,0)	**0.62**	(0,0,**1**)	0.00	(0,0,0)	0.00	(0,0,**1**)	0.00	(0,0,0)	0.00	(0,0,0)	0.00
SA	(0,0,0)	0.00	(0,0,**1**)	0.00	(0,0,0)	0.00	(0,0,**1**)	0.00	(0,0,0)	0.00	(0,0,0)	0.00

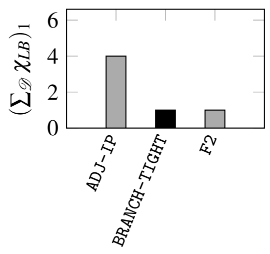
benefit from the extensions CENTRALITIES and MULTI-SOL. However, on each dataset, some instantiations of LSAPE-GED did benefit from the extensions, as some algorithms using CENTRALITIES and MULTI-SOL were Pareto optimal on almost all datasets (cf. Table 8).

We also observe that the average aggregated joint upper bound scores of the extensions MULTI-START and RANDPOST of the paradigm LS-GED are, respectively, clearly larger and clearly smaller than 0.5 (cf. Figure 12a). That is, on average, instantiations of LS-GED benefited from MULTI-START but not from RANDPOST. However, RANDPOST still turned out to be used by Pareto optimal algorithms on all datasets except for the datasets LETTER (H) and FP, which contain very small graphs. MULTI-START was used by Pareto optimal algorithms on all datasets (cf. Table 8). Moreover, we see that, on all six datasets, algorithms using MULTI-START and RANDPOST yielded the tightest upper bounds and the best classification coefficients (cf. Figure 12b and Figure 12d).

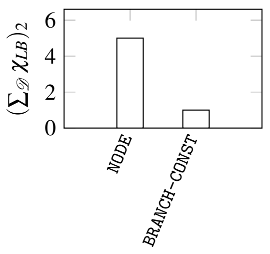
The Figures 23 to 28 in Appendix B show the dominance graphs induced by the relation $\succ_{UB}$ for each dataset and hence visualize the results in more detail.



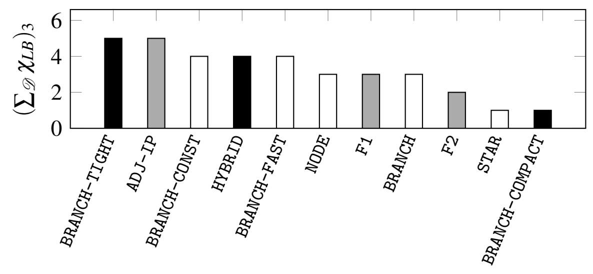
(a) Average aggregated joint lower bound scores.



(b) Optimality w. r. t. tightness of lower bound.



(c) Optimality w. r. t. runtime among heuristics producing lower bounds.



(d) Optimality w. r. t. lower bound classification coefficient.

Fig. 10 Average aggregated joint lower bound scores and numbers of datasets where heuristics are optimal w. r. t. tightness of lower bound, runtime, and lower bound classification coefficient, respectively. Only non-zero statistics are displayed.

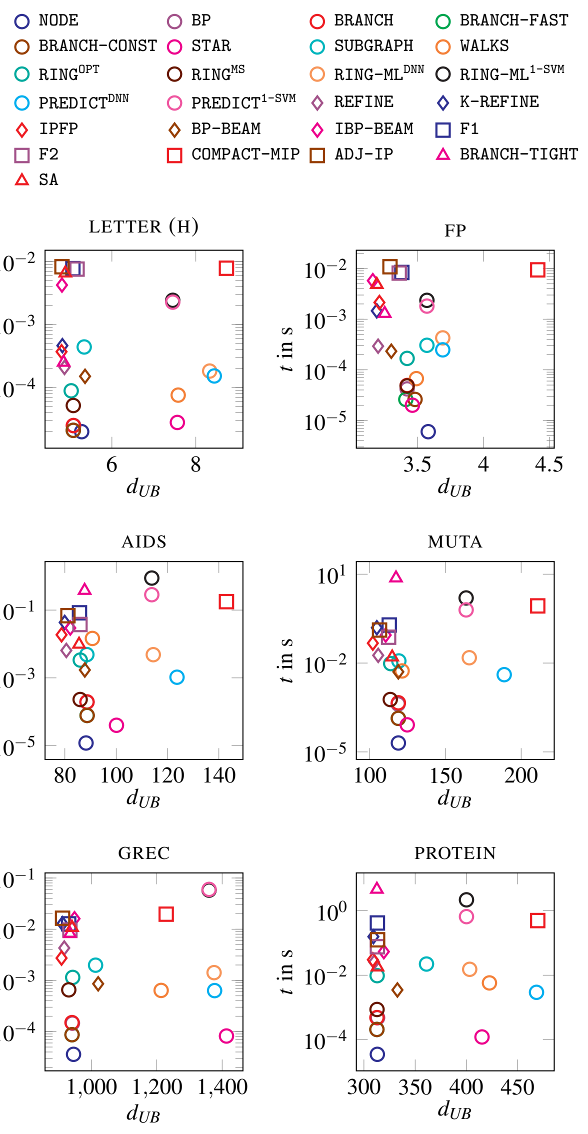


Fig. 11 Average upper bounds vs. average runtime. Instantiations of LSAPE-GED, LP-GED, and LS-GED are displayed as circles, squares, and diamonds, respectively; miscellaneous methods are displayed as triangles. For each heuristic H, the results are displayed only for the configuration that does not use any extensions.

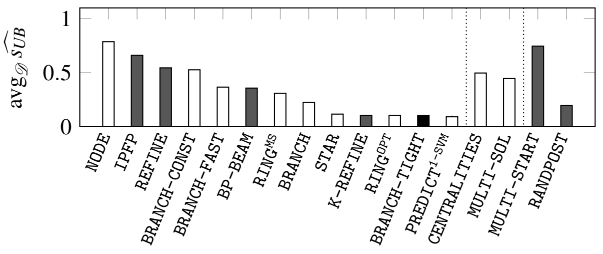
9.7 Gaps between lower and upper bounds

Table 9 shows the tightest average lower and upper bounds $d^{\star}_{LB}$ and $d^{\star}_{UB}$ for all datasets and the gaps between them. We see that the best upper bounds overestimate the best lower bounds (and hence, a fortiori, the exact GED) by at most 4.23 % and only 1.99 % on average. Given the hardness of exactly computing GED (cf. Section 1), this is a remarkable result. On all datasets, $d^{\star}_{UB}$ was computed by IPFP (cf. Section 9.6). On FP, $d^{\star}_{LB}$ was computed by BRANCH-TIGHT; on PROTEIN, it was computed by F2; and on all other datasets, it was computed by ADJ-IP (cf. Section 9.5).

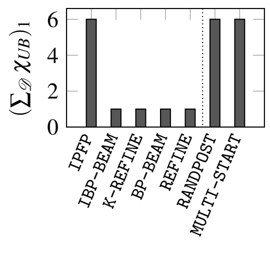
9.8 Effect of graph sizes

We also carried out experiments for evaluating the effect of the graph sizes on the compared methods. For this, we partitioned the datasets AIDS, MUTA, and PROTEIN that also contain larger graphs into subsets of graphs whose numbers of nodes is between 1 and 10, 11 and 20, and so forth. Subsequently, we randomly sampled 10 graphs from each subset with at least 10 graphs, and, for each sample $\mathscr{D}_{\text{sample}}$, ran all compared methods on all pairs of graphs $(G,H) \in \mathscr{D}_{\text{sample}} \times \mathscr{D}_{\text{sample}}$.

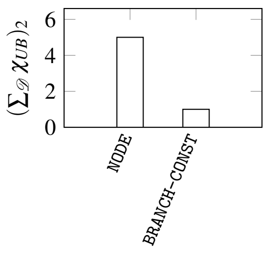
Figure 13 shows the observed trends for the average runtimes t and the average lower and upper bounds d_{LB} and d_{UB}. In order not to overcrowd the plots, trends are displayed only for the five methods with the best average aggregated joint lower and upper bound scores: BRANCH-CONST, NODE,



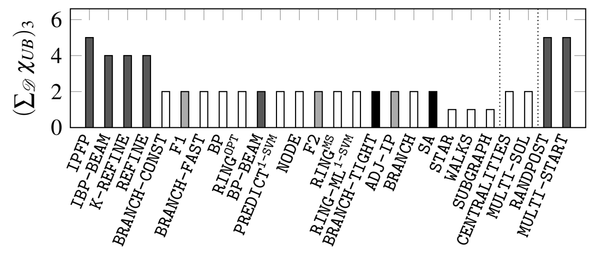
(a) Average aggregated joint upper bound scores.



(b) Optimality w. r. t. tightness of upper bound.



(c) Optimality w. r. t. runtime among heuristics producing upper bounds.



(d) Optimality w. r. t. upper bound classification coefficient.

Fig. 12 Average aggregated joint upper bound scores and numbers of datasets where heuristics are optimal w. r. t. tightness of upper bound, runtime, and upper bound classification coefficient, respectively. Only non-zero statistics are displayed.

Table 9 Tightest average lower and upper bounds.

dataset	$d^{\star}_{LB}$	$d^{\star}_{UB}$	gap in %
AIDS	73.45	76.18	3.58
MUTA	93.76	97.90	4.23
PROTEIN	302.80	307.65	1.58
LETTER (H)	4.72	4.75	0.63
GREC	898.83	904.70	0.65
FP	3.04	3.08	1.30
average	—	—	1.99

ADJ-IP, F2, and BRANCH-FAST for d_{LB} (cf. Figure 10a), and NODE, IPFP, REFINE, BRANCH-CONST, and BRANCH-FAST for d_{UB} (cf. Figure 12a). As expected, instantiations of LSAPE-GED were faster than instantiations of LS-GED, which, in turn, were faster than instantiations of LP-GED. We also see that, for all three datasets, there are hardly any crossing points between the trends for the lower and upper bounds. This is interesting, because it means that the graph sizes have little effect on the question which of two heuristic $\mathtt{H}_1$ and $\mathtt{H}_2$ yields the tighter lower or upper bound. Finally, we observe that, on PROTEIN, the gap between the tightest average lower and upper bounds is very narrow across all graph sizes. On AIDS and MUTA, the gaps grow with increasing graph sizes, but are still moderate also on the samples that contain the largest graphs (also cf. Table 9).

Table 10 Maximum and average lower bound classification coefficients for all datasets, and slopes and p-values of the linear regression models $c_{LB} \sim d_{LB}$.

dataset	$c^{\star}_{LB}$	avg $c_{LB}(\mathtt{ALG})$	m_{LB}	p_{LB}
AIDS	0.15	0.13	$1.11 \cdot 10^{-3}$	$2.51 \cdot 10^{-5}$
MUTA	0.01	0.00	$-2.33 \cdot 10^{-5}$	$4.76 \cdot 10^{-1}$
PROTEIN	0.04	0.03	$6.27 \cdot 10^{-5}$	$1.07 \cdot 10^{-4}$
LETTER (H)	0.29	0.23	$4.16 \cdot 10^{-2}$	$2.87 \cdot 10^{-9}$
GREC	0.37	0.32	$1.97 \cdot 10^{-4}$	$9.29 \cdot 10^{-5}$
FP	0.12	0.09	$3.38 \cdot 10^{-2}$	$1.11 \cdot 10^{-9}$
average	0.16	0.14	$1.28 \cdot 10^{-2}$	—

9.9 Classification coefficients vs. tightness of lower and upper bounds

Figure 14 and Table 10 relate the lower bounds of the algorithms producing lower bounds to the obtained lower bound classification coefficients. Figure 14 contains plots for all datasets. In each of them, each black dot represents an algorithm that yields a lower bound and the grey line visualizes the obtained linear regression model $c_{LB} \sim d_{LB}$. For each dataset, Table 10 summarizes the slopes and p-values of the models, as well as the maximum and average lower bound classification coefficients.

We observe that, on MUTA, all obtained classification coefficients either equal 0.00 or 0.01. This can be explained by the fact that, for both of its classes, MUTA contains graphs of very different sizes, which leads to a small difference between intra- and inter-class distances. As we have $c_{LB}(\mathtt{ALG}) \in \{0.00, 0.01\}$ for all algorithms ALG that produce lower bounds, the obtained linear regression model has a very high p-value and hence is not statistically significant.

For all other datasets, the obtained linear regression models have p-values smaller than 10^{-3} and are hence highly significant. Furthermore, all linear regression models except the statistically insignificant model for MUTA have a positive slope. That is, tight lower bounds tend to go hand in hand with good classification coefficients.

Figure 15 and Table 11 relate the upper bounds of the algorithms producing upper bounds to the obtained upper bound classification coefficients. Figure 15 contains plots for all datasets. In each of them, each black dot represents an algorithm that yields an upper bound and the grey line visualizes the obtained linear regression model $c_{UB} \sim d_{UB}$.

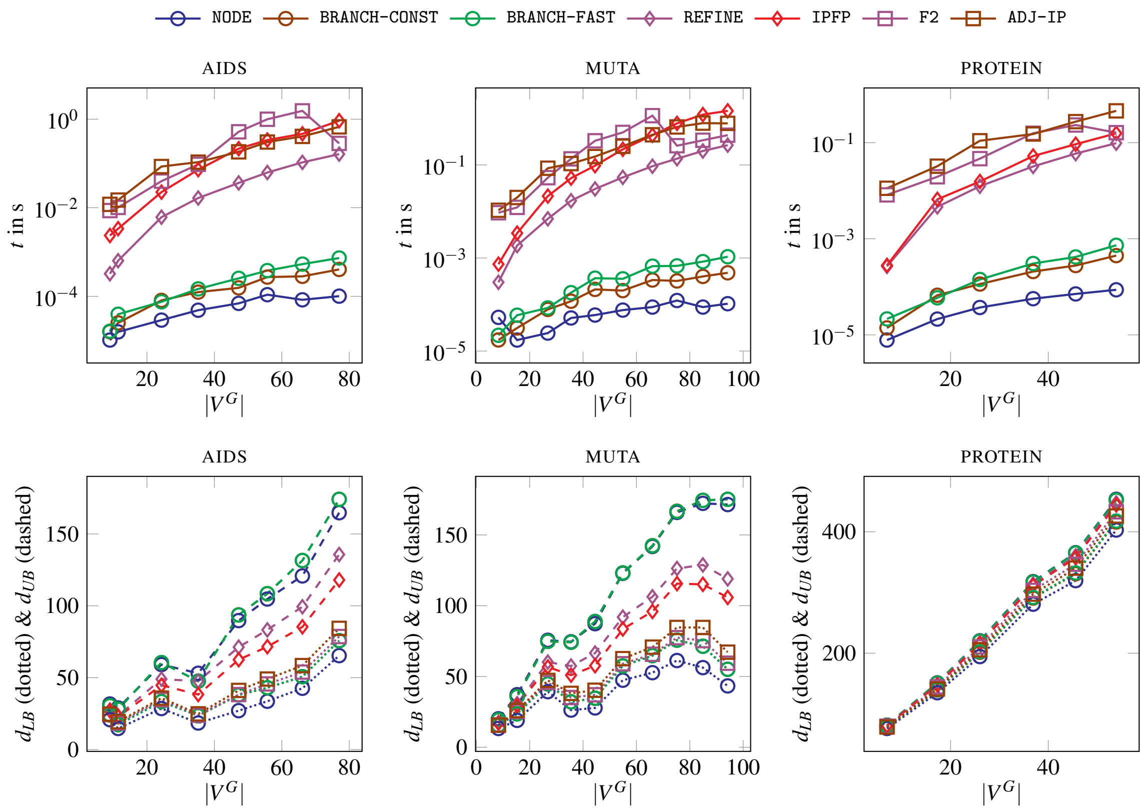


Fig. 13 ffect of graph sizes on methods with best average aggregated joint lower and upper bound scores. For each heuristic `H`, the results are displayed only for the configuration that does not use any extensions.

