Computing Optimal Assignments in Linear Time for Graph Matching

Nils M. Kriege 1  Pierre-Louis Giscard 2  Franka Bause 1  Richard C. Wilson 3

Abstract

Finding an optimal assignment between two sets of objects is a fundamental problem arising in many applications, including the matching of ‘bag-of-words’ representations in natural language processing and computer vision. Solving the assignment problem typically requires cubic time and its pairwise computation is expensive on large datasets. In this paper, we develop an algorithm which can find an optimal assignment in linear time when the cost function between objects is represented by a tree distance. We employ the method to approximate the edit distance between two graphs by matching their vertices in linear time. To this end, we propose two tree distances, the first of which reflects discrete and structural differences between vertices, and the second of which can be used to compare continuous labels. We verify the effectiveness and efficiency of our methods using synthetic and real-world datasets.

1. Introduction

Vast amounts of data are now available for machine learning, including text documents, images, graphs and many more. Learning from such data typically involves computing a similarity or distance function between the data objects. Since many of these datasets are large, the efficiency of the comparison methods is critical. This is particularly challenging since taking the structure adequately into account often is a hard problem. For example, no polynomial-time algorithms are known even for the basic task to decide whether two graphs have the same structure (Johnson, 2005). Therefore, the pragmatic approach of describing such data with a ‘bag-of-words’ or ‘bag-of-features’ is commonly used. In this representation, a series of objects are identified in the data and each object is described by a label or feature. The labels are placed in a bag where the order in which they appear does not matter.

In the most basic form, such bags can be represented by histograms or feature vectors, two of which are compared by counting the number of co-occurrences of a label in the bags. This is a common approach not only for images and text, but also for graph comparison, where a number of graph kernels have been proposed which use different substructures as elements of the bag (Vishwanathan et al., 2010). In the more general case, two bags of features are compared by summing over all pairs of features weighted by a similarity function between the features. However, in both cases, the method is not ideal, since each feature corresponds to a specific element in the data object, and so can correspond to no more than one element in a second data object. The co-occurrence counting method allows each feature to match to multiple features in the other dataset.

Another approach is to explicitly compute the best assignment between the features of two bags. The pyramid match kernel was proposed to approximate correspondences between bags of features in $\mathbb{R}^d$ by employing a space-partitioning tree structure and counting how often points fall into the same bin (Grauman & Darrell, 2007a; b). For graphs, the optimal assignment kernel was proposed, which establishes a correspondence between the vertices of two graphs (Fröhlich et al., 2005). However, solving the assignment problem in general has $O(n^3)$ running time where $n$ is the size of the bags, and can be slow on large datasets. Moreover, it was soon realised that this approach does not lead to valid kernels in the general case (Vert, 2008). Therefore, Johansson & Dubhashi (2015) derived kernels from optimal assignments by first sampling a fixed set of so-called landmarks and representing graphs by their optimal assignment similarities to landmarks. Kriege et al. (2016) demonstrated that a specific choice of a weight function (derived from a hierarchy) does in fact generate a valid kernel for the optimal assignment method and allows computation in linear time. The approach is not designed to actually construct the assignment.

In this paper, we show how an assignment of minimum cost can be computed in linear time when the costs of assigning individual features is determined by a tree distance.
Our method has applications beyond computing the optimal assignment between fixed sets and can be used for graph matching. The graph edit distance is one of the most widely accepted approaches to graph matching with applications ranging from cheminformatics to computer vision (Stauffer et al., 2017). It is defined as the minimum cost of edit operations required to transform one graph into another graph. The concept has been proposed for pattern recognition tasks more than 30 years ago (Sanfeliu & Fu, 1983). However, its computation is NP-hard, since it generalises the classical NP-complete maximum common subgraph problem (Bunke, 1997; Garey & Johnson, 1979). Recent binary linear programming formulations in combination with highly-optimised general purpose solvers are the most efficient exact approaches, but are still limited to small graphs (Lerouge et al., 2017). Even a restricted special case of the graph edit distance is APX-hard, i.e., there is a constant $c > 1$, such that no polynomial-time algorithm can approximate it within the factor $c$, unless $P=NP$ (Lin, 1994). However, heuristics based on bipartite matching or the assignment problem turned out to be effective tools (Riesen & Bunke, 2009) and are widely used in practice (Stauffer et al., 2017). The approach requires cubic running time, which is still not feasible for large graphs. Therefore, it has been proposed to use non-exact algorithms for solving the assignment problem. Simple greedy algorithms reduce the running time to $O(n^2)$ (Riesen et al., 2015a;b). For large graphs the quadratic running time is still problematic in practice. Moreover, many applications require a large number of distance computations, e.g., for solving classification tasks or performing similarity search in graph databases (Zeng et al., 2009).

**Our contribution.** We consider the optimal assignment problem under a cost function that is a tree distance. Using general costs, the input to the assignment problem is encoded by a matrix of quadratic size. The tree distance, however, can be represented compactly by a weighted tree of linear size. For this case, we propose an exact linear time algorithm for constructing an optimal assignment from the tree. We show how to embed the optimal assignment costs in an $\ell_1$ space without distortion. On this basis, we develop an algorithm for approximating the graph edit distance in linear time. To this end, we propose techniques for generating trees representing cost metrics for approximating the graph edit distance: (i) based on Weisfeiler-Lehman refinement to quantify discrete and structural differences, (ii) based on hierarchical clustering for continuous vertex attributes. We show experimentally that our linear time algorithm scales to large graphs and datasets. Our approach outperforms both exact and approximate methods for computing the graph edit distance in terms of running time and provides state-of-the-art classification accuracy. For some datasets with discrete labels our method even beats these approaches in terms of accuracy.

### 2. Fundamentals

We summarise basic concepts and results on tree distances, the assignment problem and the graph edit distance.

#### 2.1. Tree metrics and ultrametrics

A dissimilarity function $d : \mathcal{X} \times \mathcal{X}$ is a metric on $\mathcal{X}$, if it is (i) non-negative, (ii) symmetric, (iii) zero iff two objects are equal and (iv) satisfies the triangle inequality. A metric $d$ on $\mathcal{X}$ is an ultrametric if $d(x,y) \leq \max\{d(x,z),d(z,y)\}$ for all $x,y,z \in \mathcal{X}$, and $d$ is a tree metric if $d(x,y) + d(v,w) \leq \max\{d(x,v) + d(y,w),d(x,w) + d(y,v)\}$ for all $v,w,x,y \in \mathcal{X}$.

