# The Minimum Edit Arborescence Problem and Its Use in Compressing Graph Collections*

Lucas Gnecco[1] [0000−0002−1561−2080], Nicolas Boria[1] [0000−0002−0548−4257], Sébastien Bougleux[2] [0000−0002−4581−7570], Florian Yger[1] [0000−0002−7182−8062], and David B. Blumenthal[3] [0000−0001−8651−750X]

[1] PSL Université Paris-Dauphine, LAMSADE, Paris, France
{lucas.gnecco,nicolas.boria,florian.yger}@dauphine.fr
[2] UNICAEN, ENSICAEN, CNRS, GREYC, Caen, France
sebastien.bougleux@unicaen.fr
[3] Department Artificial Intelligence in Biomedical Engineering (AIBE), Friedrich-Alexander University Erlangen-Nürnberg (FAU), Erlangen, Germany
david.b.blumenthal@fau.de

**Abstract.** The inference of minimum spanning arborescences within a set of objects is a general problem which translates into numerous application-specific unsupervised learning tasks. We introduce a unified and generic structure called *edit arborescence* that relies on edit paths between data in a collection, as well as the Minimum Edit Arborescence Problem, which asks for an edit arborescence that minimizes the sum of costs of its inner edit paths. Through the use of suitable cost functions, this generic framework allows to model a variety of problems. In particular, we show that by introducing *encoding size preserving edit costs*, it can be used as an efficient method for compressing collections of labeled graphs. Experiments on various graph datasets, with comparisons to standard compression tools, show the potential of our method.

**Keywords:** Edit arborescence · Edit distance · Lossless compression.

## 1 Introduction

The discovery of some underlying structure within a collection of data is the main goal of unsupervised learning. Among the different kinds of graph structures available for structure inference, arborescences play an essential role, because they contain the minimal number of edges required to connect all the entries of the collection and induce a meaningful hierarchy within the data. For these reasons, arborescences are widely used in structure inference, in numerous fields ranging from bioinformatics [12] to computational linguistics [19]. For constructing arborescences, distances have to be computed for data objects within the collection.

* Supported by Agence Nationale de la Recherche (ANR), projects STAP ANR-17-CE23-0021 and DELCO ANR-19-CE23-0016. FY acknowledges the support of the ANR as part of the "Investissements d'avenir" program, reference ANR-19-P3IA-0001 (PRAIRIE 3IA Institute).

While the computation of distances is trivial for many kinds of simple data (e. g., vectors in Euclidean space), it is often challenging for more complex kinds of data such as strings, trees, or graphs. For such data, *edit distances* — measuring the distance between two objects $o_1$ and $o_2$ as the cost of modifications needed to transform $o_1$ into $o_2$ — provide meaningful measures.

In this work, we propose a unified and generic framework for minimum arborescence computation on collections of structured data for which an edit distance is available. We introduce the concept of *edit arborescence* which generalizes the concept of edit path (a sequence of edit operations, or modifications), and we formalize the Minimum Edit Arborescence Problem (*MEA*). By using appropriate edit cost functions over the edit operations, as well as different sets of allowed edit operations, this generic framework allows to tackle a variety of specific problems, such as event detection in time series [16], morphological forests inference over a language vocabulary [18], or structured data compression [10].

As a proof of concept, we focus on the latter application, and address the problem of compressing a collection of labeled graphs. To the best of our knowledge, this problem has not been addressed in the literature. In graph stores, each graph is encoded individually using space-efficient representations based on different, mainly lossless compression schemes [3,4,8], but without taking into account the other graphs in the store. This is also the case for lossy graph compression schemes [22]. All of these compression schemes are beyond the main focus of this paper, and we refer the reader to the above references.

Contrary to these schemes, our compression method relies heavily on reference-based compression underpinned by an arborescence connecting the graphs of the collection. Intuitively, each graph is represented by an edit path between its parent graph and itself. Each graph can thus be reconstructed recursively up to the root element of the arborescence, which we define as the empty graph. Similar ideas have been proposed for compressing web graphs seen as a temporal graphs with edge insertions and deletions [1], or collections of bitvectors using the Hamming distance [9], recently applied to graph annotations (colors) [2]. While these approaches can be considered as early examples of using *MEA* in compression, our formulation is more general.

We first formalize the concepts of edit distances and arborescences in Section 2. In Section 3, we introduce edit arborescences and define *MEA*. Section 4 deals specifically with graph data and the graph edit distance, and formalizes the Minimum Graph Edit Arborescence Problem (*MGEA*). Section 5 provides detailed explanations on how to make use of the *MGEA* to address the compression of a set of labeled graphs. In Section 6, we report the results of the experimental evaluation. Finally, Section 7 concludes the paper and points out to future work.

## 2 Preliminaries

We consider data (sequence, tree, graph) defined by a combinatorial structure and labels attached to the elements of this structure. Labels may be of any type. Unlabeled and unstructured data are special cases.

***Edit Distance.*** Given a space $\Omega$ of all data of a fixed type, an *edit path* is a sequence of elementary modifications (or *edit operations*) transforming an object of $\Omega$ into another one. Typical edit operations are the deletion and the insertion of an element of the structure, and the substitution of an attribute attached to an element. Given a cost function $c \geq 0$ defined on edit operations, the *edit distance* $d_c : \Omega \times \Omega \to \mathbb{R}_{\geq 0}$ measures the minimal total cost required to transform $x \in \Omega$ into $y \in \Omega$, up to an equivalence relation: $d_c(x, y) \coloneqq \min_{P \in \mathcal{P}(x,y)} c(P)$, with $c(P) \coloneqq \sum_{o \in P} c(o)$ the cost of an edit path $P$, and $\mathcal{P}(x, y)$ the set of all edit paths transforming $x$ into an element of $[y] \coloneqq \{z \in \Omega \mid y \sim z\}$, the equivalence class of $y$ for an equivalence relation $\sim$ on $\Omega$. Equality is the equivalence relation usually considered for strings or sequences (Hamming, Levenshtein or discrete time warping distances). Isomorphism is used for trees and graphs.

The set $\Omega$ equipped with an edit distance $d_c$ defines an *edit space* $(\Omega, d_c)$. We assume that $d_c$ is metric or pseudometric (if $\sim$ is not equality). The set $\Omega$ contains a null (or empty) element denoted by $0_\Omega \in \Omega$. Any other element of $(\Omega, d_c)$ can be constructed by insertion operations only from $0_\Omega$.