For each dataset, Table 11 summarizes the slopes and p-values of the models, as well as the maximum and average upper bound classification coefficients.

We again note that, on MUTA, all obtained classification coefficients either equal 0.00 or 0.01. Since we tested many more algorithms that compute upper bounds than algorithms that yield lower bounds,[4] the linear regression model $c_{UB} \sim d_{UB}$ for MUTA nonetheless has a p-value smaller than 10^{-3} and is hence still highly statistically significant. However, its p-value is much larger than the p-values of the linear regression models $c_{UB} \sim d_{UB}$ we obtained for the other datasets.

We observe that, for all datasets, the slopes of the linear regression models $c_{UB} \sim d_{UB}$ are positive. We can hence draw the same conclusion as for the lower bounds, namely, that tight upper bounds tend to go hand in hand with good classification coefficients. These findings allow us to positively answer the meta-question Q1 raised in the introduction: It is indeed beneficial to use GED as a guidance for the design of graph distance measures that are to be used within pattern recognition frameworks.

[4] To be precise, we tested 19 algorithms that compute lower bounds and 173 algorithms that compute upper bounds. The reason for this is that the extensions of the paradigms `LSAPE-GED` and `LS-GED` only affect the upper bounds.

Table 11 Maximum and average upper bound classification coefficients for all datasets, and slopes and p-values of the linear regression models $c_{UB} \sim d_{UB}$.

dataset	$c^{\star}_{UB}$	avg $c_{UB}(\mathtt{ALG})$	m_{UB}	p_{UB}
AIDS	0.15	0.11	$-1.95 \cdot 10^{-3}$	$5.47 \cdot 10^{-136}$
MUTA	0.01	0.01	$-5.36 \cdot 10^{-5}$	$3.76 \cdot 10^{-5}$
PROTEIN	0.04	0.03	$-1.45 \cdot 10^{-4}$	$3.91 \cdot 10^{-43}$
LETTER (H)	0.33	0.25	$-4.97 \cdot 10^{-2}$	$3.67 \cdot 10^{-43}$
GREC	0.35	0.27	$-5.97 \cdot 10^{-4}$	$3.12 \cdot 10^{-82}$
FP	0.11	0.10	$-5.29 \cdot 10^{-2}$	$1.69 \cdot 10^{-35}$
average	0.17	0.13	$-1.76 \cdot 10^{-2}$	—

The second meta-question Q2 asked whether lower or upper bounds for GED are better suited for use as graph distance measures within classification frameworks. Since the classification coefficients induced by the lower and upper bounds turned out to be very similar, this question cannot be answered as straightforwardly as Q1. However, there is a tendency: While the average lower bound classification coefficients were slightly better than the average upper bound classification coefficients, the opposite can be observed for the maximum lower and upper bound classification coefficients. Moreover, on average, the slopes of the linear regression models $c_{UB} \sim d_{UB}$ are slightly steeper than the slopes of the

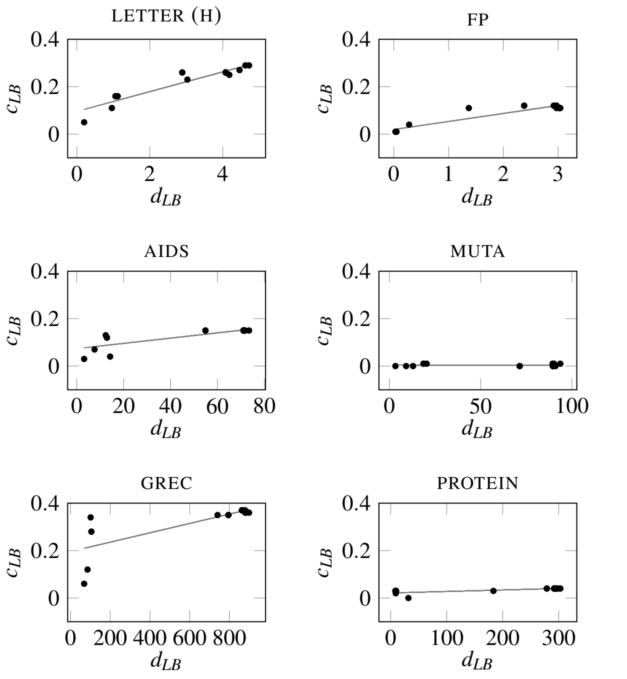


Fig. 14 Average lower bounds vs. lower bound classification coefficients. Each black dot represents one algorithm that computes a lower bound. The linear regression model $c_{LB} \sim d_{LB}$ is displayed in grey.

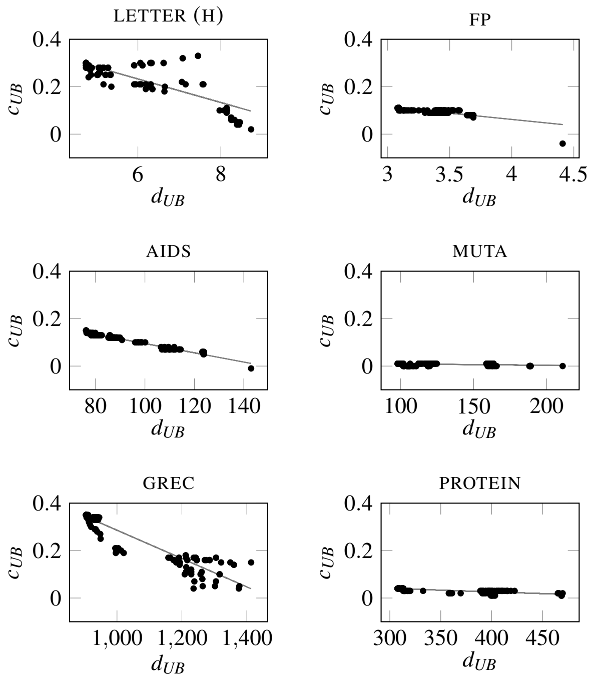


Fig. 15 Average upper bounds vs. upper bound classification coefficients. Each black dot represents one algorithm that computes an upper bound. The linear regression model $c_{UB} \sim d_{UB}$ is displayed in grey.

linear regression models $c_{LB} \sim d_{LB}$. Together, these observations suggest that the upper bound classification coefficients benefit more from tight upper bounds than the lower bound classification coefficients benefit from tight lower bounds. As a rule of thumb, we can hence conclude that tight upper bounds for GED (e. g., the upper bound computed by `IPFP`) should be used for classification purposes, if one is willing to invest a lot of time in the computation of the graph distance measure. Otherwise, a quickly computable lower bound such as the one produced by `BRANCH-CONST` should be employed.

10 Conclusions and future work

In this paper, we provided a systematic overview of the state of the art for heuristically computing GED. In total, we presented 30 different heuristics that were initially suggested in 28 different articles published from 2006 on. Thirteen heuristics were modeled as instantiations or extensions of the paradigm `LSAPE-GED`, which generalizes algorithms that upper and, possibly, lower bound GED via transformations to LSAPE. Four heuristics were modeled as instantiations of the paradigm `LP-GED`, which uses linear programming for computing lower and upper bounds for GED. Seven heuristics were modeled as instantiation or extensions of the paradigm `LS-GED`, a local search based approach for upper bounding GED. The remaining six heuristics do not fit within any of the suggested paradigms and were hence presented as miscellaneous heuristics.

We reimplemented all methods in C++ and empirically evaluated them by carrying out experiments on six different benchmark datasets. As the gap between tightest average lower bounds and tightest average upper bounds never exceeded 4.23 %, the experiments showed that despite the high theoretical complexity of approximating GED, on small to medium-sized graphs as the ones contained in the test datasets, GED can be bounded within tight margins.

On average, the instantiation `ADJ-IP` of `LP-GED` suggested in [36] computed the tightest lower bounds, followed by the `LP-GED` instantiation `F2` [44] and the miscellaneous heuristic `BRANCH-TIGHT` [9]. On all datasets, the tightest upper bounds were computed by the instantiation `IPFP` of `LS-GED` suggested in [7, 14, 16]. The instantiations `NODE` [36], `BRANCH-CONST` [70, 71], and `BRANCH-FAST` [8, 9] of `LSAPE-GED` achieved excellent tradeoffs between tightness, runtime, and classification coefficient — both w. r. t. the produced lower and w. r. t. the produced upper bounds.

Furthermore, we addressed a tacit assumption made in many publications on GED, which states that the tighter a lower or upper bound for GED, the better its performance when used as a graph distance measure within pattern recognition frameworks. Our experiments provided thorough evidence to support this assumption. They hence justify the ongoing competition for tight upper and lower bounds.

Given the small gaps between the tightest currently available lower and upper bounds for GED, we see little room for further tightening these bounds. Instead, we suggest that future work on the heuristic computation of GED should focus on the task of speeding up those existing heuristics that yield the tightest currently available bounds.

References

1. Abu-Aisheh, Z., Gaüzere, B., Bougleux, S., Ramel, J.Y., Brun, L., Raveaux, R., Héroux, P., Adam, S.: Graph edit distance contest 2016: Results and future challenges. Pattern Recognit. Lett. **100**, 96–103 (2017). DOI 10.1016/j.patrec.2017.10.007
2. Abu-Aisheh, Z., Raveaux, R., Ramel, J.: 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, *LNCS*, vol. 9069, pp. 138–147. Springer, Cham (2015). DOI 10.1007/978-3-319-18224-7_14
3. Babai, L.: Graph isomorphism in quasipolynomial time [extended abstract]. In: D. Wichs, Y. Mansour (eds.) STOC 2016, pp. 684–697. ACM (2016). DOI 10.1145/2897518.2897542
4. Blumenthal, D.B., Bougleux, S., Gamper, J., Brun, L.: Ring based approximation of graph edit distance. In: X. Bai, E. Hancock, T. Ho, R. Wilson, B. Biggio, A. Robles-Kelly (eds.) S+SSPR 2018, *LNCS*, vol. 11004, pp. 293–303. Springer, Cham (2018). DOI 10.1007/978-3-319-97785-0_28
5. Blumenthal, D.B., Bougleux, S., Gamper, J., Brun, L.: GEDLIB: A C++ library for graph edit distance computation. In: D. Conte, J.Y. Ramel, P. Foggia (eds.) GbRPR 2019, *LNCS*, vol. 11510, pp. 14–24. Springer, Cham (2019). DOI 10.1007/978-3-030-20081-7_2
6. Blumenthal, D.B., Bougleux, S., Gamper, J., Brun, L.: Upper bounding GED via transformations to LSAPE based on rings and machine learning. arXiv:1907.00203 [cs.DS] (2019)
7. Blumenthal, D.B., Daller, E., Bougleux, S., Brun, L., Gamper, J.: Quasimetric graph edit distance as a compact quadratic assignment problem. In: ICPR 2018, pp. 934–939. IEEE Computer Society (2018). DOI 10.1109/ICPR.2018.8546055
8. Blumenthal, D.B., Gamper, J.: Correcting and speeding-up bounds for non-uniform graph edit distance. In: ICDE 2017, pp. 131–134. IEEE Computer Society (2017). DOI 10.1109/ICDE.2017.57
9. Blumenthal, D.B., Gamper, J.: Improved lower bounds for graph edit distance. IEEE Trans. Knowl. Data Eng. **30**(3), 503–516 (2018). DOI 10.1109/TKDE.2017.2772243
10. Blumenthal, D.B., Gamper, J.: On the exact computation of the graph edit distance. Pattern Recognit. Lett. (2018). DOI 10.1016/j.patrec.2018.05.002. In press.
11. Bonacich, P.: Power and centrality: A family of measures. Am. J. Sociol. **92**(5), 1170–1182 (1987). DOI 10.1086/228631
12. Boria, N., Blumenthal, D.B., Bougleux, S., Brun, L.: Improved local search for graph edit distance. arXiv:1907.02929 [cs.DS] (2019)
13. Boria, N., Bougleux, S., Brun, L.: Approximating GED using a stochastic generator and multistart IPFP. In: X. Bai, E.R. Hancock, T.K. Ho, R.C. Wilson, B. Biggio, A. Robles-Kelly (eds.) S+SSPR 2018, pp. 460–469. Springer, Cham (2018). DOI 10.1007/978-3-319-97785-0_44
14. Bougleux, S., Brun, L., Carletti, V., Foggia, P., Gaüzère, B., Vento, M.: Graph edit distance as a quadratic assignment problem. Pattern Recognit. Lett. **87**, 38–46 (2017). DOI 10.1016/j.patrec.2016.10.001
15. Bougleux, S., Gaüzère, B., Blumenthal, D.B., Brun, L.: Fast linear sum assignment with error-correction and no cost constraints. Pattern Recognit. Lett. (2018). DOI 10.1016/j.patrec.2018.03.032
16. Bougleux, S., Gaüzère, B., Brun, L.: Graph edit distance as a quadratic program. In: ICPR 2016, pp. 1701–1706. IEEE Computer Society (2016). DOI 10.1109/ICPR.2016.7899881
17. Bougleux, S., Gaüzère, B., Brun, L.: A Hungarian algorithm for error-correcting graph matching. In: P. Foggia, C. Liu, M. Vento (eds.) GbRPR 2017, *LNCS*, vol. 10310, pp. 118–127. Springer, Cham (2017). DOI 10.1007/978-3-319-58961-9_11
18. Brin, S., Page, L.: The anatomy of a large-scale hypertextual web search engine. Comput. Netw. **30**(1-7), 107–117 (1998). DOI 10.1016/S0169-7552(98)00110-X
19. Brun, L., Foggia, P., Vento, M.: Trends in graph-based representations for pattern recognition. Pattern Recognit. Lett. (2018). DOI 10.1016/j.patrec.2018.03.016. In press.
20. Bunke, H., Allermann, G.: Inexact graph matching for structural pattern recognition. Pattern Recognit. Lett. **1**(4), 245–253 (1983). DOI 10.1016/0167-8655(83)90033-8
21. Carletti, V., Gaüzère, B., Brun, L., Vento, M.: Approximate graph edit distance computation combining bipartite matching and exact neighborhood substructure distance. In: C. Liu, B. Luo, W.G. Kropatsch, J. Cheng (eds.) GbRPR 2015, *LNCS*, vol. 9069, pp. 188–197. Springer, Cham (2015). DOI 10.1007/978-3-319-18224-7_19
22. Chang, C.C., Lin, C.J.: LIBSVM: A library for support vector machines. ACM Trans. Intell. Syst. Technol. **2**(3), 27 (2011). DOI 10.1145/1961189.1961199
23. Conte, D., Foggia, P., Sansone, C., Vento, M.: Thirty years of graph matching in pattern recognition. Int. J. Pattern Recognit. Artif. Intell. **18**(3), 265–298 (2004). DOI 10.1142/S0218001404003228
24. Cordella, L.P., Foggia, P., Sansone, C., Vento, M.: A (sub)graph isomorphism algorithm for matching large graphs. IEEE Trans. Pattern Anal. Mach. Intell. **26**(10), 1367–1372 (2004). DOI 10.1109/TPAMI.2004.75
25. Cortés, X., Serratosa, F., Moreno-García, C.F.: On the influence of node centralities on graph edit distance for graph classification. In: C. Liu, B. Luo, W.G. Kropatsch, J. Cheng (eds.) GbRPR 2015, *LNCS*, vol. 9069, pp. 231–241. Springer, Cham (2015). DOI 10.1007/978-3-319-18224-7_23
26. Daller, É., Bougleux, S., Gaüzère, B., Brun, L.: Approximate graph edit distance by several local searches in parallel. In: A. Fred, G.S. di Baja, M.D. Marsico (eds.) ICPRAM 2018, pp. 149–158. SciTePress (2018). DOI 10.5220/0006599901490158
27. Ferrer, M., Serratosa, F., Riesen, K.: A first step towards exact graph edit distance using bipartite graph matching. In: C. Liu, B. Luo, W.G. Kropatsch, J. Cheng (eds.) GbRPR 2015, *LNCS*, vol. 9069, pp. 77–86. Springer, Cham (2015). DOI 10.1007/978-3-319-18224-7_8
28. Fischer, A., Suen, C.Y., Frinken, V., Riesen, K., Bunke, H.: Approximation of graph edit distance based on Hausdorff matching. Pattern Recognit. **48**(2), 331–343 (2015). DOI 10.1016/j.patcog.2014.07.015
29. Foggia, P., Percannella, G., Vento, M.: Graph matching and learning in pattern recognition in the last 10 years. Int. J. Pattern Recognit. Artif. Intell. **28**(1), 1450001:1–1450001:40 (2014). DOI 10.1142/S0218001414500013
30. Frank, M., Wolfe, P.: An algorithm for quadratic programming. Nav. Res. Logist. Q. **3**(12), 95–110 (1956). DOI 10.1002/nav.3800030109
31. Gao, X., Xiao, B., Tao, D., Li, X.: A survey of graph edit distance. Pattern Anal. Appl. **13**(1), 113–129 (2010). DOI 10.1007/s10044-008-0141-y
32. Gaüzère, B., Bougleux, S., Riesen, K., Brun, L.: Approximate graph edit distance guided by bipartite matching of bags of walks. In: P. Fränti, G. Brown, M. Loog, F. Escolano, M. Pelillo (eds.) S+SSPR 2014, *LNCS*, vol. 8621, pp. 73–82. Springer, Cham (2014). DOI 10.1007/978-3-662-44415-3_8
33. Guennebaud, G., Jacob, B., et al.: Eigen v3 (2010). URL `http://eigen.tuxfamily.org`
34. Gurobi Optimization LLC: Gurobi Optimizer Reference Manual. URL `http://www.gurobi.com`
35. Henry, E.R.: Classification and Uses of Finger Prints. Routledge, London (1900)