The relation between restricted classes of distances and path lengths in trees has been investigated in detail, e.g., in phylogenetics (Semple & Steel, 2003). A weighted tree $T$ with positive real-valued edge weights $w : E(T) \to \mathbb{R}_{>0}$ represents the distance $d : V(T) \times V(T) \to \mathbb{R}_{>0}$ defined as $d_{(T,w)}(u,v) = \sum_{e \in P(u,v)} w(e)$, where $P(u,v)$ denotes the unique path from $u$ to $v$, for all $u,v \in V(T)$. For every ultrametric $d$ on $\mathcal{X}$ there is a rooted tree $T$ with leaves $\mathcal{X}$ and positive real-valued edge weights, such that (i) $d$ is the path length between leaves in $T$, (ii) all paths from any leaf to the root have equal length. For every tree metric $d$ on $\mathcal{X}$ there is a tree $T$ with $\mathcal{X} \subseteq V(T)$ and positive real-valued edge weights, such that $d$ corresponds to the path lengths in $T$. Note that an ultrametric always is a tree metric. For the clarity of notation we distinguish between the elements of $\mathcal{X}$ and the nodes of a tree $T$ by introducing a bijective map $\varphi : \mathcal{X} \to V(T)$. We will refer to both a label $u \in \mathcal{X}$ and the associated node by the same letter, with the meaning clear from the context. We consider the distance $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geq 0}$ defined as $d(x,y) = d_{(T,w)}(\varphi(x),\varphi(y))$.

#### 2.2. The assignment problem

The assignment problem is a well-studied classical combinatorial problem (Burkard et al., 2012). Given a triple $(A,B,c)$, where $A$ and $B$ are sets of distinct objects with $|A| = |B| = n$ and $c : A \times B \to \mathbb{R}$ a cost function, the problem asks for a one-to-one correspondence $M$ between $A$ and $B$ with minimum costs. The cost of $M$ is $c(M) = \sum_{(a,b) \in M} c(a,b)$. Assuming an arbitrary, but fixed ordering of the elements of $A$ and $B$, an assignment instance can also be given by a cost matrix $C \in \mathbb{R}^{n \times n}$, where $c_{i,j} = c(a_i,b_j)$ and $a_i$ is the $i$th element of $A$ and $b_j$ is the $j$th element of $B$. Note that in this case the input $C$ is of size $\Theta(n^2)$, where otherwise the input size depends on the representation of the cost function $c$. The assignment problem is equivalent to finding a minimum weight perfect matching in a complete bipartite graph on the two sets $A$ and $B$ with edge weights according to $c$. Unless $C$ is
sparse or contains only integral values from a bounded interval, the best known algorithms require cubic time, which is achieved by the well-known Hungarian method.

Here we consider the assignment problem for two sets of objects where the objects are labelled by elements of $\mathcal{X}$. There exists a labelling $\ell : A \cup B \rightarrow \mathcal{X}$ associating each object $a \in A \cup B$ with a label $\ell(a) \in \mathcal{X}$. Furthermore, we may associate objects with tree nodes using the map $\varphi : \ell \rightarrow t$. We then have $c(a, b) = d(\ell(a), \ell(b)) = d((T, a), (\varphi(x), \varphi(y)))$. We denote by $D^\varphi_{OA}(A, B)$ the cost of an optimal assignment between $A$ and $B$, and the assignment problem as the quadruple $(A, B, T, \varphi)$.

2.3. The graph edit distance

The graph edit distance measures the minimum cost required to transform a graph into another graph by adding, deleting and substituting vertices and edges. Each edit operation $o$ is assigned a cost $c(o)$, which may depend on the attributes associated with the affected vertices and edges. A sequence of edit operations $(o_1, \ldots, o_k)$ that transforms a graph $G$ into another graph $H$ is called an edit path from $G$ to $H$. We denote the set of all possible edit paths from $G$ to $H$ by $\mathcal{T}(G, H)$. Let $G$ and $H$ be attributed graphs, the graph edit distance from $G$ to $H$ is defined by

$$d(G, H) = \min \left\{ \sum_{i=1}^{k} c(o_i) \middle| (o_1, \ldots, o_k) \in \mathcal{T}(G, H) \right\}.$$ 

In order to obtain a meaningful measure of dissimilarity for graphs, a cost function must be tailored to the particular attributes that are present in the considered graphs.

Approximating the graph edit distance by assignments. Computing the graph edit distance is an $\text{NP}$-hard problem and solving practical instances by exact approaches if often not feasible. Therefore, Riesen & Bunke (2009) proposed to derive a suboptimal edit path between graphs from an optimal assignment of their vertices, where the assignment costs also encode the local edge structure. For two graphs $G$ and $H$ with vertices $U = \{u_1, \ldots, u_n\}$ and $V = \{v_1, \ldots, v_m\}$, an assignment cost matrix $C$ is created according to

$$C = \begin{bmatrix}
  c_{1,1} & c_{1,2} & \cdots & c_{1,m} \\
  c_{2,1} & c_{2,2} & \cdots & c_{2,m} \\
  \vdots & \vdots & \ddots & \vdots \\
  c_{n,1} & c_{n,2} & \cdots & c_{n,m} \\
  c_{e,1} & \infty & \cdots & \infty \\
  \infty & c_{e,2} & \cdots & 0 \\
  \vdots & \vdots & \ddots & \vdots \\
  \infty & \cdots & \infty & c_{e,m} \\
  \infty & \cdots & \infty & 0 \\
\end{bmatrix},$$

where the entries are estimations of the cost for substituting, deleting and inserting vertices in $G$. In more detail, the entry $c_{i,v}$ is the cost for deleting $u_i$ increased by the costs for deleting the edges incident to $u_i$. The entry $c_{e,j}$ is the cost for inserting $v_j$ and all edges incident to $v_j$. Finally $c_{i,j}$ is the thecost made up of the cost for substituting the vertex $u_i$ by $v_j$ and the cost of an optimal assignment between the incident edges w.r.t. the edge substitution, deletion and insertion costs. An optimal assignment for $C$ allows to derive an edit path between $G$ and $H$. Its cost is not necessarily minimum possible, but Riesen & Bunke (2009) show experimentally that this procedure leads to a sufficiently good approximation of the graph edit distance for many real-world problems.