***Arborescences.*** A directed graph (digraph) is a pair $G \coloneqq (V, E)$, where $V \coloneqq \{v_0, .., v_n\}$ is a set of nodes and $E \subseteq V \times V$ is a set of directed edges. Within such a graph, a *spanning arborescence* is a rooted, oriented, spanning tree, i. e., a set of edges that induces exactly one directed path from a root node $r \in V$ to each other node in $V \setminus \{r\}$. By assuming w. l. o. g. that the root element is $v_0$ and reminding that all other nodes in an arborescence have a unique parent node, an arborescence can be represented by a sequence of node indices $\mathcal{A}$ such that, for all $i \in [1, n]$, $\mathcal{A}[i]$ denotes the index of the unique parent node of node $v_i$. The set of edges of $\mathcal{A}$ is denoted by $E^{\mathcal{A}}$.

## 3 The Minimum Edit Arborescence Problem

In this section, we introduce and describe the generic Minimum Edit Arborescence Problem (*MEA*), a versatile problem. An instance of *MEA* is a finite dataset $X$ living in an edit space $(\Omega, d_c)$. *MEA* asks for a minimum-cost edit arborescence rooted at the null element.

Given a set $X \coloneqq \{x_0, x_1, ..., x_n\} \subset \Omega$ such that $x_0 \coloneqq 0_\Omega$, we define an *edit arborescence* as a pair $(\mathcal{A}, \Psi)$, where $\mathcal{A}$ is a sequence of $n$ indices that defines an arborescence rooted at the index 0, such that for all $i \in [1, n]$, $\mathcal{A}[i]$ is the parent-index of $i$. $\Psi \coloneqq (P_1, ..., P_n)$ is a sequence of edit paths, such that $P_i \in \mathcal{P}(x_{\mathcal{A}[i]}, x_i)$ holds for all $i \in [1, n]$, i. e., $P_i$ is an edit path between $x_i$ and its parent in $\mathcal{A}$. $\mathbb{A}(X)$ is the set of all edit arborescences on $X$.

**Definition 1** **(***MEA***).** *Given a finite set $X \subset \Omega$ and an edit cost function $c$, the* Minimum Edit Arborescence Problem *(MEA) asks for an edit arborescence $(A^\star, \Psi^*)$ on $X \cup \{0_\Omega\}$, which is rooted at the null element $0_\Omega \in \Omega$ and has a minimum cost $c(\Psi^\star)$ among all $(\mathcal{A}, \Psi) \in \mathbb{A}(X)$, with $c(\Psi) \coloneqq \sum_{P \in \Psi} c(P)$.*

By definition, it holds that $c(\Psi^\star) = \min_{(\mathcal{A},\Psi)\in\mathbb{A}_c(X)} \sum_{P_i\in\Psi} d_c(x_{\mathcal{A}[i]}, x_i)$, where $\mathbb{A}_c(X)$ is the set of edit arborescences in $(\Omega, d_c)$, i. e., edit arborescences with edit paths restricted to minimal-cost edit paths w. r. t. $c$. This generic definition can translate into various optimization problems, with different characteristics in terms of complexity and/or approximability, depending on the edit space.

***Exact Solver.*** Whenever exact edit distances and corresponding edit paths can be computed, the following procedure produces an optimal solution for *MEA*:

1. Construct the complete directed weighted graph on the set $X \cup \{0_\Omega\}$, denoted by $\mathcal{K}(X, d_c) \coloneqq (V^{\mathcal{K}}, E^{\mathcal{K}}, w)$, with node set $V^{\mathcal{K}} \coloneqq X \cup \{0_\Omega\}$ and edge weights $w(u, v) \coloneqq d_c(u, v)$ for all $(u, v) \in E^{\mathcal{K}}$. Note that any edge entering the root can be removed.
2. Solve the Minimum Spanning Arborescence Problem (*MSA*) on $\mathcal{K}(X, d_c)$ with $0_\Omega$ as root node.

For a connected weighted directed graph $G$ and a root node $r$ in $G$, the Minimum Spanning Arborescence Problem (*MSA*) asks for a spanning arborescence $A^\star$ on $G$, which is rooted in $r$ and has minimum weight $w(A^\star)$, where $w(A) \coloneqq \sum_{(u,v)\in A} w(u, v)$ [13]. *MSA* can be solved in polynomial time, e. g., in $O(|V^G|^2)$ time with Tarjan's implementation [23] of Edmonds' algorithm [13]. Hence, the main difficulty of the problem consists in computing the edge weights in $\mathcal{K}$, i. e., the edit distances between elements of $X$.

**Lemma 1.** *As long as the edit space $(\Omega, d_c)$ allows for a polynomial time computation of minimum-cost edit paths, the corresponding version of MEA belongs to the complexity class $\mathcal{P}$.*

*Proof.* By assumption, $d_c$ is computed in polynomial time by some algorithm `ALG-DIST` that is called $O(n^2)$ times with complexity $O_{\texttt{ALG-DIST}}$ in order to generate the complete graph $\mathcal{K}(X, d_c)$. So, *MEA* can be solved in $O(n^2 O_{\texttt{ALG-DIST}} + (n+1)^2)$ time complexity by using Tarjan's implementation of Edmond's algorithm. □

***A Heuristic for Non-Polynomial Cases.*** We adapt the algorithm described above to cases where the edit distance is not solvable in polynomial time. The method is based on approximations or heuristics to estimate the edit distance, and allows the user to choose the desired balance between computation time and accuracy. Also, the algorithm takes advantage of prior knowledge over the data (such as relevant candidate couples of elements) which is often available in practical cases. Given a set $X \subset \Omega$, Algorithm 1 computes a *low-cost* edit arborescence $(\mathcal{A}, \Psi) \in \mathbb{A}(X)$ based on approximations or heuristics to estimate the edit distance. It starts by constructing a size-reduced auxiliary digraph $\mathcal{K}$ (lines 1 to 3) that connects $0_\Omega$ to each element $x_i \in X$, and each $x_i$ to $k \leq |X| - 1$ randomly selected elements of $X \setminus \{x_i\}$. If some promising edges are known *a priori* (e. g., if $X$ has an implicit internal structure), they are added to $\mathcal{K}$. Then, Algorithm 1 computes optimal or low-cost edit paths whose costs provide weights for the edges of $\mathcal{K}$ (lines 4 to 8). For $0_\Omega$' out-edges, optimal edit paths can be