36. Justice, D., Hero, A.: A binary linear programming formulation of the graph edit distance. IEEE Trans. Pattern Anal. Mach. Intell. **28**(8), 1200–1214 (2006). DOI 10.1109/TPAMI.2006.152
37. Karmarkar, N.: A new polynomial-time algorithm for linear programming. Combinatorica **4**(4), 373–396 (1984). DOI 10.1007/BF02579150
38. Kuhn, H.W.: The Hungarian method for the assignment problem. Nav. Res. Logist. Q. **2**(1-2), 83–97 (1955). DOI 10.1002/nav.3800020109
39. Le Digabel, S.: Algorithm 909: NOMAD: Nonlinear optimization with the MADS algorithm. ACM Trans. Math. Softw. **37**(4), 44:1–44:15 (2011). DOI 10.1145/1916461.1916468
40. Le Gall, F.: Powers of tensors and fast matrix multiplication. In: K. Nabeshima, K. Nagasaka, F. Winkler, Á. Szántó (eds.) ISSAC 2014, pp. 296–303. ACM (2014). DOI 10.1145/2608628.2608664
41. Lee, L., Lumsdaine, A., Siek, J.: The Boost Graph Library: User Guide and Reference Manual. Addison-Wesley Longman, Boston, MA (2002)
42. Leordeanu, M., Hebert, M., Sukthankar, R.: An integer projected fixed point method for graph matching and MAP inference. In: Y. Bengio, D. Schuurmans, J.D. Lafferty, C.K.I. Williams, A. Culotta (eds.) NIPS 2009, pp. 1114–1122. Curran Associates (2009)
43. Lerouge, J., Abu-Aisheh, Z., Raveaux, R., Héroux, P., Adam, S.: Exact graph edit distance computation using a binary linear program. In: A. Robles-Kelly, M. Loog, B. Biggio, F. Escolano, R. Wilson (eds.) S+SSPR 2016, *LNCS*, vol. 10029, pp. 485–495. Springer, Cham (2016). DOI 10.1007/978-3-319-49055-7_43
44. Lerouge, J., Abu-Aisheh, Z., Raveaux, R., Héroux, P., Adam, S.: New binary linear programming formulation to compute the graph edit distance. Pattern Recognit. **72**, 254–265 (2017). DOI 10.1016/j.patcog.2017.07.029
45. Lin, C.L.: Hardness of approximating graph transformation problem. In: D.Z. Du, X.S. Zhang (eds.) Algorithms and Computation, *LNCS*, vol. 834, pp. 74–82. Springer, Berlin, Heidelberg (1994). DOI 10.1007/3-540-58325-4_168
46. Munkres, J.: Algorithms for the assignment and transportation problems. SIAM J. Appl. Math. **5**(1), 32–38 (1957). DOI 10.1137/0105003
47. Nissen, S.: Implementation of a fast artificial neural network library (FANN). Tech. rep., Department of Computer Science, University of Copenhagen (2003). URL `http://fann.sourceforge.net/report/`
48. Ozdemir, E., Gunduz-Demir, C.: A hybrid classification model for digital pathology using structural and statistical pattern recognition. IEEE Trans. Med. Imaging **32**(2), 474–483 (2013). DOI 10.1109/TMI.2012.2230186
49. Riesen, K.: Structural Pattern Recognition with Graph Edit Distance. Advances in Computer Vision and Pattern Recognition. Springer, Cham (2015). DOI 10.1007/978-3-319-27252-8
50. Riesen, K., Bunke, H.: 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, *LNCS*, vol. 5342, pp. 287–297. Springer, Berlin, Heidelberg (2008). DOI 10.1007/978-3-540-89689-0_33
51. Riesen, K., Bunke, H.: Approximate graph edit distance computation by means of bipartite graph matching. Image Vis. Comput. **27**(7), 950–959 (2009). DOI 10.1016/j.imavis.2008.04.004
52. Riesen, K., Bunke, H.: Graph Classification and Clustering Based on Vector Space Embedding, *Series in Machine Perception and Artificial Intelligence*, vol. 77. World Scientific, Singapore (2010). DOI 10.1142/7731
53. Riesen, K., Bunke, H., Fischer, A.: Improving graph edit distance approximation by centrality measures. In: ICPR 2014, pp. 3910–3914. IEEE Computer Society (2014). DOI 10.1109/ICPR.2014.671
54. Riesen, K., Ferrer, M.: Predicting the correctness of node assignments in bipartite graph matching. Pattern Recognit. Lett. **69**, 8–14 (2016). DOI 10.1016/j.patrec.2015.10.007
55. Riesen, K., Ferrer, M., Fischer, A., Bunke, H.: Approximation of graph edit distance in quadratic time. In: C. Liu, B. Luo, W.G. Kropatsch, J. Cheng (eds.) GbRPR 2015, *LNCS*, vol. 9069, pp. 3–12. Springer, Cham (2015). DOI 10.1007/978-3-319-18224-7_1
56. Riesen, K., Fischer, A., Bunke, H.: Combining bipartite graph matching and beam search for graph edit distance approximation. In: N.E. Gayar, F. Schwenker, C. Suen (eds.) ANNPR 2014, *LNCS*, vol. 8774, pp. 117–128. Springer, Cham (2014). DOI 10.1007/978-3-319-11656-3_11
57. Riesen, K., Fischer, A., Bunke, H.: Computing upper and lower bounds of graph edit distance in cubic time. In: N.E. Gayar, F. Schwenker, C. Suen (eds.) ANNPR 2014, *LNCS*, vol. 8774, pp. 129–140. Springer, Heidelberg (2014). DOI 10.1007/978-3-319-11656-3
58. Riesen, K., Fischer, A., Bunke, H.: Improved graph edit distance approximation with simulated annealing. In: P. Foggia, C. Liu, M. Vento (eds.) GbRPR 2017, *LNCS*, vol. 10310, pp. 222–231. Springer, Cham (2017). DOI 10.1007/978-3-319-58961-9_20
59. Sanfeliu, A., Fu, K.S.: A distance measure between attributed relational graphs for pattern recognition. IEEE Trans. Syst. Man Cybern. **13**(3), 353–362 (1983). DOI 10.1109/TSMC.1983.6313167
60. Schomburg, I., Chang, A., Ebeling, C., Gremse, M., Heldt, C., Huhn, G., Schomburg, D.: BRENDA, the enzyme database: Updates and major new developments. Nucleic Acids Res. **32**(Database-Issue), 431–433 (2004). DOI 10.1093/nar/gkh081
61. Stauffer, M., Fischer, A., Riesen, K.: A novel graph database for handwritten word images. In: A. Robles-Kelly, M. Loog, B. Biggio, F. Escolano, R. Wilson (eds.) S+SSPR 2016, *LNCS*, vol. 10029, pp. 553–563. Springer, Cham (2016). DOI 10.1007/978-3-319-49055-7_49
62. Stauffer, M., Tschachtli, T., Fischer, A., Riesen, K.: A survey on applications of bipartite graph edit distance. In: P. Foggia, C. Liu, M. Vento (eds.) GbRPR 2017, *LNCS*, vol. 10310, pp. 242–252. Springer, Cham (2017). DOI 10.1007/978-3-319-58961-9_22
63. Strassen, V.: Gaussian elimination is not optimal. Numer. Math. **13**(4), 354–356 (1969). DOI 10.1007/BF02165411
64. Uno, T.: Algorithms for enumerating all perfect, maximum and maximal matchings in bipartite graphs. In: H.W. Leong, H. Imai, S. Jain (eds.) ISAAC 1997, *LNCS*, vol. 1350, pp. 92–101. Springer, Berlin, Heidelberg (1997). DOI 10.1007/3-540-63890-3_11
65. Uno, T.: A fast algorithm for enumerating bipartite perfect matchings. In: P. Eades, T. Takaoka (eds.) ISAAC 2001, *LNCS*, vol. 2223, pp. 367–379. Springer, Berlin, Heidelberg (2001). DOI 10.1007/3-540-45678-3_32
66. Vento, M.: A long trip in the charming world of graphs for pattern recognition. Pattern Recognit. **48**(2), 291–301 (2015). DOI 10.1016/j.patcog.2014.01.002
67. Wang, X., Ding, X., Tung, A.K.H., Ying, S., Jin, H.: An efficient graph indexing method. In: A. Kementsietsidis, M.A.V. Salles (eds.) ICDE 2012, pp. 210–221. IEEE Computer Society (2012). DOI 10.1109/ICDE.2012.28
68. Zeng, Z., Tung, A.K.H., Wang, J., Feng, J., Zhou, L.: Comparing stars: On approximating graph edit distance. PVLDB **2**(1), 25–36 (2009). DOI 10.14778/1687627.1687631
69. Zhao, X., Xiao, C., Lin, X., Zhang, W., Wang, Y.: Efficient structure similarity searches: a partition-based approach. VLDB J. **27**(1), 53–78 (2018). DOI 10.1007/s00778-017-0487-0
70. Zheng, W., Zou, L., Lian, X., Wang, D., Zhao, D.: Graph similarity search with edit distance constraint in large graph databases. In: Q. He, A. Iyengar, W. Nejdl, J. Pei, R. Rastogi (eds.) CIKM 2013, pp. 1595–1600. ACM (2013). DOI 10.1145/2505515.2505723
71. Zheng, W., Zou, L., Lian, X., Wang, D., Zhao, D.: Efficient graph similarity search over large graph databases. IEEE Trans. Knowl. Data Eng. **27**(4), 964–978 (2015). DOI 10.1109/TKDE.2014.2349924