The costs derived by Riesen & Bunke (2009) are directly related to edit costs of various operations on the graph, but unfortunately these costs are not suitable for our optimal assignment strategy, which must utilise a tree metric. For this reason, we use a different set of costs, described in Section 4. The optimal assignment is recovered using the method described in the next section. This assignment then induces an edit path which is used to compute a good approximation to the edit distance.

3. Optimal assignments under a tree metric

We consider the assignment problem under the assumption that the costs are derived from a tree metric and propose an efficient algorithm for constructing a solution. For a dataset of sets of objects we obtain a distortion-free embedding of the pairwise optimal assignment costs into an $\ell_1$ space.

3.1. Structural results

Let $(A, B, T, \varphi)$ be an assignment instance as described in Section 2.2. We associate the objects of $A$ and $B$ with the nodes of the tree $T$ according to the map $\varphi$, cf. Figure 1a. An assignment $M$ between the objects of $A$ and $B$ is associated with a collection of paths $\mathcal{P}$ in $T$, such that there is a bijection between pairs $(a, b) \in M$ and paths $(\varphi(a), \ldots, \varphi(b)) \in \mathcal{P}$. In particular, the cost of the assignment equals the sum of weighted path lengths, i.e.,

$$c(M) = \sum_{P \in \mathcal{P}} \sum_{e \in P} w(e). \quad (1)$$

We do not construct the set $\mathcal{P}$ explicitly, but use this notion to develop efficient methods and prove their correctness. Using Eq. (1), we can attribute the total costs of an optimal assignment to the individual edges by counting how often they occur on shortest paths. Deleting an edge $uv$ yields two connected components, one containing the node $u$ and the other containing $v$. Let $A^*_{uv}$ and $A^*_{\overline{uv}}$ denote the number of objects in $A$ associated by $\varphi$ with nodes in the connected component containing $u$ and $v$, respectively, cf. Figure 1b.
Figure 1: A tree representing a metric and the objects of an assignment instance associated to its nodes. In this instance \( A = \{a_1, a_2, a_3, a_4, a_5\} \) and these objects are labelled \((t, t, w, w, w)\). Similarly, the five objects in \( B \) are labelled \((s, s, t, w, x)\). The objects are associated to tree nodes by the map \( \phi \), and \( \bigcirc \) denotes the elements of the set \( A \) and \( \bullet \) of \( B \), respectively.

**Lemma 3.1.** Let \( \mathcal{P} \) be the collection of shortest paths associated with an optimal assignment between \( A \) and \( B \) under a cost function represented by the weighted tree \( T \). Each edge \( uv \) in \( T \) appears \( |A_{uv} - B_{uv}| = |A_{uv} - B_{uv}| \) times in a path in \( \mathcal{P} \).

**Proof.** Splitting \( T \) at an edge defines a bipartition of \( A \) and \( B \). Since \( |A| = |B| \) holds, it follows that \( |A_{uv} - B_{uv}| = |A_{uv} - B_{uv}| \) for every edge \( uv \). When \( uv \) appears in \( |A_{uv} - B_{uv}| \) assignment paths, the maximum number of assignments are made within each partition, with all of the smaller of \( A \) and \( B \) assigned within the partition. We may assign at most \( \min\{A_{uv}, B_{uv}\} \) objects within the connected component containing \( u \), and at least the remaining \( |A_{uv} - B_{uv}| \) objects must be assigned to objects in the connected component containing \( v \). Therefore, \( uv \) appears at least this number of times in shortest paths in \( \mathcal{P} \).

It remains to be shown that the assignment cannot be optimal when the edge \( uv \) is contained in more paths. Assume \( \mathcal{P}' \) corresponds to an optimal solution and contains the edge \( uv \) more than \( |A_{uv} - B_{uv}| \) times. Then, there exist at least one element of \( A \) and \( B \) assigned across the partitions with assignment paths \( P_1 = (a_1, \ldots, u, v, \ldots, b_1) \) and \( P_2 = (b_2, \ldots, u, v, \ldots, a_2) \) in \( \mathcal{P}' \) that are not in \( \mathcal{P} \). Note that \( a_1, b_2 \) are in the connected component containing \( u \) and \( b_1, a_2 \) in the component containing \( v \). Consider the paths \( P_1' = (a_1, \ldots, u, \ldots, b_2) \) and \( P_2' = (a_2, \ldots, v, \ldots, b_1) \), which both do not contain \( uv \). The collection of paths \( \mathcal{P}'' = \mathcal{P}' \cup \{P_1', P_2'\} \setminus \{P_1, P_2\} \) also defines an assignment, where the edges are contained in the same number of paths with exception of the edge \( uv \), which appears in two paths less. Since \( w(uv) > 0 \), the associated solution has cost \( c(\mathcal{P}'') = c(\mathcal{P}') - 2w(uv) \) and hence \( \mathcal{P}' \) cannot correspond to an optimal solution, contradicting the assumption. \( \square \)

This result allows us to compute the optimal assignment cost as a weighted sum over the edges in the tree representing the cost metric.

**Theorem 3.2.** Let \((A, B, T, \phi)\) be an assignment instance with tree edge weights \( w \). The cost of an optimal assignment is

\[
D^\phi_{OA}(A, B) = \sum_{uv \in E(T)} |A_{uv} - B_{uv}| \cdot w(uv).
\]

**Proof.** Directly follows from Eq. (1) and Lemma 3.1. \( \square \)

3.2. Constructing an optimal assignment

In order to compute an optimal assignment, and not just its cost, we again associate the objects of \( A \) and \( B \) with the nodes of the tree \( T \). Then we pick an arbitrary leaf \( v \) and assign the maximum possible number of elements between the subsets of \( A \) and \( B \) associated with \( v \). The remaining objects are passed to its neighbour and the considered leaf is deleted. Iterating the approach until the tree is eventually empty, yields an assignment between all objects of \( A \) and \( B \). Algorithm 1 implements this approach.

**Theorem 3.3.** Algorithm 1 computes an optimal assignment in time \( O(n + t) \), where \( n = |A| = |B| \) is the input size and \( t = |V(T)| \) the size of the tree \( T \).