**Algorithm 1** A generic heuristic for *MEA*.

**Require:** A finite set $X$ of elements from an edit space $(\Omega, d)$ with origin $0_\Omega$, a parameter $k \in [0, |X-1|]$, two edit distance heuristics `ALG-1` and `ALG-2`.
**Ensure:** A low-cost edit arborescence $(\mathcal{A}, \Psi)$ for the *MEA* problem.
1: Set $x_0 := 0_\Omega$ and initialize auxiliary graph $\mathcal{K}(X \cup x_0, E^{\mathcal{K}}, w)$ with $E^{\mathcal{K}} := \{x_0\} \times X$.
2: **for** $x \in X$ **do** Sample $k$ children $\tilde{X} \in \binom{X \setminus \{x\}}{k}$ and set $E^{\mathcal{K}} := E^{\mathcal{K}} \cup (\{x\} \times \tilde{X})$.
3: **if** prior information available **then** Add promising edges to $E^{\mathcal{K}}$.
4: **for** $(x_i, x_j) \in E^{\mathcal{K}}$ **do**
5: **if** $i = 0$ **then** Analytically compute the edit path $P_{ij}$.
6: **else if** identifiers available **then** Compute edit path $P_{ij}$ induced by identifiers.
7: **else** Call `ALG-1` to compute low-cost edit path $P_{ij}$.
8: Set $w(x_i, x_j) := c(P_{ij})$.
9: Run Edmonds' algorithm on $\mathcal{K}$ to obtain $\mathcal{A}$.
10: **if** tightening **then for** $i \in [1, n]$ **do** Call `ALG-2` to compute tighter edit path $P_{\mathcal{A}[i]i}$.
11: **for** $i \in [1, n]$ **do** Set $\Psi[i] := P_{\mathcal{A}[i]i}$.
12: **return** $(\mathcal{A}, \Psi)$

computed analytically (insertions only). If identifying attributes are available for all elements of $X$ (e. g., unique node labels if $X$ is a set of graphs), it is sometimes possible to compute optimal edit paths from these identifiers. Otherwise, low-cost edit paths are computed by calling a polynomial edit distance heuristic `ALG-1`. Once all edge weights for $\mathcal{K}$ have been computed, an optimal arborescence $\mathcal{A}$ on $\mathcal{K}$ is constructed by Edmonds' algorithm (line 9). Optionally, a tighter edit distance heuristic `ALG-2` can be called to shorten the paths in $\mathcal{A}$ before returning the edit arborescence (line 10). The more precise, and thus potentially more costly heuristic `ALG-2` is called only $n$ times.

## 4 Minimum Graph Edit Arborescence Problem

In the remainder of the paper, we will focus on the specific case of *MEA* where the space $\Omega$ is a space of labeled graphs.

***Graphs.*** We assume that graphs are finite, simple, undirected, and labeled. However, all presented techniques can be straightforwardly adapted to directed or unlabeled graphs. A labeled graph $G$ is a four-tuple $G := (V^G, E^G, \ell_V^G, \ell_E^G)$, where $V^G$ and $E^G$ are sets of nodes and edges, while $\ell_V^G : V^G \to \Sigma_V$ and $\ell_E^G : E^G \to \Sigma_E$ are labeling functions that annotate nodes and edges with labels from alphabets $\Sigma_V$ and $\Sigma_E$, respectively. $\mathbb{G}(\Sigma_V, \Sigma_E)$, or $\mathbb{G}$ for short, denotes the set of all graphs for fixed alphabets $\Sigma_V$ and $\Sigma_E$. $0_\mathbb{G}$ denotes the empty graph (the null element of $\mathbb{G}$). Two graphs $G, H \in \mathbb{G}$ are *isomorphic*, denoted by $G \simeq H$, if and only if there is a bijection between $V^G$ and $V^H$ that preserves both edges and labels.

***Edit Operations and Edit Paths.*** We consider the following elementary edit operations, where $\epsilon$ is a dummy node and $\epsilon_\ell$ is a dummy label:

- Node deletion (nd): $(v, \epsilon_\ell)$, with $v \in V^G$ isolated.
- Edge deletion (ed): $(e, \epsilon_\ell)$, with $e \in E^G$.
- Node relabeling (nr): $(v, \ell) \in V^G \times (\Sigma_V \setminus \{\ell_V^G(v)\})$.
- Edge relabeling (er): $(e, \ell) \in E^G \times (\Sigma_E \setminus \{\ell_E^G(e)\})$.
- Node insertion (ni): $(\epsilon, \ell)$, with $\ell \in \Sigma_V$.
- Edge insertion (ei): $(e, \ell) \in (\binom{V^G}{2} \setminus E^G) \times \Sigma_E$.

For each edit path $P$ composed of such operations, there are many equivalent edit paths with a same edit cost, obtained just by reordering the operations in $P$. In particular, as the deletion of a node assumes that its incident edges have been previously deleted, these operations can be replaced by node-edge deletions (ned): delete all the edges incident to a node and then delete this node. So we can distinguish two different types of edge deletions: implied edge deletion (i-ed), i. e., an edge deletion in a node-edge deletion, and non-implied edge deletion (ni-ed), i. e., an edge deletion between two nodes that are not deleted by $P$. The cost of an edit path $P$ can thus be rewritten as $c(P) = \sum_{t \in T} \sum_{o \in P^t} c^t(o)$, where $P^t$ is the (possibly empty) set of all edit operations of type $t \in T$, with $T \coloneqq \{\text{ni-ed}, \text{i-ed}, \text{nd}, \text{nr}, \text{er}, \text{ni}, \text{ei}\}$, and $c^t$ is an edit cost function for type $t$.

*Remark 1.* Any concatenation $\sigma_{\text{ni-ed}}(P^{\text{ni-ed}}) \sqcup \sigma_{\text{i-ed}}(P^{\text{i-ed}}) \sqcup \ldots \sqcup \sigma_{\text{ei}}(P^{\text{ei}})$ of edit operations, with $\sigma_t$ a permutation on $P^t$, defines an edit path equivalent to $P$.

*Remark 2.* $c(P) = \sum_{t \in T} c^t |P^t|$ if $c_t$ is a constant for each type of operation $t$.

***Node Maps and Induced Edit Paths.*** A *node map* (or *error-correcting bipartite matching*) between a graph $G$ and a graph $H$ is a relation $\pi \in (V^G \cup \{\epsilon\}) \times (V^H \cup \{\epsilon\})$ such that the following two conditions hold:

- For each node $u \in V^G$, there is exactly one node $v \in V^H \cup \{\epsilon\}$ such that $(u, v) \in \pi$. We denote this node $v$ by $\pi(u)$.
- For each node $v \in V^H$, there is exactly one node $u \in V^G \cup \{\epsilon\}$ such that $(u, v) \in \pi$. We denote this node $u$ by $\pi^{-1}(v)$.