A Datasets and edit cost functions

- *The datasets* AIDS *and* MUTA*:* Graphs contained in AIDS and MUTA represent molecular compounds. The molecules represented by the graphs contained in AIDS are divided into the class of molecules that do and the class of molecules that do not exhibit activity agains HIV. Similarly, the molecules represented by the graphs contained in MUTA are divided into the class of molecules that do and the class of molecules that do not cause genetic mutation. The nodes of the graphs contained in AIDS and MUTA are labeled with chemical symbols, and their edges are labeled with a valence (either 1 or 2). Node edit costs are defined as $c_V(\alpha,\alpha') := 5.5 \cdot \delta_{\alpha \neq \alpha'}$, $c_V(\alpha,\varepsilon) := 2.75$, and $c_V(\varepsilon,\alpha') := 2.75$, for all $(\alpha,\alpha') \in \Sigma_V \times \Sigma_V$. Edge edit costs are defined as $c_E(\beta,\beta') := 1.65 \cdot \delta_{\beta \neq \beta'}$, $c_E(\beta,\varepsilon) := 0.825$, and $c_E(\varepsilon,\beta') := 0.825$, for all $(\beta,\beta') \in \Sigma_E \times \Sigma_E$.
- *The dataset* PROTEIN*:* Graphs contained in PROTEIN represent proteins which are annotated with their EC classes (EC1, EC2, EC3, EC4, EC5, and EC6) [60]. Nodes are labeled with tuples (t,s), where t is the node's type (helix, sheet, or loop) and s is its amino acid sequence. Nodes are connected via structural or sequential edges or both, i. e., edges (u_i,u_j) are labeled with tuples (t_1,t_2), where t_1 is the type of the first edge connecting u_i and u_j and t_2 is the type of the second edge connecting u_i and u_j (possibly `null`). Node edit costs are defined as $c_V(\alpha,\alpha') := 16.5 \cdot \delta_{\alpha.t \neq \alpha'.t} + 0.75 \cdot \delta_{\alpha.t = \alpha'.t} \cdot \mathrm{LD}(\alpha.s,\alpha'.s))$, $c_V(\alpha,\varepsilon) := 8.25$, and $c_V(\varepsilon,\alpha') := 8.25$, for all $(\alpha,\alpha') \in \Sigma_V \times \Sigma_V$, where $\mathrm{LD}(\cdot,\cdot)$ is Levenshtein's string edit distance. Edge edit costs are defined as $c_E(\beta,\beta') := 0.25 \cdot \mathrm{LSAPE}(\mathbf{C}^{\beta,\beta'})$, $c_E(\beta,\varepsilon) := 0.25 \cdot f(\beta)$, and $c_E(\varepsilon,\beta') := 0.25 \cdot f(\beta')$, for all $(\beta,\beta') \in \Sigma_E \times \Sigma_E$, where $f(\beta) := 1 + \delta_{\beta.t_2 \neq \mathtt{null}}$ and $\mathbf{C}^{\beta,\beta'} \in \mathbb{R}^{(f(\beta)+1)\times(f(\beta')+1)}$ is constructed as $c^{\beta,\beta'}_{r,s} := 2 \cdot \delta_{\beta.t_r \neq \beta'.t_s}$ and $c^{\beta,\beta'}_{r,f(\beta')+1} := c^{\beta,\beta'}_{f(\beta)+1,s} := 1$, for all $(r,s) \in [f(\beta)] \times [f(\beta')]$.
- *The dataset* LETTER (H)*:* Graphs contained in LETTER (H) represent highly distorted drawings of the capital letters A, E, F, H, I, K, L, M, N, T, V, W, X, Y, and Z. Nodes are labeled with two-dimensional Euclidean coordinates. Edges are unlabeled. Node edit costs are defined as $c_V(\alpha,\alpha') := 0.75 \cdot \|\alpha - \alpha'\|$, $c_V(\alpha,\varepsilon) := 0.675$, and $c_V(\varepsilon,\alpha') := 0.675$, for all $(\alpha,\alpha') \in \Sigma_V \times \Sigma_V$, where $\|\cdot\|$ is the Euclidean norm. The edge edit costs c_E are defined as $c_E(1,\varepsilon) := c_E(\varepsilon,1) := 0.425$.
- *The dataset* GREC*:* Graphs contained in GREC represent 22 different symbols from electronic and architectural drawings. Nodes are labeled with tuples (t,x,y), where t equals one of four node types and (x,y) is a two-dimensional Euclidean coordinate. Nodes are connected via line or arc edges or both, i. e., edges (u_i,u_j) are labeled with tuples (t_1,t_2), where t_1 is the type of the first edge connecting u_i and u_j and t_2 is the type of the second edge connecting u_i and u_j (possibly `null`). Node edit costs are defined as $c_V(\alpha,\alpha') := 0.5 \cdot \|\alpha.(x,y) - \alpha'.(x,y)\| \cdot \delta_{\alpha.t = \alpha'.t} + 90 \cdot \delta_{\alpha.t \neq \alpha'.t}$, $c_V(\alpha,\varepsilon) := 45$, and $c_V(\varepsilon,\alpha') := 45$, for all $(\alpha,\alpha') \in \Sigma_V \times \Sigma_V$. Edge edit costs are defined as $c_E(\beta,\beta') := 0.5 \cdot \mathrm{LSAPE}(\mathbf{C}^{\beta,\beta'})$, $c_E(\beta,\varepsilon) := 0.5 \cdot f(\beta)$, and $c_E(\varepsilon,\beta') := 0.5 \cdot f(\beta')$, for all $(\beta,\beta') \in \Sigma_E \times \Sigma_E$, where $f(\beta) := 1 + \delta_{\beta.t_2 \neq \mathtt{null}}$ and $\mathbf{C}^{\beta,\beta'} \in \mathbb{R}^{(f(\beta)+1)\times(f(\beta')+1)}$ is constructed as $c^{\beta,\beta'}_{r,s} := 30 \cdot \delta_{\beta.t_r \neq \beta'.t_s}$ and $c^{\beta,\beta'}_{r,f(\beta')+1} := c^{\beta,\beta'}_{f(\beta)+1,s} := 15$ for all $(r,s) \in [f(\beta)] \times [f(\beta')]$.
- *The dataset* FP*:* Graphs contained in FP represent fingerprint images which are annotated with their classes (arch, left loop, right loop, and whorl) from the Galton-Henry classification system [35]. Nodes are unlabeled and edges are labeled with an orientation $\beta \in \mathbb{R}$ with $-\pi/2 < \beta \leq \pi/2$. Node edit costs are defined as $c_V(1,\varepsilon) := c_V(\varepsilon,1) := 0.525$. Edge edit costs are defined as $c_E(\beta,\beta') := 0.5 \cdot \min\{|\beta - \beta'|, \pi - |\beta - \beta'|\}$, $c_E(\beta,\varepsilon) := 0.375$, and $c_E(\varepsilon,\beta') := 0.375$, for all $(\beta,\beta') \in \Sigma_E \times \Sigma_E$.

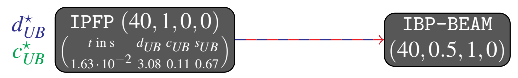

$d^\star_{UB}$
$c^\star_{UB}$
IPFP $(40,1,0,0)$
t in s d_{UB} c_{UB} s_{UB}
$1.63 \cdot 10^{-2}$ 3.08 0.11 0.67
IBP-BEAM $(40,0.5,1,0)$

Fig. 16 Snapshot of the dominance graph induced by $\succ_{UB}$ on the dataset FP shown in Figure 24.

B Visualization of experiments via dominance graphs

The Figures 17 to 28 visualize the transitive reductions of the dominance graphs induced by $\succ_{LB}$ (Figures 17 to 22) and $\succ_{UB}$ (Figures 23 to 28), and hence provide more detailed views on the results of the experiments reported in Section 9.5 and Section 9.6. In the dominance graphs, instantiations of `LSAPE-GED` are displayed black on white, instantiations of `LP-GED` are displayed black on light grey, instantiations of `LS-GED` are displayed white on dark grey, and miscellaneous heuristics are displayed white on black. For all algorithms instantiating `LSAPE-GED`, we display the configuration (K,γ) of the extensions `MULTI-SOL` and `CENTRALITIES` in addition to the name of the heuristic. Similarly, for all algorithms instantiating `LS-GED`, we display the configuration (K,ρ,L,η) of the extensions `MULTI-START` and `RANDPOST`. Recall that instantiations of `LSAPE-GED` are run without extensions just in case $(K,\gamma) = (1,0)$, and that instantiations of `LS-GED` are run without extensions just in case $(K,\rho,L,\eta) = (1,1,0,0)$ (cf. Section 9.2 for more details). For Pareto optimal algorithms, we also show the test metrics $d_{LB|UB}$, t, and $c_{LB|UB}$, and the joint score $s_{LB|UB}$.

As the extensions `MULTI-SOL` and `CENTRALITIES` of `LSAPE-GED` improve the computed upper bounds at the price of increased runtimes but have no effect on the obtained lower bounds, for all instantiations of `LSAPE-GED`, we only show the baseline configurations $(K,\gamma) = (1,0)$ in the dominance graphs induced by $\succ_{LB}$. In the dominance graphs induced by $\succ_{UB}$, for each heuristic `H`, we only display those configurations that are Pareto optimal (i. e., maximal w. r. t. $\succ_{UB}$) or have a maximal joint score s_{UB} among all tested configurations of `H`.

In the transitive reduction of the dominance graphs induced by $\succ_{LB}$, we draw an arc from $\mathtt{ALG}_1$ to $\mathtt{ALG}_2$ just in case $\mathtt{ALG}_1 \succ_{LB} \mathtt{ALG}_2$ and there is no algorithm $\mathtt{ALG}_3$ such that $\mathtt{ALG}_1 \succ_{LB} \mathtt{ALG}_3 \succ_{LB} \mathtt{ALG}_2$. Arcs are blue if, additionally, $\mathtt{ALG}_1$ yielded a tighter lower bound than $\mathtt{ALG}_2$, red if $\mathtt{ALG}_1$ was faster than $\mathtt{ALG}_2$, and green if $\mathtt{ALG}_1$ had a better classification coefficient than $\mathtt{ALG}_2$. Multicolored arcs indicate that several of these relations holds. The graphs are oriented from left to right, such that an algorithm is Pareto optimal just in case it appears in the left-most layer. The colored labels $d^\star_{LB}$, $t^\star_{LB}$, and $c^\star_{LB}$ highlight those Pareto optimal algorithms that, respectively, yielded the tightest lower bound, exhibited the best runtime behaviour among all heuristics that compute lower bounds, or gave the best lower bound classification coefficient. The dominance graphs induced by $\succ_{UB}$ are constructed analogously.

Example 3 Figure 16 exemplifies the visualizations of the dominance graphs induced by $\succ_{LB}$ and $\succ_{UB}$. It shows a snapshot of the dominance graph induced by $\succ_{UB}$ on the dataset FP shown in Figure 24. We see that `IPFP` run with the configuration $(K,\rho,L,\eta) = (40,1,0,0)$ was Pareto optimal on FP. The blue label $d^\star_{LB}$ indicates that `IPFP` $(40,1,0,0)$ computed the tightest average upper bound on FP; the green label $c^\star_{LB}$ tells us that it also yielded the best upper bound classification coefficient. Furthermore, we see that $t(\mathtt{IPFP}\,(40,1,0,0)) = 1.63 \cdot 10^{-2}\,\mathrm{s}$, $d_{UB}(\mathtt{IPFP}\,(40,1,0,0)) = 3.08$, $c_{UB}(\mathtt{IPFP}\,(40,1,0,0)) = 0.11$, and $s_{UB}(\mathtt{IPFP}\,(40,1,0,0)) = 0.67$.

The blue-red arc from `IPFP` $(40,1,0,0)$ to `IBP-BEAM` $(40,0.5,1,0)$ tells us that, on FP, `IPFP` with $(K,\rho,L,\eta) = (40,1,0,0)$ dominated `IBP-BEAM` with $(K,\rho,L,\eta) = (40,0.5,1,0)$. More precisely, we have $t(\mathtt{IBP\text{-}BEAM}\,(40,0.5,1,0)) > t(\mathtt{IPFP}\,(40,1,0,0)) = 1.63 \cdot 10^{-2}\,\mathrm{s}$, $d_{UB}(\mathtt{IBP\text{-}BEAM}\,(40,0.5,1,0)) > d_{UB}(\mathtt{IPFP}\,(40,1,0,0)) = 3.08$, and $c_{UB}(\mathtt{IBP\text{-}BEAM}\,(40,0.5,1,0)) = c_{LB}(\mathtt{IPFP}\,(40,1,0,0)) = 0.11$. As `IPFP` and `IBP-BEAM` instantiate `LS-GED`, they are shown white on dark grey.

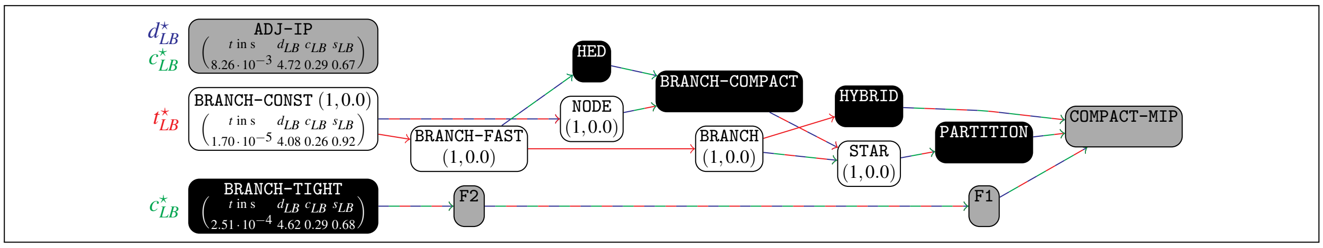


Fig. 17 Transitive reduction of dominance graph for lower bounds on the dataset LETTER (H).

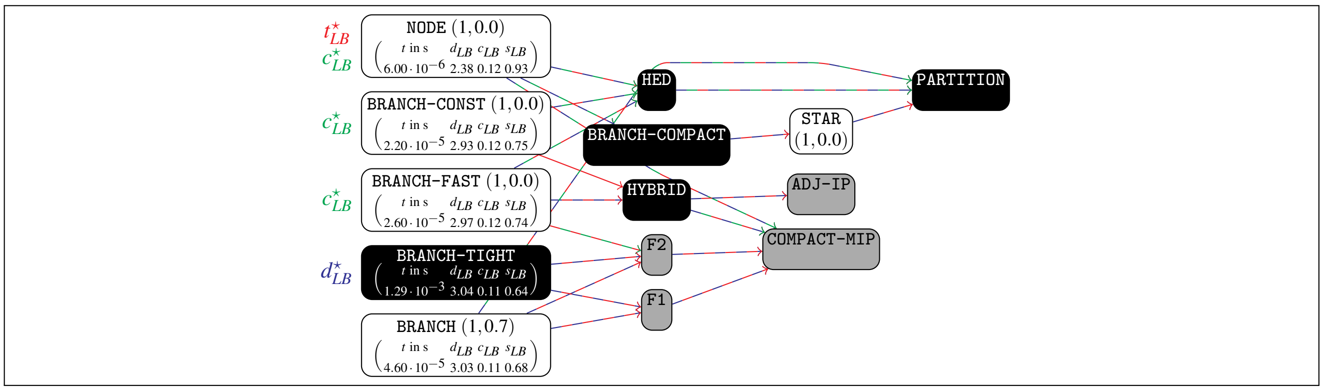

$t^\star_{LB}$ $c^\star_{LB}$
NODE (1,0.0)
t in s d_{LB} c_{LB} s_{LB}
$6.00\cdot10^{-6}$ 2.38 0.12 0.93
$c^\star_{LB}$
BRANCH-CONST (1,0.0)
$2.20\cdot10^{-5}$ 2.93 0.12 0.75
$c^\star_{LB}$
BRANCH-FAST (1,0.0)
$2.60\cdot10^{-5}$ 2.97 0.12 0.74
$d^\star_{LB}$
BRANCH-TIGHT
$1.29\cdot10^{-3}$ 3.04 0.11 0.64
BRANCH (1,0.7)
$4.60\cdot10^{-5}$ 3.03 0.11 0.68
HED
PARTITION
BRANCH-COMPACT
STAR (1,0.0)
HYBRID
ADJ-IP
COMPACT-MIP
F2
F1

Fig. 18 Transitive reduction of dominance graph for lower bounds on the dataset FP.