**Proof.** Since \( |A| = |B| \) and every object of \( A \) is associated with exactly one object of \( B \), the algorithm constructs an assignment. The cost of the assignment corresponds to the number of objects that are passed to neighbours along the weighted edges in lines 8 and 9. Whenever a node \( v \) is processed in the while-loop, it has exactly one remaining neighbour \( n \). Since \( v \) is deleted after the end of the iteration, objects are passed along the edge \( nv \) only in this iteration. After calling the procedure \( \text{PAIRELEMENTS} \) in line 6 either \( A_v \) or \( B_v \) or both are empty. Since all objects in the connected component of \( T \setminus nv \) that contains \( v \) must have been passed to \( v \) in previous iterations, exactly \( |A_{uv} - B_{uv}| \) objects are passed to \( n \). This is the number of occurrences of \( nv \) in every optimal solution according to Lemma 3.1. Therefore, \( M \) is an optimal assignment.

The total running time over all iterations for the procedure \( \text{PAIRELEMENTS} \) is \( O(n) \), the size of the assignment. All the other individual operations within the while-loop can be implemented in constant time when using linked lists to
Algorithm 1: Optimal assignment from a cost tree.

Input: Assignment instance \((A, B, T, g)\).
Data: Sets \(A_v\) and \(B_v\) for each vertex \(v\) of \(T\), partial assignment \(M\).
Output: An optimal assignment \(M\).

1. for all \(a \in A\) do \(A_{g(a)} \leftarrow A_{g(a)} \cup \{a\}\) \(\triangleright \) Initialisation
2. for all \(b \in B\) do \(B_{g(b)} \leftarrow B_{g(b)} \cup \{b\}\)
3. \(M \leftarrow \emptyset\)
4. while \(V(T) \neq \emptyset\) do
5. \(v \leftarrow\) arbitrary vertex in \(T\) with degree 1
6. \(M \leftarrow M \cup $\textsc{PairElements}(A_v, B_v)\)
7. \(n \leftarrow $\textsc{Neighbour}(v)\) \(\triangleright \) Get distinct neighbour
8. \(A_n \leftarrow A_n \cup A_v\)
9. \(B_n \leftarrow B_n \cup B_v\)
10. \(T \leftarrow T \setminus v\) \(\triangleright \) Remove the node \(v\) from \(T\)

Procedure \textsc{PairElements}(\(A_v, B_v\))

11. \(N \leftarrow \emptyset\)
12. while \(A_v \neq \emptyset\) and \(B_v \neq \emptyset\) do
13. \(a \leftarrow $\textsc{GetAndRemoveElement}(A_v)\)
14. \(b \leftarrow $\textsc{GetAndRemoveElement}(B_v)\)
15. \(N \leftarrow N \cup \{(a, b)\}\)
16. return \(N\)

Every optimal assignment can be obtained by Algorithm 1 depending on the order in which the objects are retrieved by \textsc{GetAndRemoveElement} in line 13 and 14.

3.3. Improving the running time

We consider the setting, where the map \(g\) and the weighted tree \((T, w)\) encoding the cost metric \(c\) is fixed and the distance \(D_{OA}\) should be computed for a large number of pairs. The individual assignment instances possibly only populate a small fraction of the nodes of \(T\) and only a small subtree may be relevant for Algorithm 1. We show that this subtree can be identified efficiently.

Given a tree \(T\) and a set \(N \subseteq V(T)\), let \(T_N\) denote the minimal subtree of \(T\) with \(N \subseteq V(T_N)\).

Lemma 3.4. Given a tree \(T\) and a set \(N \subseteq V(T)\), the subtree \(T_N\) can be computed in time \(O(|T_N|)\) after a preprocessing step of time \(O(|T|)\).

Proof. In the preprocessing step we pick an arbitrary node of \(T\) as root and compute the depth of every vertex w.r.t. the root using breadth-first search. Let \(d_{\text{min}} = \min\{\text{depth}(v) \mid v \in N\}\). For every node \(v \in N\), we (i) add \(v\) to the result set \(R\), and (ii) if the parent \(p\) of \(v\) is not in \(R\) and \(\text{depth}(p) \geq d_{\text{min}}\), set \(v\) to \(p\) and continue with step (i). Let \(R_{\text{min}} = \{r \in R \mid \text{depth}(r) = d_{\text{min}}\}\). If \(|R_{\text{min}}| > 1\), add the parents of all \(r \in R_{\text{min}}\) to \(R\), decrease \(d_{\text{min}}\) by one. Repeat this step until \(|R_{\text{min}}| = 1\). Eventually, we have \(N \subseteq R\) and \(T_N = T[R]\). Every node in \(R\) is processed only once and the running time is \(O(|R|) = O(|T_N|)\).

Assuming that the tree and the depth of all nodes are given, the result directly improves the running time of Algorithm 1 to \(O(n + |T_N|)\).

3.4. Embedding optimal assignment costs

We show how sets can be embedded in a vector space such that the Manhattan distance between these vectors equals the optimal assignment costs between sets w.r.t. a given cost function. Let \(D = \{D_1, \ldots, D_k\}\) be a dataset with \(|D_i| = |D_j|\) for all \(i, j \in \{1, \ldots, k\}\). Let the cost function \(c\) between the objects \(\bigcup_i D_i\) be determined by a tree distance represented by the weighted tree \(T\), and the map \(g\) from the objects to the nodes of the tree. We consider the following map \(\phi_c\) from \(A \in D\) to points in \(R^d\) with \(d = |E(T)|\) and components indexed by the edges of \(T\):

\[\phi_c(A) = [A_{uv} \cdot w(uv)]_{uv \in E(T)}\]

The Manhattan distance between these vectors is equal to the optimal assignment costs between the sets.

Theorem 3.5. Let \(c\) be a cost function and \(\phi_c\) defined as above, then \(D_{OA}^c(A, B) = \|\phi_c(A) - \phi_c(B)\|_1\).

Proof. We calculate

\[\|\phi_c(A) - \phi_c(B)\|_1 = \sum_{uv \in E(T)} |A_{uv} - B_{uv}| \cdot w(uv)\]

where the last equality follows from the Theorem 3.2.