Let $\Pi(V^G, V^H)$ be the set of all node maps. Each node map $\pi \in \Pi(G, H)$ can be transformed into an edit path, denoted by $P[\pi]$ (*induced edit path*), such that, for each $(u, v) \in \pi$, there is a corresponding edit operation $(u, \ell)$: $u$ is deleted if $v = \epsilon$, it is relabeled if $(u, v) \in V^G \times V^H$ and $\ell_V^G(u) \neq \ell_V^H(v)$, or a new node is inserted if $u = \epsilon$. Operations on edges are induced by the operations on nodes, i. e., from the pairs $((u, \pi(u)), (v, \pi(v))$ with $u, v \in V^G \cup \{\epsilon\}$. For details, we refer to [6]. What is important here is that any type of edit operation is taken into account by a node map. In particular, implied and non-implied edge deletions can be distinguished with a specific cost for each type.

***Graph Edit Distance.*** The cost of an optimal edit path from a graph $G$ to a graph $H' \simeq H$ defines the graph edit distance (GED) from $G$ to $H$ ($d_c$ with graph isomorphism as equivalence relation): $\text{GED}(G, H) \coloneqq \min_{P \in \mathcal{P}(G,H)} c(P)$. GED is hard to compute and approximate, even when restricting to simple special

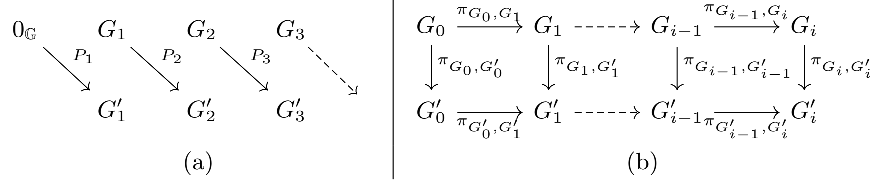


Fig. 1: (a) Paths in a non-reconstructible edit arborescence. (b) Composition of node maps used to construct reconstructible edit arborescences.

cases [24,5]. However, many heuristics are able to reach tight upper and/or lower bounds. They are based on a reformulation of GED as an Error-Correcting Graph Matching Problem: $\mathrm{GED}(G, H) = \min_{\pi \in \Pi(V^G, V^H)} c(P[\pi])$, which is equivalent to the above definition under mild assumptions on the edit cost function $c$. We refer to [20,6] for an overview.

***Problem Formulation and Hardness.*** We can now define *MGEA*:

**Definition 2 (*MGEA*).** *The* Minimum Graph Edit Arborescence Problem *(MGEA) is a MEA problem with* $\Omega \coloneqq \mathbb{G}$*,* $X \coloneqq \{G_1, ..., G_n\}$ *and* $d_c \coloneqq \mathrm{GED}$*.*

As the problem of computing GED is $\mathcal{NP}$-hard, Lemma 1 does not apply here.

**Theorem 1.** *MGEA is* $\mathcal{NP}$*-hard.*

The proof is omitted here due to space constraints, we refer the reader to [15] for a detailed proof, based on a reduction from the Hamiltonian cycle problem.

## 5 Arborescence-Based Compression

In this section, we show how to leverage *MGEA* for compressing a set of labeled graphs. For this, we introduce reconstructible and non-reconstructible edit arborescences, formulate the Arborescence-Based Compression Problem (*ABC*), and present an encoding for induced edit paths.

***Reconstructible Edit Arborescence.*** When $d_c \coloneqq \mathrm{GED}$, since the definition of GED is based on graph isomorphism, applying an induced edit path $P[\pi_{G,H}]$ to a graph $G$ yields a graph $H' \simeq H$. When using such edit paths within an edit arborescence $(\mathcal{A}, \Psi) \in \mathbb{A}(X)$, with $X \subset \mathbb{G}$ a set of graphs, the configuration described in Figure 1(a) occurs. Namely, the edit paths in $\Psi$ may be disjoint due to the isomorphism relation between the source graphs $G_i$ and target graphs $G'_i$. Thus, the arborescence does not allow the reconstruction of any graph that is not directly connected to the root element. In this sense, only specific edit arborescences allow to reconstruct all graphs in $X$ up to isomorphism.

**Definition 3 (Reconstructability).** *An edit arborescence* $(\mathcal{A}, \Psi) \in \mathbb{A}(X)$ *is* reconstructible *if and only if each graph* $G_i \in X$ *can be constructed up to isomorphism by applying the sequence of edit paths* $P \coloneqq (P_{s^1}, P_{s^2}, \ldots, P_i)$ *to the empty graph* $0_{\mathbb{G}}$*, where* $(0_{\mathbb{G}}, G_{s^1}, G_{s^2}, \ldots, G_i)$ *is the path from* $0_{\mathbb{G}}$ *to* $G_i$ *in* $\mathcal{A}$*.*

By composition of node maps (Figure 1(b)), it is easy to show the following property (proof omitted due to space constraints).

**Lemma 2.** *For any edit arborescence $(\mathcal{A}, \Psi)$ on a set of graphs $X$, there is a reconstructible edit arborescence $(\mathcal{A}, \Psi')$ on a set $X'$, such that $X'$ is isomorphic to $X$ and it holds that $c(\Psi') = c(\Psi)$.*

By Definition 3, the set of graphs $X$ can be reconstructed up to graph isomorphism based on the encoding of a reconstructible edit arborescence.

***Problem Formulation.*** For compressing a finite set of graphs $X \subset \mathbb{G}$, we are interested in finding a reconstructible edit arborescence $(\mathcal{A}, \Psi) \in \mathbb{A}(X)$ with a small encoding size $|\mathrm{C}(\mathcal{A}, \Psi)|$, where $\mathrm{C}(\cdot)$ denotes the encoding (a binary string) for the code C. Ideally we would like to minimize this size over $\mathbb{A}(X)$ and all possible codes. To encode an edit arborescence, we encode both of its elements, i. e., the arborescence $\mathcal{A}$ defined as a sequence of indices, and the sequence $\Psi$ of edit paths induced by the node maps. In order to derive a useful expression for the code length function $|\mathrm{C}(\cdot)|$, the edit path encodings are concatenated. Thus, the encoding size to optimize is given by $|\mathrm{C}(\mathcal{A}, \Psi)| = |\mathrm{C}(M_{\mathcal{A}})| + |\mathrm{C}(\mathcal{A})| + \sum_{P \in \Psi} |\mathrm{C}(P)|$, where $M_{\mathcal{A}}$ is the overhead for decoding the different parts of $\mathrm{C}(\mathcal{A}, \Psi)$. In order to optimize this size, we must define an *encoding size preserving* cost function which forces the encoding sizes of edit paths to coincide with their edit cost.