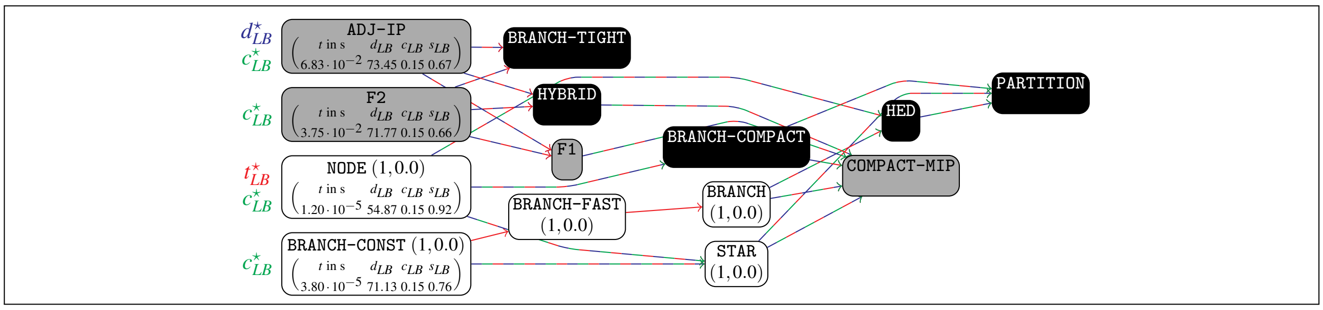

$d^\star_{LB}$ $c^\star_{LB}$
ADJ-IP
t in s d_{LB} c_{LB} s_{LB}
$6.83\cdot10^{-2}$ 73.45 0.15 0.67
$c^\star_{LB}$
F2
$3.75\cdot10^{-2}$ 71.77 0.15 0.66
$t^\star_{LB}$ $c^\star_{LB}$
NODE (1,0.0)
$1.20\cdot10^{-5}$ 54.87 0.15 0.92
$c^\star_{LB}$
BRANCH-CONST (1,0.0)
$3.80\cdot10^{-5}$ 71.13 0.15 0.76
BRANCH-TIGHT
HYBRID
F1
BRANCH-FAST (1,0.0)
BRANCH-COMPACT
BRANCH (1,0.0)
STAR (1,0.0)
HED
PARTITION
COMPACT-MIP

Fig. 19 Transitive reduction of dominance graph for lower bounds on the dataset AIDS.

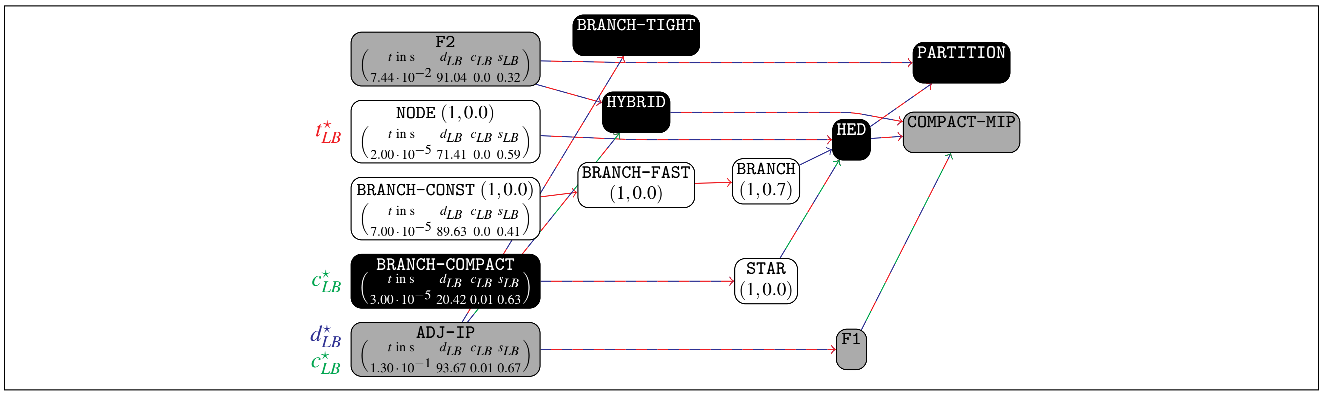
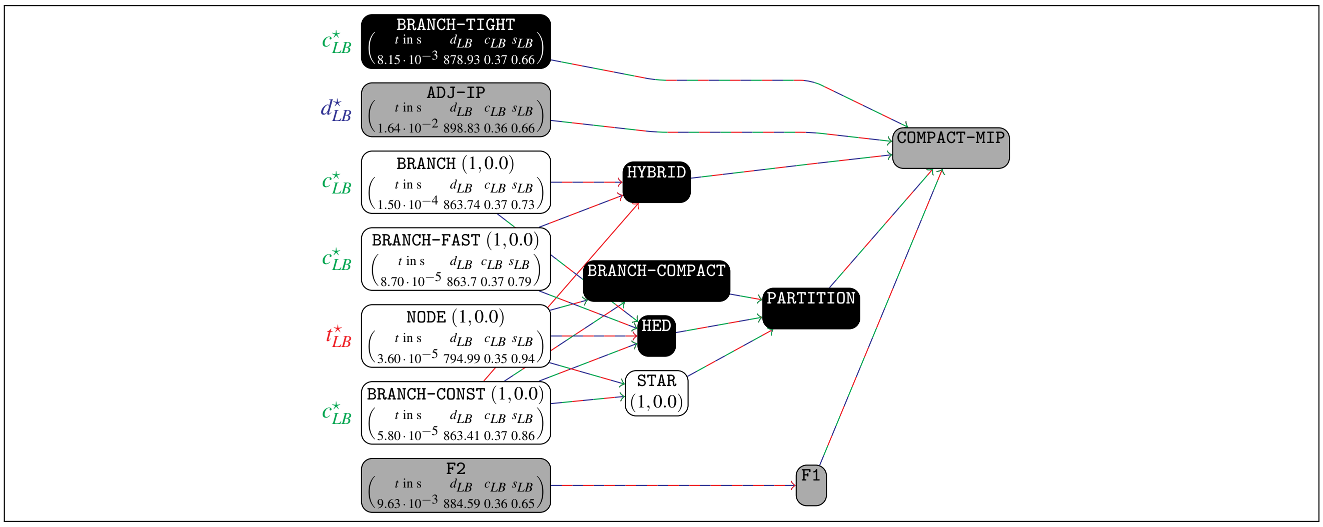
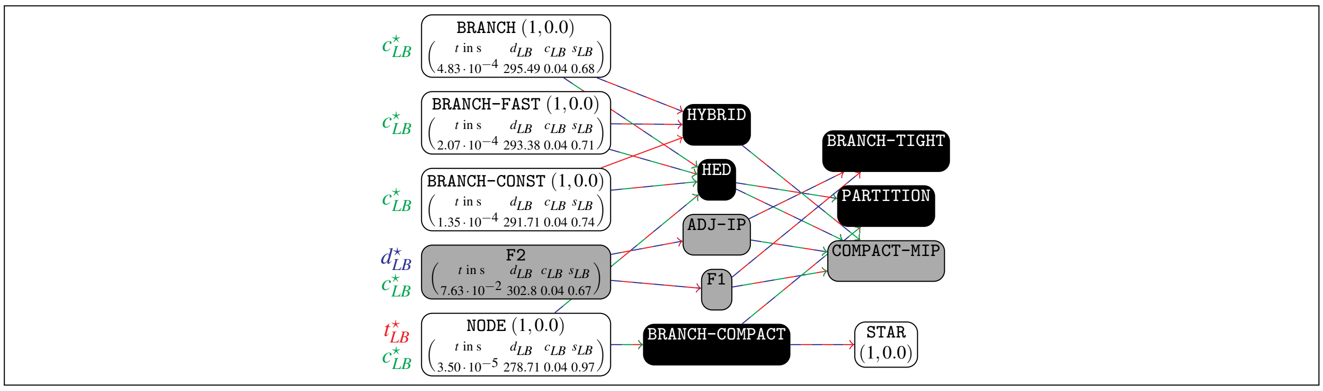

F2
t in s d_{LB} c_{LB} s_{LB}
$7.44\cdot10^{-2}$ 91.04 0.0 0.32
$t^\star_{LB}$
NODE (1,0.0)
$2.00\cdot10^{-5}$ 71.41 0.0 0.59
BRANCH-CONST (1,0.0)
$7.00\cdot10^{-5}$ 89.63 0.0 0.41
$c^\star_{LB}$
BRANCH-COMPACT
$3.00\cdot10^{-5}$ 20.42 0.01 0.63
$d^\star_{LB}$ $c^\star_{LB}$
ADJ-IP
$1.30\cdot10^{-1}$ 93.67 0.01 0.67
BRANCH-TIGHT
HYBRID
BRANCH-FAST (1,0.0)
BRANCH (1,0.7)
STAR (1,0.0)
HED
PARTITION
COMPACT-MIP
F1

Fig. 20 Transitive reduction of dominance graph for lower bounds on the dataset MUTA.



$c^\star_{LB}$ BRANCH-TIGHT t in s d_{LB} c_{LB} s_{LB} $8.15\cdot10^{-3}$ 878.93 0.37 0.66
$d^\star_{LB}$ ADJ-IP t in s d_{LB} c_{LB} s_{LB} $1.64\cdot10^{-2}$ 898.83 0.36 0.66
$c^\star_{LB}$ BRANCH $(1,0.0)$ t in s d_{LB} c_{LB} s_{LB} $1.50\cdot10^{-4}$ 863.74 0.37 0.73
$c^\star_{LB}$ BRANCH-FAST $(1,0.0)$ t in s d_{LB} c_{LB} s_{LB} $8.70\cdot10^{-5}$ 863.7 0.37 0.79
$t^\star_{LB}$ NODE $(1,0.0)$ t in s d_{LB} c_{LB} s_{LB} $3.60\cdot10^{-5}$ 794.99 0.35 0.94
$c^\star_{LB}$ BRANCH-CONST $(1,0.0)$ t in s d_{LB} c_{LB} s_{LB} $5.80\cdot10^{-5}$ 863.41 0.37 0.86
F2 t in s d_{LB} c_{LB} s_{LB} $9.63\cdot10^{-3}$ 884.59 0.36 0.65
COMPACT-MIP
HYBRID
BRANCH-COMPACT
PARTITION
HED
STAR $(1,0.0)$
F1

Fig. 21 Transitive reduction of dominance graph for lower bounds on the dataset GREC.



$c^\star_{LB}$ BRANCH $(1,0.0)$ t in s d_{LB} c_{LB} s_{LB} $4.83\cdot10^{-4}$ 295.49 0.04 0.68
$c^\star_{LB}$ BRANCH-FAST $(1,0.0)$ t in s d_{LB} c_{LB} s_{LB} $2.07\cdot10^{-4}$ 293.38 0.04 0.71
$c^\star_{LB}$ BRANCH-CONST $(1,0.0)$ t in s d_{LB} c_{LB} s_{LB} $1.35\cdot10^{-4}$ 291.71 0.04 0.74
$d^\star_{LB}$ $c^\star_{LB}$ F2 t in s d_{LB} c_{LB} s_{LB} $7.63\cdot10^{-2}$ 302.8 0.04 0.67
$t^\star_{LB}$ $c^\star_{LB}$ NODE $(1,0.0)$ t in s d_{LB} c_{LB} s_{LB} $3.50\cdot10^{-5}$ 278.71 0.04 0.97
HYBRID
HED
ADJ-IP
F1
BRANCH-COMPACT
BRANCH-TIGHT
PARTITION
COMPACT-MIP
STAR $(1,0.0)$

Fig. 22 Transitive reduction of dominance graph for lower bounds on the dataset PROTEIN.