This makes the optimal assignment costs available to fast indexing methods and nearest-neighbour search algorithms, e.g., following the locality sensitive hashing paradigm.

### 4. Approximating the graph edit distance in linear time

We combine our assignment algorithm with the idea of Riesen & Bunke (2009) to approximate the graph edit distance detailed in Section 2.3. To this end, we propose two
methods for constructing a tree distance, such that the optimal assignment between the vertices of two graphs w.r.t. to these distances is suitable for approximating the graph edit distance. In order to quantify discrete and structural differences, we propose to use the Weisfeiler-Lehman method and, for graphs with continuous labels, hierarchical clustering. Note that both approaches can be combined to form a tree taking both, discrete and continuous labels, into account. The tree distances we consider are in fact ultrametrics. We discuss the general limitations of ultrametrics in representing distances as used for the cost matrix of Eq. (2.3) in the appendix, Section B. In order to compare graphs of different size we introduce artificial vertices that represent vertex insertion and deletion operations.

4.1. Weisfeiler-Lehman trees for discrete and structural differences

Weisfeiler-Lehman refinement (WL), also known as colour refinement or naïve vertex classification, is a classical heuristic for graph isomorphism testing. It iteratively refines partitions of the vertices of a graph, where the vertices in the same cell are said to have the same colour. In each iteration two vertices with the same colour obtain different new colours if their neighbourhood differs w.r.t. the current colouring. More formally, given a parameter \( h \) and a graph \( G \) with initial colours \( \tau_0 \), a sequence \( (\tau_1, \ldots, \tau_h) \) of refined colours is computed, where \( \tau_i \) is obtained from \( \tau_{i-1} \) by the following procedure. For every vertex \( v \in V(G) \), sort the multiset of colours \( \{ \tau_{i-1}(u) : uv \in E(G) \} \) to obtain a unique sequence of colours and add \( \tau_{i-1}(v) \) as first element. Assign a new colour \( \tau_i(v) \) to every vertex \( v \) by employing an injective mapping from colour sequences to new colours. It was observed by Kriege et al. (2016) that colour refinement applied to a set of graphs under the same injective mapping yields a hierarchy of partitions of the vertices, which forms a tree. Let \( \varphi \) associate the vertices with the node representing their final colour after \( h \) iterations. Then the path length between two nodes in the tree represents the number of refinement steps in which the associated vertices have different colours. Assuming that \( h \) is fixed, we obtain a linear running time for approximating the graph edit distance with Theorem 3.3 and Lemma 3.4.

The WL method is used successfully to derive efficient and expressive graph kernels (Shervashidze et al., 2011; Kriege et al., 2016). The prediction accuracy obtained with such kernels crucially depends on the number refinement iterations \( h \), which is typically determined by expensive grid search, e.g., from \( \{0, \ldots, 7\} \). Note that our approach to approximate the graph edit distance is not sensitive to a particular choice of \( h \), but can be expected to always benefit from more iterations. If \( h \) is chosen such that \( \tau_h = \tau_{h+1} \), we can in fact show that a slight modification of our algorithm guarantees that the computed edit distance is zero if and only if \( G \) and \( H \) are isomorphic, assuming that \( G \) is amenable to the WL procedure. This means that WL succeeds in distinguishing \( G \) from any non-isomorphic graph \( H \) (Arvind et al., 2015). Then, there is a choice for the module PAIRELEMENTS such that Algorithm 1 is guaranteed to construct an isomorphism between two copies of \( G \), i.e., the algorithm output is optimal. These choices for PAIRELEMENTS are explicitly constructed in the appendix, Section A.

4.2. Hierarchical clustering for continuous labels

We apply the bisecting \( k \)-means algorithm (Steinbach et al., 2000a;b) to obtain a hierarchical clustering of the continuous vertex labels of all graphs. This is then used as the tree defining the assignment costs. We use Lloyd’s algorithm (Lloyd, 1982) for each 2-means problem and perform bisection steps until 300 leaves are created. Assuming that the number of reassignments in 2-means is constant as well as the dimension of the node labels, we obtain a linear running time for creating the tree. With Theorem 3.3 and Lemma 3.4 this again yields a linear time method for approximating the graph edit distance.

5. Experimental Evaluation

Our goal in this section is to answer the following questions experimentally.

**Q1** How does our approach scale w.r.t. the graph and dataset size compared to other methods?

**Q2** How accurately does it approximate the graph edit distance for common datasets?

**Q3** How does it perform regarding runtime and accuracy in classification tasks?

**Q4** How does our method compare to other approaches for graph classification?

5.1. Method

We have implemented the following methods for computing or approximating the graph edit distance in Java using the same code base where possible.

**Exact** Binary linear programming approach to compute the graph edit distance exactly (Lerouge et al., 2017). We implemented the most efficient formulation (F2) and solved all instance using Gurobi 7.5.2.

**BP** Approximate graph edit distance using the assignment problem as proposed by Riesen & Bunke (2009).

**Greedy** The greedy graph edit distance proposed by Riesen et al. (2015a) solving the assignment problem by a row-wise greedy algorithm.

**Linear** Our approach based on assignments under a tree distance. For graphs with discrete labels we used Weisfeiler-Lehman trees with \( h = 7 \), and bisecting
\( k \)-means clustering otherwise.

The experiments were conducted using Java v1.8.0 on an Intel Core i7-3770 CPU at 3.4GHz (Turbo Boost disabled) with 16GB of RAM. The methods BP, Greedy and Linear use a single processor only, the Gurobi solver for the Exact method was allowed to use all four cores with additional Hyper-Threading. We used the graph classification benchmark sets contained in the IAM Graph Database (Riesen & Bunke, 2008)\(^1\) and the repository of benchmark datasets for graph kernels (Kersting et al., 2016). The datasets AIDS, Mutagenicity and NCI1 represent small molecules and have discrete labels only. The Letter datasets have continuous vertex labels representing 2D coordinates and differ w.r.t. the level of distortion, (L)–low, (M)–medium and (H)–high. In order to systematically investigate the runtime we generated random graphs according to the Erdős–Rényi model with edge probability 0.15. We used the predefined train, test and validation sets when available or generated them randomly using \( 1/3 \) of the objects for each set, balanced by class label. We performed \( k \)-nearest neighbours classification based on the graph edit distance. The costs for vertex insertion and deletion were both set to \( \tau_{\text{vertex}} \) and the costs for insertion and deletion of edges were set to \( \tau_{\text{edge}} \). The costs for substituting vertices or edges are determined by the Euclidean distance in case of continuous labels. In case of discrete labels we assume cost 0 for equal labels and 1 otherwise. We use the validation set to select the parameters \( k \in \{1,3,5\}, \tau_{\text{vertex}} \in \{0.1,0.5,0.9,1.3,1.7\} \) and \( \tau_{\text{edge}} \in \{0.1,0.5,0.9,1.3,1.7\} \) by grid search. The approach resembles the experimental settings used by Riesen & Bunke (2009). The reported runtimes were either obtained using the selected parameters of \( \tau_{\text{vertex}} = \tau_{\text{edge}} = 1 \). For the linear method, the runtimes include the time for constructing the tree.