**Definition 4 (Encoding Size Preservation).** *Let* C *be a code for edit paths. An edit cost function $c$ is* encoding size preserving *w. r. t. code* C *if and only if there is a constant $\gamma$ such that $|\mathrm{C}(P)| = c(P) + \gamma$ holds for any edit path $P$. Put differently, an encoding size preserving cost function assigns to each edit operation the space required in memory to encode the operations with code* C.

Assuming that a code C and an encoding size preserving cost function $c$ w. r. t. C exist, the encoding size for any edit arborescence $(\mathcal{A}, \Psi) \in \mathbb{A}(X)$ can be rewritten as $|\mathrm{C}(\mathcal{A}, \Psi)| = |\mathrm{C}(M_{\mathcal{A}})| + |\mathrm{C}(\mathcal{A})| + c(\Psi) + \gamma|X|$. Since the encoding size for $\mathcal{A}$ depends only on the number of nodes, the problem of minimizing $|\mathrm{C}(\mathcal{A}, \Psi)|$ amounts to minimizing $c(\Psi)$. Consequently, finding a compact encoding of a set of graphs $X$ reduces to a *MGEA* problem as introduced in Section 4.

**Definition 5 (*ABC*).** *Let $X \coloneqq \{G_1, \ldots, G_n\} \subset \mathbb{G}$ be a finite set of graphs,* C *be a code for edit paths, and $c$ be an encoding size preserving edit cost function for* C*. Then, the* ARBORESCENCE-BASED COMPRESSION PROBLEM *(ABC) asks for a minimum weight reconstructible edit arborescence $(\mathcal{A}, \Psi')$ on some set of graphs $X' \coloneqq \{G'_1, \ldots, G'_n\}$ such that, for all $i \in [1, n]$, $G'_i \simeq G_i$.*

We stress that, thanks to the use of encoding size preserving edit costs, the value that is optimized by *ABC* corresponds to the length of the code $\mathrm{C}(\mathcal{A}, \Psi')$ up to a constant. In other words, solving *ABC* produces the most compact arborescence-based representation of $X$. Given the simple correspondence between reconstructible edit arborescences and their non-reconstructible counterparts, *ABC* reduces to *MGEA* by restricting to encoding size preserving edit costs. Since *MGEA* is $\mathcal{NP}$-hard (Theorem 1), we propose to heuristically compute a low-cost edit arborescence as detailed below.

**Algorithm 2** *ABC* encoding of graph collections.

**Require:** A set of graphs $X$, a code C, and an edit cost function $c$.
**Ensure:** Encoding $\mathrm{C}(\mathcal{A}, \Psi')$ of a reconstructible edit arborescence $(\mathcal{A}, \Psi')$ on $X$.
1: Compute $(\mathcal{A}, \Psi)$ with Algorithm 1.
2: Initialize list $L \coloneqq [(0_{\mathbb{G}}, \pi_{\mathrm{id}})]$, where $\pi_{\mathrm{id}}$ is the identity.
3: Initialize encoding $\mathrm{C}(\mathcal{A}, \Psi') \coloneqq \mathrm{C}(M_{\mathcal{A}})\mathrm{C}(E^{\mathcal{A}})$.
4: **while** $L \neq \emptyset$ **do**
5: Pop an element $(G_i, \pi_{G_i,G'_i})$ from $L$.
6: **for all** children $G_j$ of $G_i$ in $\mathcal{A}$ **do**
7: Get $\pi \in \Pi(G_i, G_j)$ with $P[\pi] = \Psi[j]$ and initialize node ID $v' \coloneqq 1$.
8: Initialize $\pi' \in \Pi(G'_i, G'_j)$ as node map of insertions and deletions only.
9: **for all** $v \in V^{G_i}$ **if** $\pi(v) \neq \epsilon$ **then** Set $\pi'(\pi_{G_i,G'_i}(v)) \coloneqq v'$ and increment $v'$.
10: Concatenate $\mathrm{C}(P[\pi'])$ to $\mathrm{C}(\mathcal{A}, \Psi')$.
11: **if** $G_j$ is no leaf in $\mathcal{A}$ **then** Append $(G_j, \pi' \circ \pi_{G_i,G'_i} \circ \pi^{-1})$ to $L$.
12: **return** $\mathrm{C}(\mathcal{A}', \Psi')$

***Heuristic Solver for ABC.*** Algorithm 2 sketches our strategy to tackle the *ABC* problem. Given a set $X$ of graphs, it first uses Algorithm 1, which outputs a non-reconstructible edit arborescence $(\mathcal{A}, \Psi)$ on $X$. After initializing the code, it starts encoding a reconstructible edit arborescence by going through the arborescence in BFS order (line 4). For each new node $G_j$ with parent node $G_i$, a node map $\pi'$ from $G'_i \simeq G_i$ to $G'_j \simeq G_j$ is reconstructed (lines 7 to 9), and the code of its induced edit path is added to the code of the arborescence (line 10). If $G_j$ is not a leaf, the node map representing the isomorphism between $G_j$ and $G'_j$ is computed for later use (line 11).

*Remark 3 (Star Ratio).* In the worst case, we obtain a star $\mathcal{S} \in \mathbb{A}(X)$, which connects the empty graph $0_{\mathbb{G}}$ to all the graphs in $X$. This yields the upper bound $|\mathrm{C}(\mathcal{A}, \Psi)| \leq |\mathrm{C}(\mathcal{S})|$ on the encoding size of the obtained arborescence. Since encoding a graph from the empty graph by insertion operations only is similar to encoding the graph itself, the encoding size for the star is close to the encoding size for $X$, for a similar encoding strategy. Consequently, the *star ratio* $|\mathrm{C}(\mathcal{A})|/|\mathrm{C}(\mathcal{S})|$ provides a good indicator for the compression quality.

*Remark 4.* The encoded structure is designed to allow a straightforward decompression of any graph $G_i$, which, starting from the empty graph, simply consists in consecutively applying the edit paths along the path from the root to $G_i$ in $\mathcal{A}$.