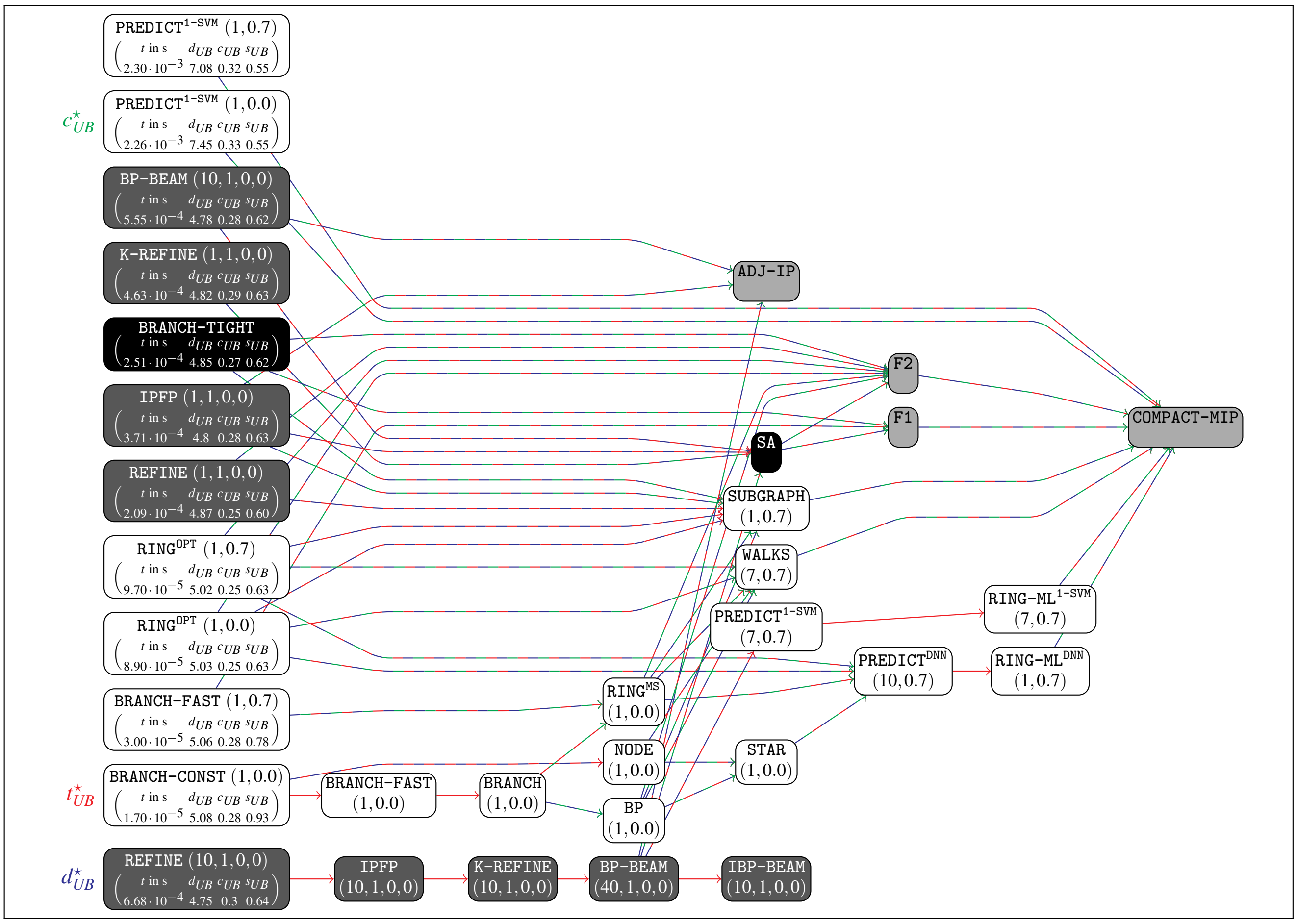

PREDICT$^{1\text{-SVM}}$ $(1,0.7)$ $\begin{pmatrix} t \text{ in s} & d_{UB} & c_{UB} & s_{UB} \\ 2.30\cdot10^{-3} & 7.08 & 0.32 & 0.55 \end{pmatrix}$
$c^\star_{UB}$
PREDICT$^{1\text{-SVM}}$ $(1,0.0)$ $\begin{pmatrix} t \text{ in s} & d_{UB} & c_{UB} & s_{UB} \\ 2.26\cdot10^{-3} & 7.45 & 0.33 & 0.55 \end{pmatrix}$
BP-BEAM $(10,1,0,0)$ $\begin{pmatrix} t \text{ in s} & d_{UB} & c_{UB} & s_{UB} \\ 5.55\cdot10^{-4} & 4.78 & 0.28 & 0.62 \end{pmatrix}$
K-REFINE $(1,1,0,0)$ $\begin{pmatrix} t \text{ in s} & d_{UB} & c_{UB} & s_{UB} \\ 4.63\cdot10^{-4} & 4.82 & 0.29 & 0.63 \end{pmatrix}$
BRANCH-TIGHT $\begin{pmatrix} t \text{ in s} & d_{UB} & c_{UB} & s_{UB} \\ 2.51\cdot10^{-4} & 4.85 & 0.27 & 0.62 \end{pmatrix}$
IPFP $(1,1,0,0)$ $\begin{pmatrix} t \text{ in s} & d_{UB} & c_{UB} & s_{UB} \\ 3.71\cdot10^{-4} & 4.8 & 0.28 & 0.63 \end{pmatrix}$
REFINE $(1,1,0,0)$ $\begin{pmatrix} t \text{ in s} & d_{UB} & c_{UB} & s_{UB} \\ 2.09\cdot10^{-4} & 4.87 & 0.25 & 0.60 \end{pmatrix}$
RING$^{\text{OPT}}$ $(1,0.7)$ $\begin{pmatrix} t \text{ in s} & d_{UB} & c_{UB} & s_{UB} \\ 9.70\cdot10^{-5} & 5.02 & 0.25 & 0.63 \end{pmatrix}$
RING$^{\text{OPT}}$ $(1,0.0)$ $\begin{pmatrix} t \text{ in s} & d_{UB} & c_{UB} & s_{UB} \\ 8.90\cdot10^{-5} & 5.03 & 0.25 & 0.63 \end{pmatrix}$
BRANCH-FAST $(1,0.7)$ $\begin{pmatrix} t \text{ in s} & d_{UB} & c_{UB} & s_{UB} \\ 3.00\cdot10^{-5} & 5.06 & 0.28 & 0.78 \end{pmatrix}$
$t^\star_{UB}$
BRANCH-CONST $(1,0.0)$ $\begin{pmatrix} t \text{ in s} & d_{UB} & c_{UB} & s_{UB} \\ 1.70\cdot10^{-5} & 5.08 & 0.28 & 0.93 \end{pmatrix}$
$d^\star_{UB}$
REFINE $(10,1,0,0)$ $\begin{pmatrix} t \text{ in s} & d_{UB} & c_{UB} & s_{UB} \\ 6.68\cdot10^{-4} & 4.75 & 0.3 & 0.64 \end{pmatrix}$
ADJ-IP
F2
F1
SA
COMPACT-MIP
SUBGRAPH $(1,0.7)$
WALKS $(7,0.7)$
PREDICT$^{1\text{-SVM}}$ $(7,0.7)$
RING-ML$^{1\text{-SVM}}$ $(7,0.7)$
PREDICT$^{\text{DNN}}$ $(10,0.7)$
RING-ML$^{\text{DNN}}$ $(1,0.7)$
RING$^{\text{MS}}$ $(1,0.0)$
NODE $(1,0.0)$
STAR $(1,0.0)$
BRANCH-FAST $(1,0.0)$
BRANCH $(1,0.0)$
BP $(1,0.0)$
IPFP $(10,1,0,0)$
K-REFINE $(10,1,0,0)$
BP-BEAM $(40,1,0,0)$
IBP-BEAM $(10,1,0,0)$

Fig. 23 Transitive reduction of dominance graph for upper bounds on the dataset LETTER (H).

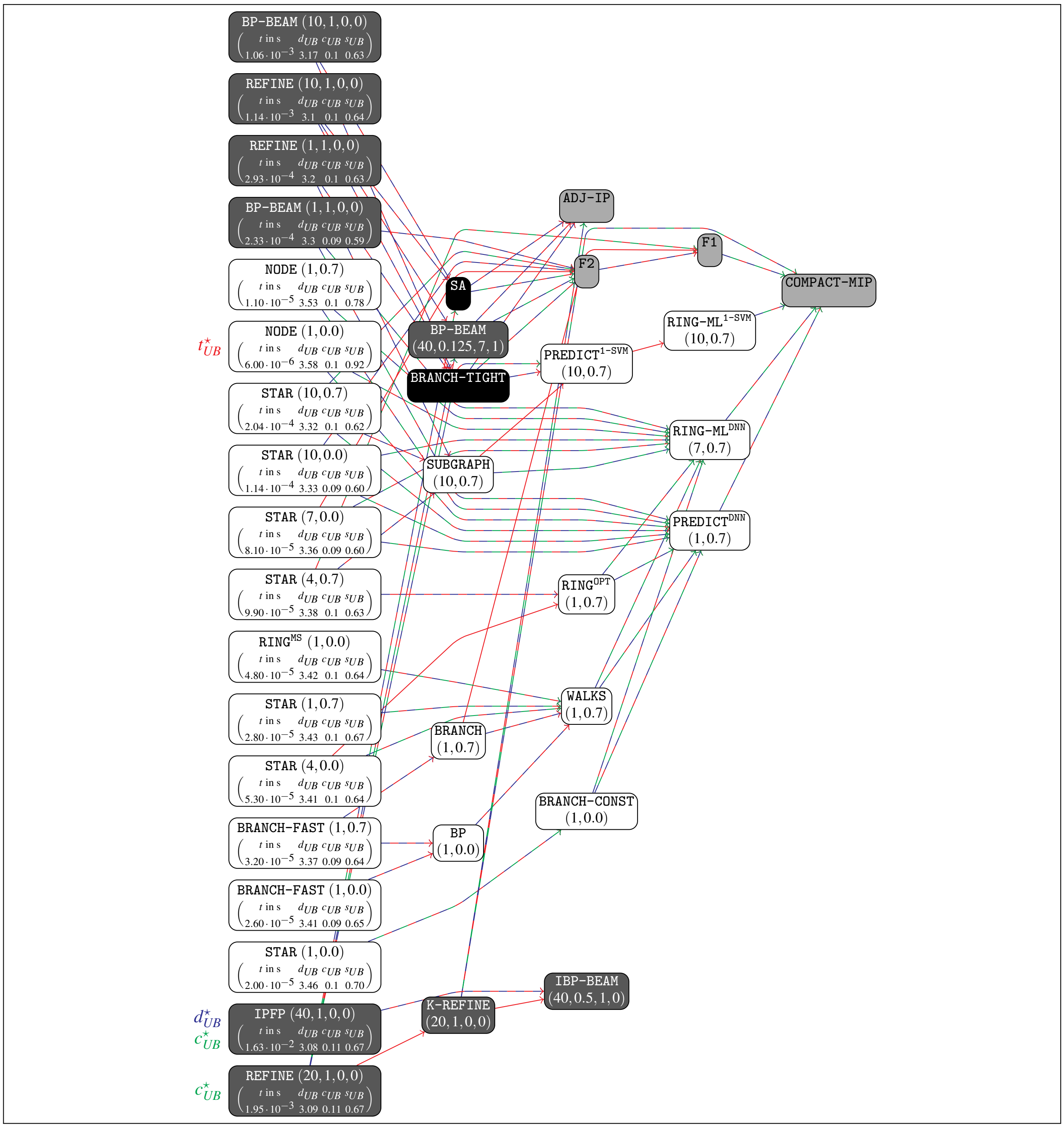

BP-BEAM (10,1,0,0) t in s d_{UB} c_{UB} s_{UB} $1.06\cdot10^{-3}$ 3.17 0.1 0.63
REFINE (10,1,0,0) t in s d_{UB} c_{UB} s_{UB} $1.14\cdot10^{-3}$ 3.1 0.1 0.64
REFINE (1,1,0,0) t in s d_{UB} c_{UB} s_{UB} $2.93\cdot10^{-4}$ 3.2 0.1 0.63
BP-BEAM (1,1,0,0) t in s d_{UB} c_{UB} s_{UB} $2.33\cdot10^{-4}$ 3.3 0.09 0.59
NODE (1,0.7) t in s d_{UB} c_{UB} s_{UB} $1.10\cdot10^{-5}$ 3.53 0.1 0.78
$t^\star_{UB}$
NODE (1,0.0) t in s d_{UB} c_{UB} s_{UB} $6.00\cdot10^{-6}$ 3.58 0.1 0.92
STAR (10,0.7) t in s d_{UB} c_{UB} s_{UB} $2.04\cdot10^{-4}$ 3.32 0.1 0.62
STAR (10,0.0) t in s d_{UB} c_{UB} s_{UB} $1.14\cdot10^{-4}$ 3.33 0.09 0.60
STAR (7,0.0) t in s d_{UB} c_{UB} s_{UB} $8.10\cdot10^{-5}$ 3.36 0.09 0.60
STAR (4,0.7) t in s d_{UB} c_{UB} s_{UB} $9.90\cdot10^{-5}$ 3.38 0.1 0.63
RING^{MS} (1,0.0) t in s d_{UB} c_{UB} s_{UB} $4.80\cdot10^{-5}$ 3.42 0.1 0.64
STAR (1,0.7) t in s d_{UB} c_{UB} s_{UB} $2.80\cdot10^{-5}$ 3.43 0.1 0.67
STAR (4,0.0) t in s d_{UB} c_{UB} s_{UB} $5.30\cdot10^{-5}$ 3.41 0.1 0.64
BRANCH-FAST (1,0.7) t in s d_{UB} c_{UB} s_{UB} $3.20\cdot10^{-5}$ 3.37 0.09 0.64
BRANCH-FAST (1,0.0) t in s d_{UB} c_{UB} s_{UB} $2.60\cdot10^{-5}$ 3.41 0.09 0.65
STAR (1,0.0) t in s d_{UB} c_{UB} s_{UB} $2.00\cdot10^{-5}$ 3.46 0.1 0.70
$d^\star_{UB}$
$c^\star_{UB}$
IPFP (40,1,0,0) t in s d_{UB} c_{UB} s_{UB} $1.63\cdot10^{-2}$ 3.08 0.11 0.67
$c^\star_{UB}$
REFINE (20,1,0,0) t in s d_{UB} c_{UB} s_{UB} $1.95\cdot10^{-3}$ 3.09 0.11 0.67
ADJ-IP
F1
F2
COMPACT-MIP
SA
BP-BEAM (40,0.125,7,1)
BRANCH-TIGHT
PREDICT$^{1\text{-SVM}}$ (10,0.7)
RING-ML$^{1\text{-SVM}}$ (10,0.7)
RING-ML^{DNN} (7,0.7)
SUBGRAPH (10,0.7)
PREDICT^{DNN} (1,0.7)
RING^{OPT} (1,0.7)
WALKS (1,0.7)
BRANCH (1,0.7)
BRANCH-CONST (1,0.0)
BP (1,0.0)
K-REFINE (20,1,0,0)
IBP-BEAM (40,0.5,1,0)

Fig. 24 Transitive reduction of dominance graph for upper bounds on the dataset FP.