For comparison with other approaches to graph classification, we used two graph kernels as a baseline. The Graph-Hopper kernel (GH) (Feragen et al., 2013) supports graphs with discrete and continuous labels by applying a Dirac or Gaussian kernel. The Weisfeiler-Lehman optimal assignment kernel (WLOA) (Kriege et al., 2016) supports only graphs with discrete labels. We used the \( C \)-SVM implementation LIBSVM (Chang & Lin, 2011), selecting \( C \in \{10^{-3},10^{-2},\ldots,10^3\} \) and \( h \in \{0,1,\ldots,7\} \) using the validation set.

5.2. Results

We report on our experimental results and answer our research questions.

\( Q1 \) Figure 2 shows the growth of the runtime with increasing graph and dataset size. Our method is the only one of those studied that scales to large graphs. The number of distance computations and thus the runtime of all methods grows quadratically with the dataset size. Even for the small random graphs on 15 vertices we generated, our method is more than one order of magnitude faster than other approximate methods for datasets of moderate size.

\( Q2 \) To compare how accurately the graph edit distance is computed, we have selected 500 pairs of graphs at random from each IAM dataset and computed their graph edit distance by all four methods. Figure 3 shows how the distance computed by the Linear method compares to the distances obtained by the other three methods. Points below the diagonal line represent pairs of graphs, for which the edit distance computed by Linear is actually smaller than the one computed by the competing approach. Compared to the Greedy approach the Linear method appears to give slightly better results on an average. For the datasets Mutagenicity, Letter (L), (M) and (H) there are more points below the diagonal than above the diagonal. When comparing to BP, this is still the case for the Mutagenicity dataset, but not for the Letter datasets. This can be explained by the fact that continuous distances for several points cannot be represented by a tree metric without distortion. In order to compare with the exact method on Mutagenicity and AIDS, we introduced a timeout of 100 seconds for each distance computation. This was necessary since hard instances may require more than several hours. In case of a timeout the best solution found so far is used, which is not guaranteed to be optimal. The Linear method shows a clear divergence from the optimal solutions, in particular for pairs of graphs with a high (exact) edit distance. However, it is likely that non-optimal solutions in this case do not harm a nearest neighbours classification.

\( Q3 \) Table 1 summarises the results of the classification experiments. The Linear approach provides a high classification accuracy comparable to BP and Greedy. For the

\( \text{(a) Graph size} \)

\( \text{(b) Dataset size} \)

Figure 2: Runtime in milliseconds to (a) compute the graph edit distance between two random graphs; and (b) between all pairs of graphs in a dataset of random graphs.
dataset Mutagenicity and NCI1 it even performs better than the other approaches. This can be explained by the ability of the Weisfeiler-Lehman tree to exploit more graph structure than BP. For the Letter datasets, the Linear method is on a par with the other methods for the version with low distortion, but performs slightly worse when the distortion increases. This observation is in accordance with the approximation quality achieved for the datasets, cf. Figure 3. The Linear method clearly outperforms all other approaches in terms of runtime. This becomes in particular clear for the dataset Mutagenicity, which contains large graphs with 30.32 vertices on an average.

Q4 The GraphHopper kernel performs worse than our Linear approach w.r.t. running time and classification accuracy. WLOA can only be applied to the molecular datasets with discrete labels. For these it performs exceptionally well regarding both accuracy and runtime. The result suggests that the notion of similarity provided by the graph edit distance is less suitable for this classification task.

6. Conclusion

We have shown that optimal assignments can be computed efficiently for tree metric costs. Although this is a severe restriction, we designed such costs functions suitable for the challenging problem of graph matching. Our approach allows to embed the optimal assignment costs in an $\ell_1$ space. It remains future work to exploit this property, e.g., for efficient nearest neighbour search in graph databases.
Acknowledgements

This work was supported by the German Science Foundation (DFG) within the Collaborative Research Center SFB 876 “Providing Information by Resource-Constrained Data Analysis”, project A6 “Resource-efficient Graph Mining”.

References

Arvind, V., Köbler, J., Rattan, G., and Verbitsky, O. On the power of color refinement. In Kosowski, A. and Walukiewicz, I. (eds.), Fundamentals of Computation Theory, pp. 339–350. Springer International Publishing, 2015. ISBN 978-3-319-22177-9. doi: 10.1007/978-3-319-22177-9_26.

Bunke, H. On a relation between graph edit distance and maximum common subgraph. Pattern Recognition Letters, 18(8):689–694, 1997. ISSN 0167-8655. doi: 10.1016/S0167-8655(97)00060-3.

Burkard, R. E., Dell’Amico, M., and Martello, S. Assignment Problems. SIAM, 2012. doi: 10.1137/1.9781611972238.

Chang, C.-C. and Lin, C.-J. LIBSVM: A library for support vector machines. ACM Transactions on Intelligent Systems and Technology, 2:27:1–27:27, May 2011. ISSN 2157-6904. doi: http://doi.acm.org/10.1145/1961189.1961199. Software available at http://www.csie.ntu.edu.tw/˜cjlin/libsvm.

Feragen, A., Kasenburg, N., Petersen, J., Bruijne, M. D., and Borgwardt, K. Scalable kernels for graphs with continuous attributes. In Burges, C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. (eds.), Advances in Neural Information Processing Systems 26, pp. 216–224, 2013. Erratum available at http://image.diku.dk/aasa/papers/graphkernels_nips_erratum.pdf.