***A Code for Induced Edit Paths.*** We show that there is a code C for edit paths and an edit cost function $c$ such that $c$ is encoding size preserving w. r. t. C. Using the notations introduced in Section 4, we encode an edit path as the concatenated string $\mathrm{C}(P[\pi]) \coloneqq \mathrm{C}(M_P)\mathrm{C}(P^{\text{ni-ed}})\mathrm{C}(P^{\text{nd}})\mathrm{C}(P^{\text{nr}})\mathrm{C}(P^{\text{er}})\mathrm{C}(P^{\text{ni}})\mathrm{C}(P^{\text{ei}})$, where $M_P$ denotes the overhead for decoding each string $\mathrm{C}(P^t)$. Note that the set $P^{\text{i-ed}}$ is not encoded, since implied edge deletions can be implicitly represented by node deletions. Similarly, we encode a set of edit operations $P^t$ as $\mathrm{C}(P^t) \coloneqq \mathrm{C}(o^t_1)\mathrm{C}(o^t_2)\cdots\mathrm{C}(o^t_{|P^t|})$, with $o^t_i \in P^t$. Any edit operation $o \coloneqq (a, \ell)$ is

encoded as $\mathrm{C}(o) := \mathrm{C}(a)\mathrm{C}(\ell)$, with $\mathrm{C}(a) := \emptyset$ if $a = \epsilon$, and $\mathrm{C}(\ell) := \emptyset$ if $\ell = \epsilon_\ell$. That is, the dummy elements $\epsilon$ and $\epsilon_\ell$ in deletion operations and node insertions are not encoded. Ultimately, the encoding size for $P[\pi]$ hence depends on how the nodes, edges, and their labels are encoded in the codes of the edit operations.

We consider fixed-length codes for nodes, edges, and their labels (other codes will be studied in future works). For a set $X \in \mathbb{G}$, nodes are encoded as integers on $\beta_V$ bits, edges are encoded as a pairs of integers on $2\beta_V$ bits, and node or edge labels are encoded on, respectively, $\beta_{\Sigma_V}$ and $\beta_{\Sigma_E}$ bits. Dictionaries can be used for the labels and encoded in the overhead $M_{\mathcal{A}}$ or known *a priori*. In order to decode each set of edit operation $P^t$, $M_P$ must contain their sizes $|P^t|$. They are encoded on $\beta_P$ bits for each edit path. We obtain $|\mathrm{C}(P[\pi])| = \beta_P + \sum_{t \in T} c^t |P^t|$, where $c^{\mathrm{nr}} := \beta_V + \beta_{\ell_V}$, $c^{\mathrm{nd}} := \beta_V$, $c^{\mathrm{ni}} := \beta_{\Sigma_V}$, $c^{\mathrm{er}} := c^{\mathrm{ei}} := 2\beta_V + \beta_{\Sigma_E}$, $c^{\text{ed-ni}} := 2\beta_V$, and $c^{\text{ed-i}} := 0$. With these constant costs, the pair $(\mathrm{C}, c)$ defined above is encoding size preserving (Remark 2 and Definition 4) with constant $\beta_P$ for any node map $\pi$, i. e., $|\mathrm{C}(P[\pi])| = c(P[\pi]) + \beta_P$. Therefore, the encoding size for a spanning edit arborescence $(\mathcal{A}, \Psi) \in \mathbb{A}(X)$ reduces to $|\mathrm{C}(\mathcal{A}, \Psi)| = |\mathrm{C}(M_{\mathcal{A}})| + |\mathrm{C}(\mathcal{A})| + c(\Psi) + \beta_P|X|$, which implies that minimizing $|\mathrm{C}(\cdot)|$ is an *ABC* problem.

## 6 Experiments

We performed an empirical evaluation of our compression method in the context of data archiving. Since no dedicated algorithms for compressing graph collections exist in this context, we compared it to the generic tar.bz compression, sufficient to highlight the potential of our method. Other generic compression tools such as zip or tar.gz yielded worse compression ratios than tar.bz in initial tests.

***Datasets.*** We used eight different datasets (Table 1). The datasets AIDS and MUTA from the IAM Graph Database Repository [21], and ACYCL, PAH, and MAO from GREYC's Chemistry Dataset[4] contain graphs modeling chemical compounds. We also tested on time-evolving minimum spanning trees (MSTs) induced by the pairwise correlations of a large-scale U. S. stocks time series dataset.[5] Such MSTs are widely used for detecting critical market events such as financial crises [11,17]. We constructed three versions of the MSTs with the code in [17]: STOCKS-F (edge labels are floating-point stocks correlations), STOCKS-I (the correlations are rounded to integers), and STOCKS-N (no edge label). For all datasets, graphs were initially stored in GXL format.[6]

***Parameters and Implementation.*** We tested two versions of our *ABC* method (Algorithm 2) — with and without additional tar.bz compression of the obtained codes. For both versions, the out-degree $k$ of all nodes in $\mathcal{K}$ was varied across $\{0.1 \cdot |X|, 0.2 \cdot |X|, \ldots, 1.0 \cdot |X|\}$, and we did 5 repetitions for each value. For the experiments reported in Table 2, we performed 10 repetitions for

[4] `https://brunl01.users.greyc.fr/CHEMISTRY/index.html`
[5] `https://www.kaggle.com/borismarjanovic/price-volume-data-for-all-us-stocks-etfs`
[6] `https://userpages.uni-koblenz.de/~ist/GXL/index.php`

Table 1: Number of graphs $|X|$, maximum and average number of nodes $|V|$, as well as node and edge label alphabet sizes $|\Sigma_V|$ and $|\Sigma_E|$ for all datasets.

| dataset | $\|X\|$ | max $\|V\|$ | avg $\|V\|$ | $\|\Sigma_V\|$ | $\|\Sigma_E\|$ | dataset | $\|X\|$ | max $\|V\|$ | avg $\|V\|$ | $\|\Sigma_V\|$ | $\|\Sigma_E\|$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ACYCL | 183 | 11 | 8.15 | 3 | 1 | MAO | 68 | 27 | 18.38 | 3 | 4 |
| MUTA | 4337 | 417 | 30.32 | 14 | 3 | STOCKS-N | 1600 | 213 | 212.99 | 213 | 0 |
| AIDS | 1500 | 95 | 15.72 | $\infty$ | 3 | STOCKS-I | 1600 | 213 | 212.99 | 213 | 100 |
| PAH | 94 | 28 | 20.7 | 1 | 1 | STOCKS-F | 1600 | 213 | 212.99 | 213 | $\infty$ |