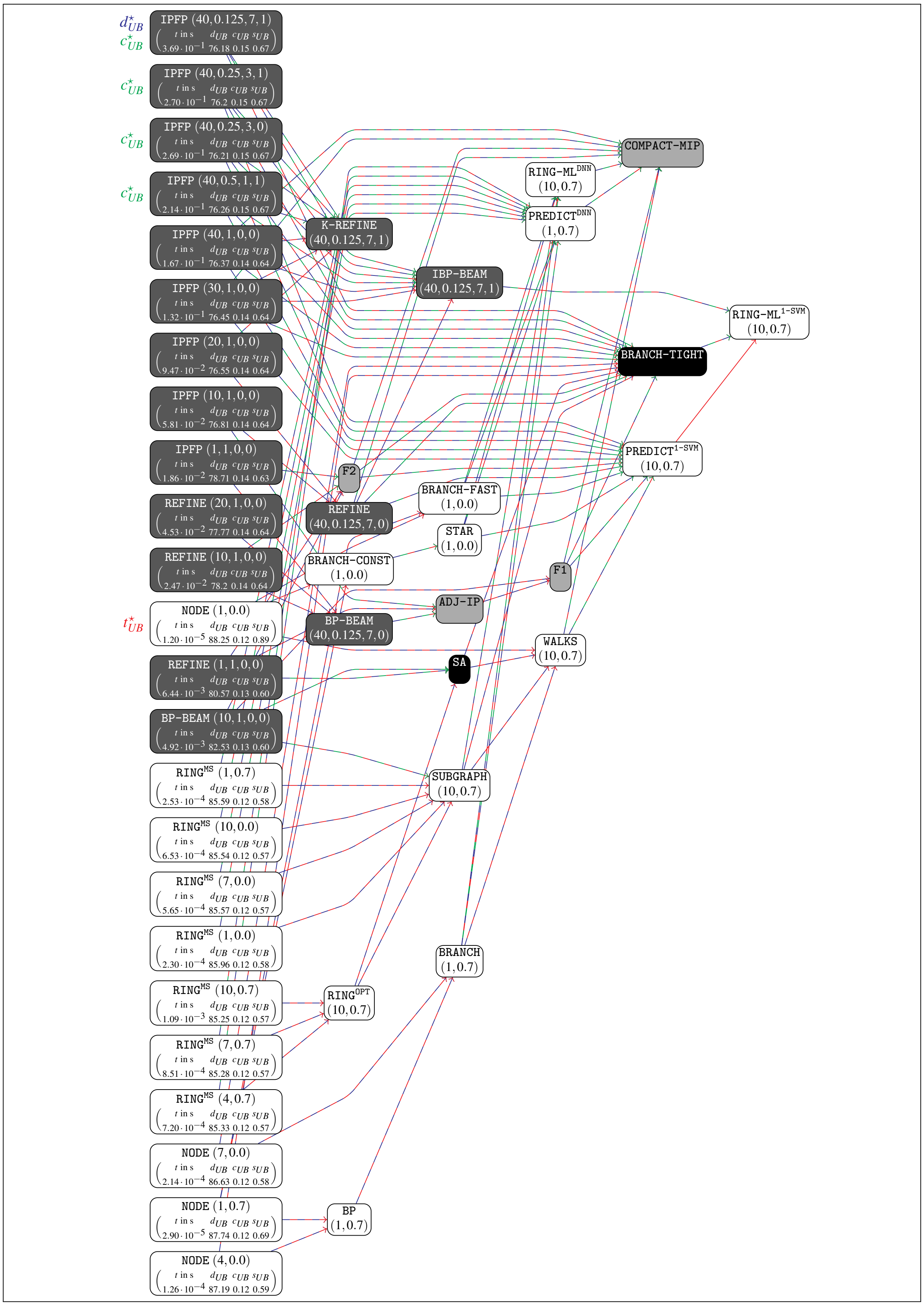

$d_{UB}^{\star}$
$c_{UB}^{\star}$
IPFP (40, 0.125, 7, 1) t in s d_{UB} c_{UB} s_{UB} $3.69 \cdot 10^{-1}$ 76.18 0.15 0.67
$c_{UB}^{\star}$
IPFP (40, 0.25, 3, 1) $2.70 \cdot 10^{-1}$ 76.2 0.15 0.67
$c_{UB}^{\star}$
IPFP (40, 0.25, 3, 0) $2.69 \cdot 10^{-1}$ 76.21 0.15 0.67
$c_{UB}^{\star}$
IPFP (40, 0.5, 1, 1) $2.14 \cdot 10^{-1}$ 76.26 0.15 0.67
IPFP (40, 1, 0, 0) $1.67 \cdot 10^{-1}$ 76.37 0.14 0.64
IPFP (30, 1, 0, 0) $1.32 \cdot 10^{-1}$ 76.45 0.14 0.64
IPFP (20, 1, 0, 0) $9.47 \cdot 10^{-2}$ 76.55 0.14 0.64
IPFP (10, 1, 0, 0) $5.81 \cdot 10^{-2}$ 76.81 0.14 0.64
IPFP (1, 1, 0, 0) $1.86 \cdot 10^{-2}$ 78.71 0.14 0.63
REFINE (20, 1, 0, 0) $4.53 \cdot 10^{-2}$ 77.77 0.14 0.64
REFINE (10, 1, 0, 0) $2.47 \cdot 10^{-2}$ 78.2 0.14 0.64
$t_{UB}^{\star}$
NODE (1, 0.0) $1.20 \cdot 10^{-5}$ 88.25 0.12 0.89
REFINE (1, 1, 0, 0) $6.44 \cdot 10^{-3}$ 80.57 0.13 0.60
BP-BEAM (10, 1, 0, 0) $4.92 \cdot 10^{-3}$ 82.53 0.13 0.60
RING$^{\text{MS}}$ (1, 0.7) $2.53 \cdot 10^{-4}$ 85.59 0.12 0.58
RING$^{\text{MS}}$ (10, 0.0) $6.53 \cdot 10^{-4}$ 85.54 0.12 0.57
RING$^{\text{MS}}$ (7, 0.0) $5.65 \cdot 10^{-4}$ 85.57 0.12 0.57
RING$^{\text{MS}}$ (1, 0.0) $2.30 \cdot 10^{-4}$ 85.96 0.12 0.58
RING$^{\text{MS}}$ (10, 0.7) $1.09 \cdot 10^{-3}$ 85.25 0.12 0.57
RING$^{\text{MS}}$ (7, 0.7) $8.51 \cdot 10^{-4}$ 85.28 0.12 0.57
RING$^{\text{MS}}$ (4, 0.7) $7.20 \cdot 10^{-4}$ 85.33 0.12 0.57
NODE (7, 0.0) $2.14 \cdot 10^{-4}$ 86.63 0.12 0.58
NODE (1, 0.7) $2.90 \cdot 10^{-5}$ 87.74 0.12 0.69
NODE (4, 0.0) $1.26 \cdot 10^{-4}$ 87.19 0.12 0.59
K-REFINE (40, 0.125, 7, 1)
IBP-BEAM (40, 0.125, 7, 1)
F2
REFINE (40, 0.125, 7, 0)
BRANCH-CONST (1, 0.0)
BP-BEAM (40, 0.125, 7, 0)
RING$^{\text{OPT}}$ (10, 0.7)
BP (1, 0.7)
COMPACT-MIP
RING-ML$^{\text{DNN}}$ (10, 0.7)
PREDICT$^{\text{DNN}}$ (1, 0.7)
RING-ML$^{\text{1-SVM}}$ (10, 0.7)
BRANCH-TIGHT
PREDICT$^{\text{1-SVM}}$ (10, 0.7)
BRANCH-FAST (1, 0.0)
STAR (1, 0.0)
F1
ADJ-IP
SA
WALKS (10, 0.7)
SUBGRAPH (10, 0.7)
BRANCH (1, 0.7)

Fig. 25 Transitive reduction of dominance graph for upper bounds on the dataset AIDS.

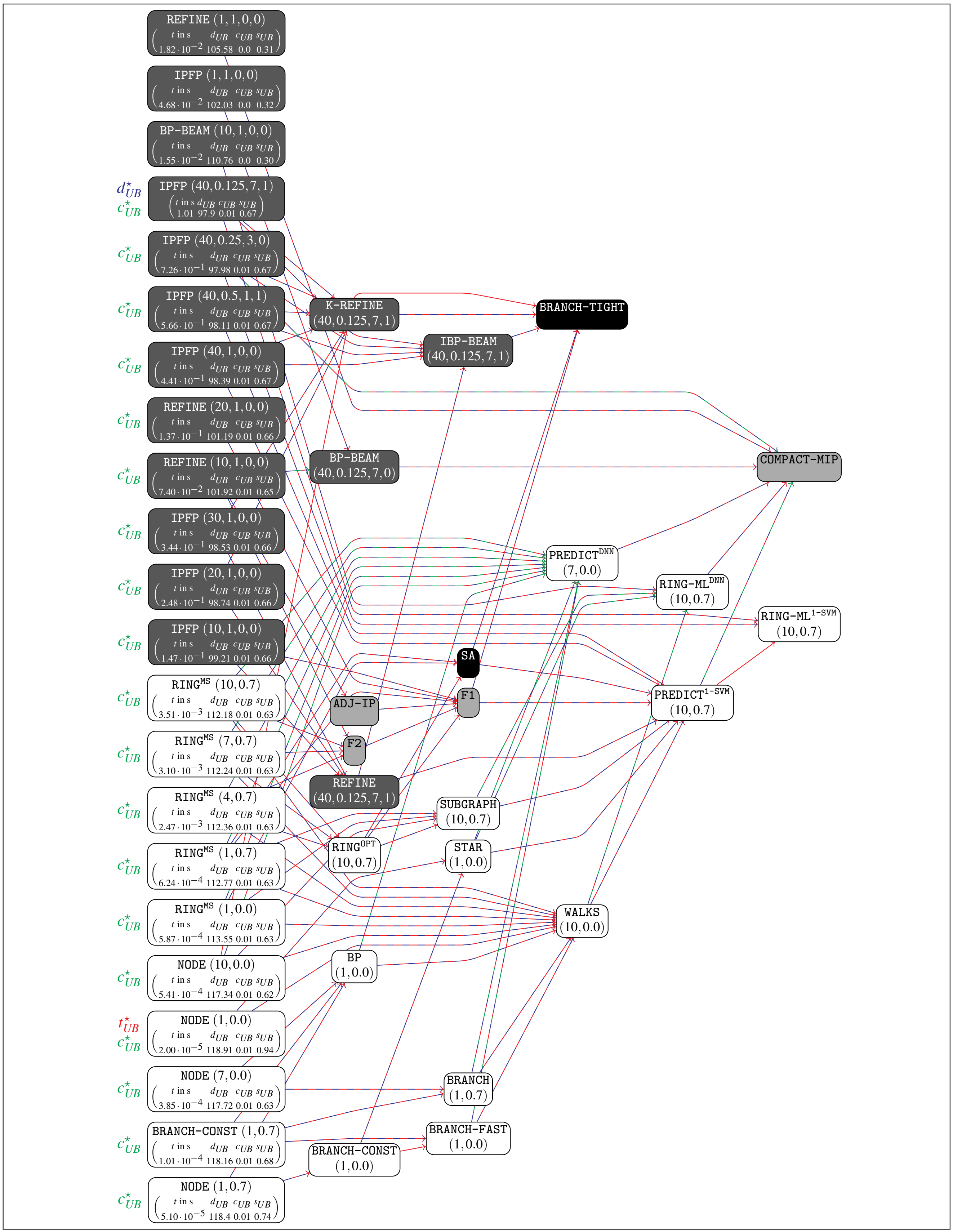

REFINE (1,1,0,0) t in s 1.82·10^−2 dUB 105.58 cUB 0.0 sUB 0.31
IPFP (1,1,0,0) t in s 4.68·10^−2 dUB 102.03 cUB 0.0 sUB 0.32
BP-BEAM (10,1,0,0) t in s 1.55·10^−2 dUB 110.76 cUB 0.0 sUB 0.30
d*UB c*UB IPFP (40,0.125,7,1) t in s 1.01 dUB 97.9 cUB 0.01 sUB 0.67
c*UB IPFP (40,0.25,3,0) t in s 7.26·10^−1 dUB 97.98 cUB 0.01 sUB 0.67
c*UB IPFP (40,0.5,1,1) t in s 5.66·10^−1 dUB 98.11 cUB 0.01 sUB 0.67
c*UB IPFP (40,1,0,0) t in s 4.41·10^−1 dUB 98.39 cUB 0.01 sUB 0.67
c*UB REFINE (20,1,0,0) t in s 1.37·10^−1 dUB 101.19 cUB 0.01 sUB 0.66
c*UB REFINE (10,1,0,0) t in s 7.40·10^−2 dUB 101.92 cUB 0.01 sUB 0.65
c*UB IPFP (30,1,0,0) t in s 3.44·10^−1 dUB 98.53 cUB 0.01 sUB 0.66
c*UB IPFP (20,1,0,0) t in s 2.48·10^−1 dUB 98.74 cUB 0.01 sUB 0.66
c*UB IPFP (10,1,0,0) t in s 1.47·10^−1 dUB 99.21 cUB 0.01 sUB 0.66
c*UB RING^MS (10,0.7) t in s 3.51·10^−3 dUB 112.18 cUB 0.01 sUB 0.63
c*UB RING^MS (7,0.7) t in s 3.10·10^−3 dUB 112.24 cUB 0.01 sUB 0.63
c*UB RING^MS (4,0.7) t in s 2.47·10^−3 dUB 112.36 cUB 0.01 sUB 0.63
c*UB RING^MS (1,0.7) t in s 6.24·10^−4 dUB 112.77 cUB 0.01 sUB 0.63
c*UB RING^MS (1,0.0) t in s 5.87·10^−4 dUB 113.55 cUB 0.01 sUB 0.63
c*UB NODE (10,0.0) t in s 5.41·10^−4 dUB 117.34 cUB 0.01 sUB 0.62
t*UB c*UB NODE (1,0.0) t in s 2.00·10^−5 dUB 118.91 cUB 0.01 sUB 0.94
c*UB NODE (7,0.0) t in s 3.85·10^−4 dUB 117.72 cUB 0.01 sUB 0.63
c*UB BRANCH-CONST (1,0.7) t in s 1.01·10^−4 dUB 118.16 cUB 0.01 sUB 0.68
c*UB NODE (1,0.7) t in s 5.10·10^−5 dUB 118.4 cUB 0.01 sUB 0.74
K-REFINE (40,0.125,7,1)
BRANCH-TIGHT
IBP-BEAM (40,0.125,7,1)
BP-BEAM (40,0.125,7,0)
COMPACT-MIP
PREDICT^DNN (7,0.0)
RING-ML^DNN (10,0.7)
RING-ML^1-SVM (10,0.7)
SA
F1
ADJ-IP
F2
PREDICT^1-SVM (10,0.7)
REFINE (40,0.125,7,1)
SUBGRAPH (10,0.7)
RING^OPT (10,0.7)
STAR (1,0.0)
WALKS (10,0.0)
BP (1,0.0)
BRANCH (1,0.7)
BRANCH-FAST (1,0.0)
BRANCH-CONST (1,0.0)

Fig. 26 Transitive reduction of dominance graph for upper bounds on the dataset MUTA.