Fröhlich, H., Wegner, J. K., Sieker, F., and Zell, A. Optimal assignment kernels for attributed molecular graphs. In Proceedings of the 22nd international conference on Machine learning, ICML ’05, pp. 225–232, New York, NY, USA, 2005. ACM. ISBN 1-59593-180-5.

Garey, M. R. and Johnson, D. S. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, 1979. ISBN 0-7167-1044-7.

Grauman, K. and Darrell, T. The pyramid match kernel: Efficient learning with sets of features. J. Mach. Learn. Res., 8:725–760, May 2007a. ISSN 1532-4435.

Grauman, K. and Darrell, T. Approximate correspondences in high dimensions. In Schölkopf, B., Platt, J. C., and Hoffman, T. (eds.), Advances in Neural Information Processing Systems 19, pp. 505–512. MIT Press, 2007b.

Johansson, F. D. and Dubhashi, D. Learning with similarity functions on graphs using matchings of geometric embeddings. In Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD ’15, pp. 467–476, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3664-2. doi: 10.1145/2783258.2783341.

Johnson, D. S. The NP-completeness column. ACM Trans. Algorithms, 1(1):160–176, July 2005. ISSN 1549-6325. doi: 10.1145/1077464.1077476.

Kersting, K., Kriege, N. M., Morris, C., Mutzel, P., and Neumann, M. Benchmark data sets for graph kernels, 2016. URL http://graphkernels.cs.tu-dortmund.de.

Kriege, N. M., Giscard, P.-L., and Wilson, R. On valid optimal assignment kernels and applications to graph classification. In Advances in Neural Information Processing Systems 29 (NIPS), pp. 1623–1631. Curran Associates, Inc., 2016.

Lerouge, J., Abu-Aisheh, Z., Raveaux, R., Héroux, P., and Adam, S. New binary linear programming formulation to compute the graph edit distance. Pattern Recognition, 72(Supplement C):254 – 265, 2017. ISSN 0031-3203.

Lin, C.-L. Hardness of approximating graph transformation problem. In Du, D.-Z. and Zhang, X.-S. (eds.), Algorithms and Computation, pp. 74–82, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg. ISBN 978-3-540-48653-4.

Lloyd, S. Least squares quantization in pcm. IEEE Transactions on Information Theory, 28(2):129–137, March 1982. ISSN 0018-9448. doi: 10.1109/TIT.1982.1056489.

Riesen, K. and Bunke, H. Iam graph database repository for graph based pattern recognition and machine learning. In da Vitoria Lobo, N., Kasparis, T., Roli, F., Kwok, J. T., Georgioupolous, M., Anagnostopoulos, G. C., and Loog, M. (eds.), Structural, Syntactic, and Statistical Pattern Recognition: Joint IAPR International Workshop, SSPR & SPR 2008, Orlando, USA, December 4-6, 2008. Proceedings, pp. 287–297, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-89689-0. doi: 10.1007/978-3-540-89689-0_33.

Riesen, K. and Bunke, H. Approximate graph edit distance computation by means of bipartite graph matching. Image and Vision Computing, 27(7):950 – 959, 2009. ISSN 0262-8856. 7th IAPR-TC15 Workshop on Graph-based Representations (GbR 2007).
Riesen, K., Ferrer, M., Dornberger, R., and Bunke, H. Greedy graph edit distance. In Perner, P. (ed.), *Machine Learning and Data Mining in Pattern Recognition*, pp. 3–16, Cham, 2015a. Springer International Publishing. ISBN 978-3-319-21024-7.

Riesen, K., Ferrer, M., Fischer, A., and Bunke, H. Approximation of graph edit distance in quadratic time. In Liu, C.-L., Luo, B., Kropatsch, W. G., and Cheng, J. (eds.), *Graph-Based Representations in Pattern Recognition*, pp. 3–12, Cham, 2015b. Springer International Publishing. ISBN 978-3-319-18224-7.

Sanfeliu, A. and Fu, K.-S. A distance measure between attributed relational graphs for pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(3): 353–362, 1983.

Semple, C. and Steel, M. *Phylogenetics*. Oxford lecture series in mathematics and its applications. Oxford University Press, 2003.

Shervashidze, N., Schweitzer, P., van Leeuwen, E. J., Mehlhorn, K., and Borgwardt, K. M. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12:2539–2561, 2011.

Stauffer, M., Tschachtli, T., Fischer, A., and Riesen, K. A survey on applications of bipartite graph edit distance. In Foggia, P., Liu, C.-L., and Vento, M. (eds.), *Graph-Based Representations in Pattern Recognition*, pp. 242–252, Cham, 2017. Springer International Publishing. ISBN 978-3-319-58961-9.

Steinbach, M., Karypis, G., and Kumar, V. A comparison of document clustering techniques. In *KDD Workshop on Text Mining*, 2000a.

Steinbach, M., Karypis, G., and Kumar, V. A comparison of document clustering techniques. Technical report, Department of Computer Science and Engineering, University of Minnesota, 2000b.

Vert, J.-P. The optimal assignment kernel is not positive definite. *CoRR*, abs/0801.4061, 2008.

Vishwanathan, S. V. N., Schraudolph, N. N., Kondor, R. I., and Borgwardt, K. M. Graph kernels. *Journal of Machine Learning Research*, 11:1201–1242, 2010.

Zeng, Z., Tung, A. K. H., Wang, J., Feng, J., and Zhou, L. Comparing stars: On approximating graph edit distance. *Proc. VLDB Endow.*, 2(1):25–36, August 2009. ISSN 2150-8097. doi: 10.14778/1687627.1687631.
A. Modification of PAIR ELEMENTS for WL-amenable graphs

These results are due to two properties of WL-amenable graphs. To introduce these, we will need to consider the elements of the stable partition of the vertex set of \( G \) at the end of the WL-procedure, called cells. For any cell \( X \), the graph \( G[X] \) denotes the induced subgraph of \( G \) with vertex set \( X \).