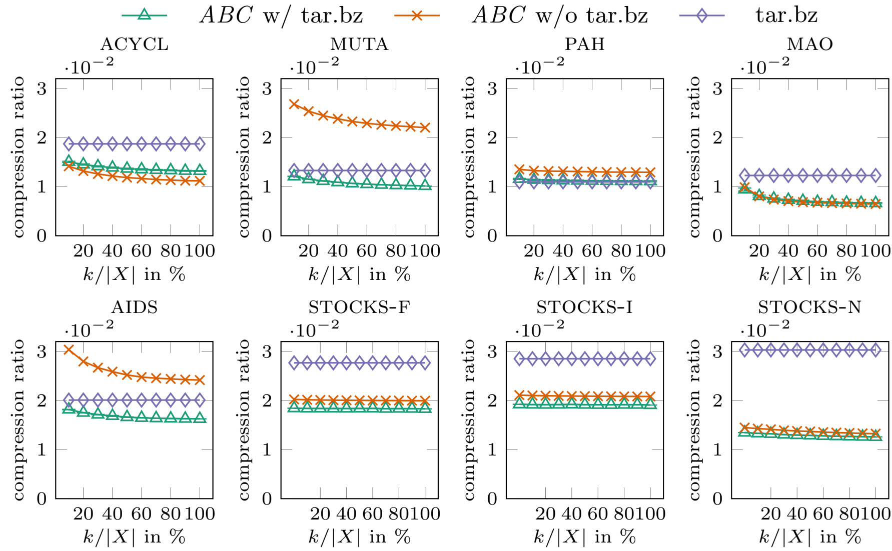


Fig. 2: Mean compression ratios w. r. t. out-degree $k$ for tar.bz and *ABC* w/ or w/o tar.bz. For STOCKS, the values for $k = 0$ in the plots correspond to the setting where the auxiliary graph $\mathcal{K}$ only contains temporal edges.

each dataset. For STOCKS, we also added all temporal edges to $\mathcal{K}$ and always used the node maps induced by the stock identities across time (cf. lines 3 and 6 in Algorithm 1). On the other datasets, the node maps were computed and refined using the GED heuristics `ALG-1` $:=$ `BRANCH-UNIFORM` [25] and `ALG-2` $:=$ `IPFP` [6]. All algorithms were implemented in C++ using the GED library GEDLIB [7] and the *MSA* library MSArbor [14].[7] Tar.bz compressions were performed at the default compression level (9, *i.e.* the highest compression). Tests were run on a Linux system with an Intel Haswell CPU (24 cores, 2.4 GHz each) and 19 GB of main memory.

***Compression Ratio.*** Figure 2 shows that, for all datasets except PAH, *ABC* with tar.bz significantly outperformed tar.bz compression alone and led to smaller

[7] `https://github.com/lucasgneccoh/gedlib`

Table 2: Mean compression and decompression times (in sec.), and standard deviations, of $ABC$ with tar.bz for $k = 0.4 \cdot |X|$, as well as mean depths, star ratios, numbers of leafs $|\mathcal{L}|$ and inner nodes $|\mathcal{I}|$ of the computed arborescences.

| dataset | $\|\mathcal{L}\|$ | $\|\mathcal{I}\|$ | avg depth | star ratio | compression | decompression |
|---|---|---|---|---|---|---|
| ACYCL | 65 | 118 | 19.5 | 0.37 | 8 ± 0.6 | 0.3 ± 0.02 |
| MUTA | 1751 | 2586 | 95.4 | 0.47 | 16052 ± 2385.0 | 14.6 ± 1.34 |
| AIDS | 641 | 859 | 46 | 0.63 | 673 ± 59.0 | 5.5 ± 0.31 |
| PAH | 35 | 59 | 13.8 | 0.38 | 16 ± 1.3 | 0.3 ± 0.04 |
| MAO | 23 | 45 | 13.3 | 0.17 | 13 ± 1.6 | 0.2 ± 0.02 |
| STOCKS-N | 133 | 1467 | 75.9 | 0.51 | 1662 ± 31.9 | 15.0 ± 0.61 |
| STOCKS-I | 153 | 1446 | 441.5 | 0.71 | 2095 ± 46.8 | 18.7 ± 0.37 |
| STOCKS-F | 148 | 1452 | 426.3 | 0.71 | 2166 ± 28.5 | 18.9 ± 0.43 |

compression ratios than $ABC$ w/o tar.bz for all datasets except ACYCL. Using out-degrees $k > 0.4 \cdot |X|$ only marginally improved compression. For STOCKS, using only temporal edges ($k = 0$) led to very good results. Moreover, STOCKS-N can be more compressed with $ABC$ than the other STOCKS datasets, as cheaper edit paths can be computed for graphs with unlabeled edges.

***Arborescence Structure, Star Ratio, and Runtime.*** Columns 2 to 4 of Table 2 provide statistics regarding the arborescences computed with $k = 0.4 \cdot |X|$. They seem to have a good balance between depth and width (number of leaves vs. number of internal nodes). The star ratios (column 5) indicate how much space is gained by using $ABC$ w. r. t. encoding each graph separately with the same underlying encoding scheme (a star ratio of 1 means no compression). Columns 6 to 9 summarize the $ABC$ compression and decompression times. The most important observation is that, although $ABC$ is much slower than tar.bz, the runtimes are still acceptable in application scenarios where a data holder wants to offer compressed graph datasets for download (compressing the largest dataset MUTA took about four to five hours). Indeed, unlike compression, decompression is fast even on the largest datasets (a couple of seconds). Runtime variations w. r. t. $k$ are detailed in Figure 3 for four datasets. As expected, the time required for computing the arborescences increases linearly with $k$, and the runtime of the refinement phase is independent of $k$. As the refinement algorithm `IPFP` is randomized, the runtimes of the refinement phase have a higher variability than the runtimes of the arborescence phase.

## 7 Conclusions

In this paper, we have proposed the concept of an *edit arborescence* and have introduced the MINIMUM EDIT ARBORESCENCE PROBLEM (*MEA*). *MEA* yields a generic framework for inferring hierarchies in finite sets of complex data objects

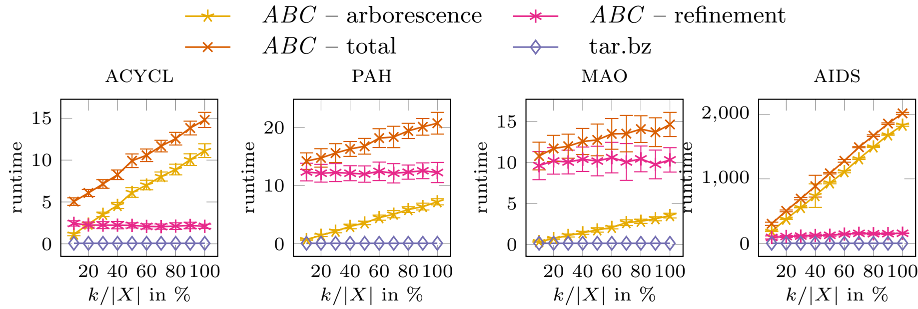


Fig. 3: Means and standard deviations of runtimes (in sec.) for $ABC$ with tar.bz and its subroutines, and tar.bz alone. $ABC$-total includes the final tar.bz step.

such as graphs or strings, which can be compared via *edit distances*. We have shown how to leverage $MEA$ for the lossless compression of collections of labeled graphs — a task, for which no dedicated algorithms are available to date. Experiments on eight datasets show that our approach $ABC$ clearly outperforms standard compression tools in terms of compression ratio and that it achieves reasonable compression and decompression times. More precisely, the experiments showed that (1) on seven out of eight datasets, our $ABC$ method clearly outperformed tar.bz compression in terms of compression ratio; (2) compressing with $ABC$ is computationally expensive but still reasonable in settings where the compression is carried out by an institutional data holder; (3) decompression is much faster and only takes a couple of seconds even for the largest test datasets.