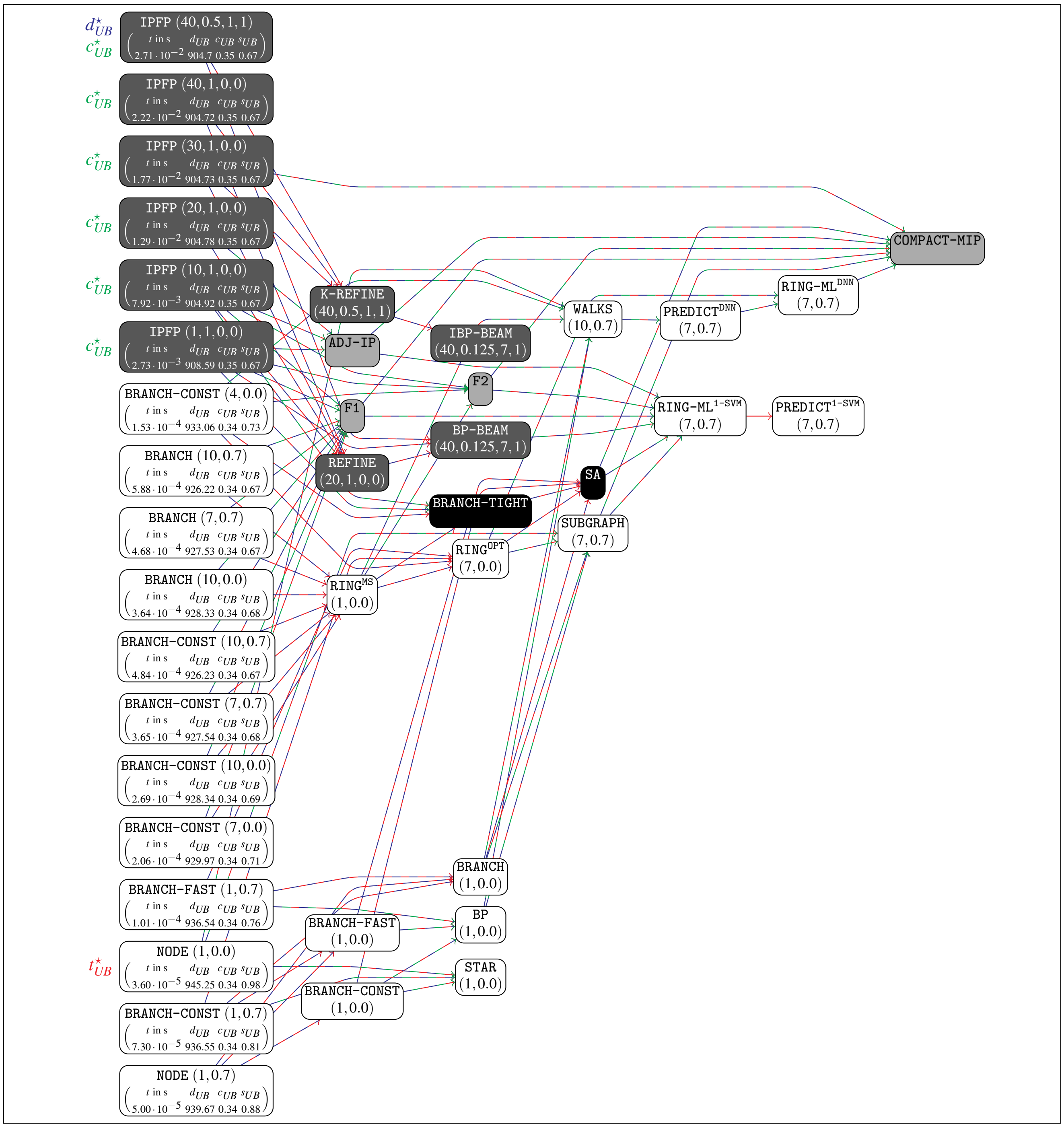

$d^\star_{UB}$
$c^\star_{UB}$
IPFP (40, 0.5, 1, 1)
t in s d_{UB} c_{UB} s_{UB}
$2.71 \cdot 10^{-2}$ 904.7 0.35 0.67
$c^\star_{UB}$
IPFP (40, 1, 0, 0)
$2.22 \cdot 10^{-2}$ 904.72 0.35 0.67
$c^\star_{UB}$
IPFP (30, 1, 0, 0)
$1.77 \cdot 10^{-2}$ 904.73 0.35 0.67
$c^\star_{UB}$
IPFP (20, 1, 0, 0)
$1.29 \cdot 10^{-2}$ 904.78 0.35 0.67
$c^\star_{UB}$
IPFP (10, 1, 0, 0)
$7.92 \cdot 10^{-3}$ 904.92 0.35 0.67
$c^\star_{UB}$
IPFP (1, 1, 0, 0)
$2.73 \cdot 10^{-3}$ 908.59 0.35 0.67
BRANCH-CONST (4, 0.0)
$1.53 \cdot 10^{-4}$ 933.06 0.34 0.73
BRANCH (10, 0.7)
$5.88 \cdot 10^{-4}$ 926.22 0.34 0.67
BRANCH (7, 0.7)
$4.68 \cdot 10^{-4}$ 927.53 0.34 0.67
BRANCH (10, 0.0)
$3.64 \cdot 10^{-4}$ 928.33 0.34 0.68
BRANCH-CONST (10, 0.7)
$4.84 \cdot 10^{-4}$ 926.23 0.34 0.67
BRANCH-CONST (7, 0.7)
$3.65 \cdot 10^{-4}$ 927.54 0.34 0.68
BRANCH-CONST (10, 0.0)
$2.69 \cdot 10^{-4}$ 928.34 0.34 0.69
BRANCH-CONST (7, 0.0)
$2.06 \cdot 10^{-4}$ 929.97 0.34 0.71
BRANCH-FAST (1, 0.7)
$1.01 \cdot 10^{-4}$ 936.54 0.34 0.76
$t^\star_{UB}$
NODE (1, 0.0)
$3.60 \cdot 10^{-5}$ 945.25 0.34 0.98
BRANCH-CONST (1, 0.7)
$7.30 \cdot 10^{-5}$ 936.55 0.34 0.81
NODE (1, 0.7)
$5.00 \cdot 10^{-5}$ 939.67 0.34 0.88
K-REFINE (40, 0.5, 1, 1)
ADJ-IP
F1
REFINE (20, 1, 0, 0)
RING^{MS} (1, 0.0)
BRANCH-FAST (1, 0.0)
BRANCH-CONST (1, 0.0)
IBP-BEAM (40, 0.125, 7, 1)
F2
BP-BEAM (40, 0.125, 7, 1)
BRANCH-TIGHT
RING^{OPT} (7, 0.0)
BRANCH (1, 0.0)
BP (1, 0.0)
STAR (1, 0.0)
WALKS (10, 0.7)
SA
SUBGRAPH (7, 0.7)
PREDICT^{DNN} (7, 0.7)
RING-ML^{DNN} (7, 0.7)
RING-ML^{1-SVM} (7, 0.7)
PREDICT^{1-SVM} (7, 0.7)
COMPACT-MIP

Fig. 27 Transitive reduction of dominance graph for upper bounds on the dataset GREC.

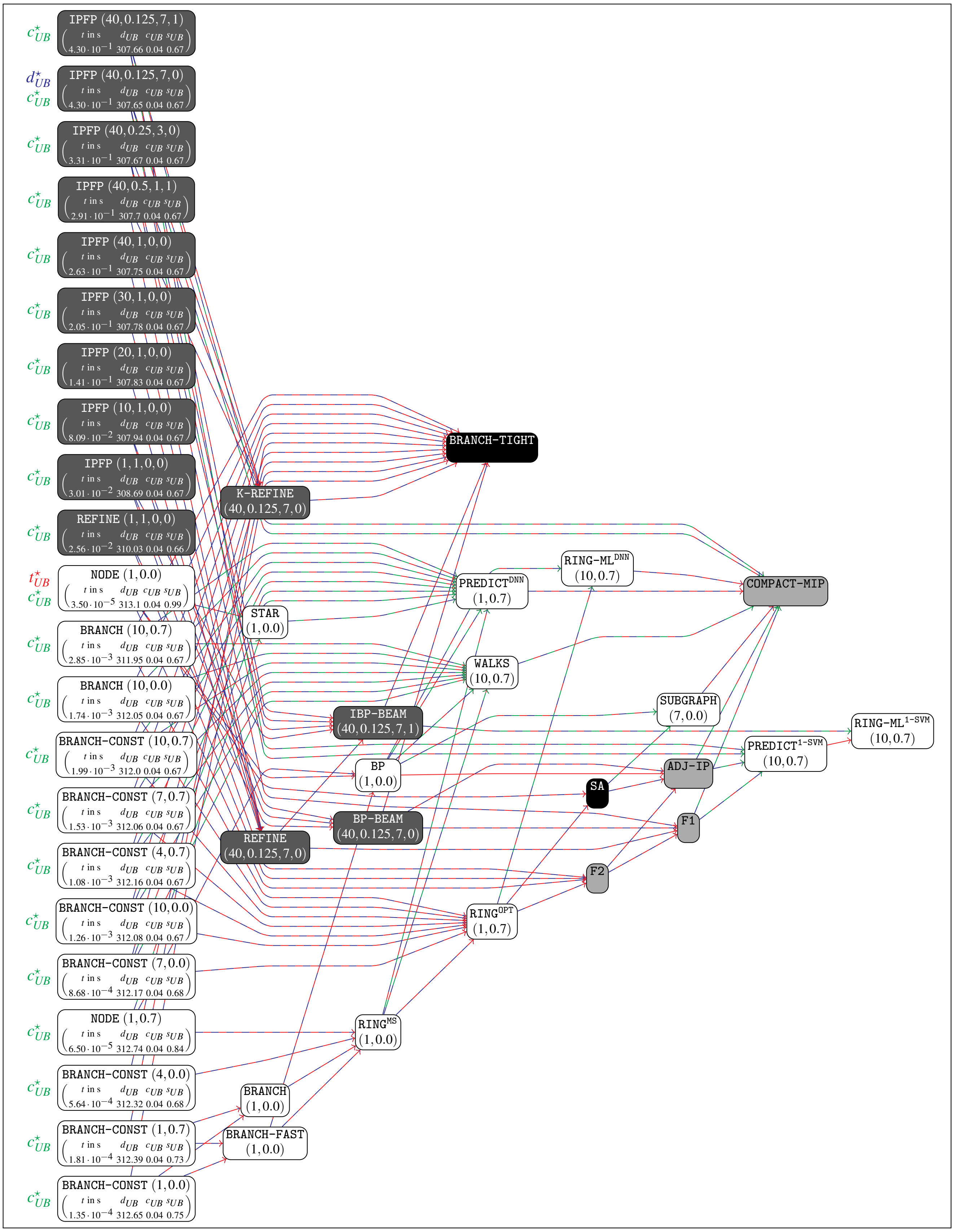

$c^\star_{UB}$ IPFP (40, 0.125, 7, 1) t in s d_{UB} c_{UB} s_{UB} 4.30 · 10^−1 307.66 0.04 0.67
$d^\star_{UB}$ $c^\star_{UB}$ IPFP (40, 0.125, 7, 0) 4.30 · 10^−1 307.65 0.04 0.67
$c^\star_{UB}$ IPFP (40, 0.25, 3, 0) 3.31 · 10^−1 307.67 0.04 0.67
$c^\star_{UB}$ IPFP (40, 0.5, 1, 1) 2.91 · 10^−1 307.7 0.04 0.67
$c^\star_{UB}$ IPFP (40, 1, 0, 0) 2.63 · 10^−1 307.75 0.04 0.67
$c^\star_{UB}$ IPFP (30, 1, 0, 0) 2.05 · 10^−1 307.78 0.04 0.67
$c^\star_{UB}$ IPFP (20, 1, 0, 0) 1.41 · 10^−1 307.83 0.04 0.67
$c^\star_{UB}$ IPFP (10, 1, 0, 0) 8.09 · 10^−2 307.94 0.04 0.67
$c^\star_{UB}$ IPFP (1, 1, 0, 0) 3.01 · 10^−2 308.69 0.04 0.67
$c^\star_{UB}$ REFINE (1, 1, 0, 0) 2.56 · 10^−2 310.03 0.04 0.66
$t^\star_{UB}$ $c^\star_{UB}$ NODE (1, 0.0) 3.50 · 10^−5 313.1 0.04 0.99
$c^\star_{UB}$ BRANCH (10, 0.7) 2.85 · 10^−3 311.95 0.04 0.67
$c^\star_{UB}$ BRANCH (10, 0.0) 1.74 · 10^−3 312.05 0.04 0.67
$c^\star_{UB}$ BRANCH-CONST (10, 0.7) 1.99 · 10^−3 312.0 0.04 0.67
$c^\star_{UB}$ BRANCH-CONST (7, 0.7) 1.53 · 10^−3 312.06 0.04 0.67
$c^\star_{UB}$ BRANCH-CONST (4, 0.7) 1.08 · 10^−3 312.16 0.04 0.67
$c^\star_{UB}$ BRANCH-CONST (10, 0.0) 1.26 · 10^−3 312.08 0.04 0.67
$c^\star_{UB}$ BRANCH-CONST (7, 0.0) 8.68 · 10^−4 312.17 0.04 0.68
$c^\star_{UB}$ NODE (1, 0.7) 6.50 · 10^−5 312.74 0.04 0.84
$c^\star_{UB}$ BRANCH-CONST (4, 0.0) 5.64 · 10^−4 312.32 0.04 0.68
$c^\star_{UB}$ BRANCH-CONST (1, 0.7) 1.81 · 10^−4 312.39 0.04 0.73
$c^\star_{UB}$ BRANCH-CONST (1, 0.0) 1.35 · 10^−4 312.65 0.04 0.75
BRANCH-TIGHT
K-REFINE (40, 0.125, 7, 0)
RING-ML$^{\text{DNN}}$ (10, 0.7)
PREDICT$^{\text{DNN}}$ (1, 0.7)
COMPACT-MIP
STAR (1, 0.0)
WALKS (10, 0.7)
SUBGRAPH (7, 0.0)
IBP-BEAM (40, 0.125, 7, 1)
RING-ML$^{\text{1-SVM}}$ (10, 0.7)
PREDICT$^{\text{1-SVM}}$ (10, 0.7)
BP (1, 0.0)
ADJ-IP
SA
BP-BEAM (40, 0.125, 7, 0)
F1
REFINE (40, 0.125, 7, 0)
F2
RING$^{\text{OPT}}$ (1, 0.7)
RING$^{\text{MS}}$ (1, 0.0)
BRANCH (1, 0.0)
BRANCH-FAST (1, 0.0)

Fig. 28 Transitive reduction of dominance graph for upper bounds on the dataset PROTEIN.