First, the stable partition of an amenable graph coincides with the orbit partition of the automorphism group of this graph. In other terms, if there exists a graph automorphism between two vertices \( v \) and \( v' \) on \( G \), then \( v \) and \( v' \) have the same colour in the stable partition (Arvind et al., 2015). This guarantees that all isomorphisms between copies of \( G \), i.e. automorphisms of \( G \), relate vertices belonging to the same cell. Second, \( G[X] \) is of only 5 possible types (see Lemma 3 of (Arvind et al., 2015)) : i) empty, ii) complete; iii) a 5-cycle; iv) a matching graph \( mK_2 \) (multiple copies of \( K_2 \)) or v) the complement of a matching graph \( mK_2 \). Note that for two isomorphic graphs PAIR ELEMENTS is called exactly for the matching cells of the stable partitions and no elements are passed to neighbours. We modify the function to first determine the nature of \( G[X] \). If it is empty or complete, any vertex matching is an automorphism of \( G \). If \( G[X] \) is a five cycle, the algorithm picks two vertices at random on the two 5-cycles to be matched, match these two vertices, then select and match their right (or left) neighbours and proceed thus until vertex exhaustion. If \( G[X] \) is a matching graph, picks two vertices at random on the two \( mK_2 \) to be matched, match these two vertices and then match their unique neighbours. Proceed thus until vertex exhaustion. If \( G[X] \) is none of the above, it is the complement of a matching graph \( mK_2 \). Then the algorithm picks two vertices at random on the two \( mK_2 \) to be matched, match them, then match the neighbours of these two vertices with any other vertices than with one another. Proceed until vertex exhaustion.

In the end the algorithm has constructed a valid automorphism of \( G \), or equivalently an isomorphism between two copies of \( G \). All automorphisms can be constructed this way by modifying the module PAIR ELEMENTS such that all possible matching are realised rather a random possible one by adding a loop over all possible choices in cases i-v).

The procedure PAIRADIO shown in Algorithm 2 replaces PAIR ELEMENTS in Algorithm 1 and implement this idea.

### Algorithm 2: Pair elements for local structure.

```plaintext
Procedure PAIRADIO(A_v, B_v)
1 X ← ∅
2 while A_v ≠ ∅ and B_v ≠ ∅ do
3 a ← GETANDREMOVEELEMENT(A_v)
4 b ← GETANDREMOVEELEMENT(B_v)
5 X ← X ∪ \{(a, b)\}
6 return X
```

to itself. Let \( A = (a_1, \ldots, a_{2n}) \) and \( B = (b_1, \ldots, b_{2n}) \) and consider \( c(a_1, b_{n+2}) = \max\{c(a_1, b_{n+1}), c(b_{n+1}, b_{n+2})\} \). We have \( c(a_1, b_{n+2}) = \infty \) and \( c(b_{n+1}, b_{n+2}) \) not specified by \( C \). However, \( c(b_{n+1}, b_{n+2}) = \max\{c(a_n+1, b_{n+1}), c(a_{n+1}, b_{n+2})\} = 0 \) and, thus, a contradiction to the strong triangle inequality, unless \( c_{1,e} = \infty \). Therefore, we have to modify the definition of the cost matrix. The entries \( \infty \) in the upper right and lower left corner have been introduced with the argument, that every node can be inserted and deleted at most once (Riesen & Bunke, 2009). This, however, is already guaranteed, since the assignment is a bijection. We simplify the cost matrix as follows

\[
C_{\text{ultra}} = \begin{bmatrix}
c_{1,1} & c_{1,2} & \cdots & c_{1,m} & \tau & \tau & \cdots & \tau \\
c_{2,1} & c_{2,2} & \cdots & c_{2,m} & \tau & \tau & \cdots & \vdots \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \ddots & \vdots \\
c_{n,1} & c_{n,2} & \cdots & c_{n,m} & \tau & \cdots & \tau & \tau \\
\tau & \tau & \cdots & \tau & 0 & 0 & \cdots & 0 \\
\tau & \vdots & \ddots & \vdots & 0 & 0 & \ddots & \vdots \\
\tau & \vdots & \vdots & \tau & \vdots & \ddots & \ddots & 0 \\
\tau & \cdots & \tau & \tau & 0 & 0 & \cdots & 0 
\end{bmatrix}
\]

where \( \tau \) is the cost for vertex deletion and insertion. Moreover, we assume that the vertex substitution costs \( c_{i,j} \leq \tau \) for \( i \in \{1, \ldots, n\}, j \in \{1, \ldots, m\} \) and can be represented by an ultrametric tree \( T_c \). We can extend this tree for vertex insertion and deletion as defined by \( C_{\text{ultra}} \) by adding a node \( n \) to the root, where the edge to the parent has weight \( \tau \). Just like the matrix (2) contains additional rows and columns for vertex insertion and deletion, we associate artificial vertices with the node \( n \) via the map \( \varphi \). Note that these can be matched at zero cost at \( n \), which represents the bottom right submatrix of (2) filled with 0.

B. Limitations of ultrametrics

The cost matrix of Eq. (2.3) is easily seen not to be in accordance with the strong triangle inequality. Consider the cost matrix \( C \) obtained for a graph \( G \) with \( n \) vertices compared

C. Dataset statistics

The statistics of the graph datasets used are summarized in Table 2. All data sets are publicly available for download (Riesen & Bunke, 2008; Kersting et al., 2016).
Table 2: Dataset statistics and properties.

| Data set          | Properties | Labels | Attributes | Reference                  |
|-------------------|------------|--------|------------|----------------------------|
|                   | Graphs | Classes | Avg. | Avg. | Vertex | Edge | Vertex | Edge |                   |
| AIDS              | 2000   | 2       | 15.69 | 16.20 | +      | +    | -      | -    | (Riesen & Bunke, 2008) |
| Letter (L)        | 2250   | 15      | 4.68  | 3.13  | -      | -    | + (2)  | -    | (Riesen & Bunke, 2008) |
| Letter (M)        | 2250   | 15      | 4.67  | 4.50  | -      | -    | + (2)  | -    | (Riesen & Bunke, 2008) |
| Letter (H)        | 2250   | 15      | 4.67  | 4.50  | -      | -    | + (2)  | -    | (Riesen & Bunke, 2008) |
| Mutagenicity      | 4337   | 2       | 30.32 | 30.77 | +      | +    | -      | -    | (Riesen & Bunke, 2008) |
| NCI1              | 4110   | 2       | 29.87 | 32.30 | +      | -    | -      | -    | (Shervashidze et al., 2011) |