## References


1. Adler, M., Mitzenmacher, M.: Towards compressing web graphs. In: Proceedings of the Data Compression Conference. p. 203. IEEE Computer Society (2001). https://doi.org/10.5555/882454.875027
2. Almodaresi, F., Pandey, P., Ferdman, M., Johnson, R., Patro, R.: An efficient, scalable, and exact representation of high-dimensional color information enabled using de bruijn graph search. Journal of Computational Biology **27**(4), 485–499 (2020). https://doi.org/10.1089/cmb.2019.0322
3. Besta, M., Hoefler, T.: Survey and taxonomy of lossless graph compression and space-efficient graph representations. CoRR **arXive:1806.01799 [cs.DS]** (2018)
4. Besta, M., Peter, E., Gerstenberger, R., Fischer, M., Podstawski, M., Barthels, C., Alonso, G., Hoefler, T.: Demystifying graph databases: Analysis and taxonomy of data organization, system designs, and graph queries. CoRR **arXive:1910.09017 [cs.DB]** (2019)
5. Blumenthal, D.B.: New Techniques for Graph Edit Distance Computation. Ph.D. thesis, Free University of Bozen-Bolzano (2019)
6. Blumenthal, D.B., Boria, N., Gamper, J., Bougleux, S., Brun, L.: Comparing heuristics for graph edit distance computation. VLDB J. **29**(1), 419–458 (2020). https://doi.org/10.1007/s00778-019-00544-1

7. Blumenthal, D.B., Bougleux, S., Gamper, J., Brun, L.: GEDLIB: A C++ library for graph edit distance computation. In: GbRPR 2019. LNCS, vol. 11510, pp. 14–24. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-20081-7_2
8. Boldi, P., Vigna, S.: The webgraph framework i: Compression techniques. p. 595–602. WWW '04 (2004). https://doi.org/10.1145/988672.988752
9. Bookstein, A., Klein, S.: Compression of correlated bit-vectors. Information Systems **16**(4), 387–400 (1991). https://doi.org/10.1016/0306-4379(91)90030-D
10. Chwatala, A.M., Raidl, G.R., Oberlechner, K.: Phylogenetic comparative methods. J. Math. Model. Algorithms **8**, 293–334 (2009). https://doi.org/10.1007/s10852-009-9109-1
11. Coelho, R., Gilmore, C.G., Lucey, B., Richmond, P., Hutzler, S.: The evolution of interdependence in world equity markets—evidence from minimum spanning trees. Physica A **376**, 455–466 (2007). https://doi.org/https://doi.org/10.1016/j.physa.2006.10.045
12. Cornwell, W., Nakagawa, S.: Phylogenetic comparative methods. Curr. Biol. **27**(9), R333–R336 (2017). https://doi.org/0.1016/j.cub.2017.03.049
13. Edmonds, J.: Optimum branchings. J. Res. Natl. Bur. Stand. B **71**(4), 233–240 (1967). https://doi.org/10.6028/jres.071b.032
14. Fischetti, M., Toth, P.: An efficient algorithm for the min-sum arborescence problem on complete digraphs. INFORMS J. Comput. **5**(4), 426–434 (1993). https://doi.org/10.1287/ijoc.5.4.426
15. Gnecco, L., Boria, N., Bougleux, S., Yger, F., Blumenthal, D.B.: The minimum edit arborescence problem and its use in compressing graph collections [extended version] (2021), `https://arxiv.org/abs/2107.14525`
16. Guralnik, V., Srivastava, J.: Event detection from time series data. In: Fayyad, U.M., Chaudhuri, S., Madigan, D. (eds.) SIGKDD 1999. pp. 33–42. ACM (1999). https://doi.org/10.1145/312129.312190
17. Liu, T., Coletti, P., Dignös, A., Gamper, J., Murgia, M.: Correlation graph analytics for stock time series data. In: EDBT 2021 (2021), `https://edbt2021proceedings.github.io/docs/p173.pdf`
18. Luo, J., Narasimhan, K., Barzilay, R.: Unsupervised learning of morphological forests. Trans. Assoc. Comput. Linguistics **5**, 353–364 (2017)
19. Moschitti, A., Pighin, D., Basili, R.: Semantic role labeling via tree kernel joint inference. In: CoNLL 2006. pp. 61–68. ACL (2006)
20. Riesen, K.: Structural Pattern Recognition with Graph Edit Distance: Approximation, Algorithms and Applications. Advances in Computer Vision and Pattern Recognition, Springer (2016). https://doi.org/10.1007/978-3-319-27252-8
21. Riesen, K., Bunke, H.: IAM graph database repository for graph based pattern recognition and machine learning. In: S+SSPR 2008. LNCS, vol. 5342, pp. 287–297. Springer, Berlin, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89689-0_33
22. Sourek, G., Zelezny, F., Kuzelka, O.: Lossless compression of structured convolutional models via lifting. CoRR **arXiv:2007.06567 [cs.LG]** (2021)
23. Tarjan, R.E.: Finding optimum branchings. Networks **7**(1), 25–35 (1977). https://doi.org/10.1002/net.3230070103
24. Zeng, Z., Tung, A.K.H., Wang, J., Feng, J., Zhou, L.: Comparing stars: On approximating graph edit distance. Proc. VLDB Endow. **2**(1), 25–36 (2009). https://doi.org/10.14778/1687627.1687631
25. Zheng, W., Zou, L., Lian, X., Wang, D., Zhao, D.: Efficient graph similarity search over large graph databases. IEEE Trans. Knowl. Data Eng. **27**(4), 964–978 (2015). https://doi.org/10.1109/TKDE.2014.2349924