A Binary Linear Programming Formulation of the Graph Edit Distance

Derek Justice and Alfred Hero

Department of Electrical Engineering and Computer Science

University of Michigan

Ann Arbor, MI 48109

Abstract

A binary linear programming formulation of the graph edit distance for unweighted, undirected graphs with vertex attributes is derived and applied to a graph recognition problem. A general formulation for editing graphs is used to derive a graph edit distance that is proven to be a metric provided the cost function for individual edit operations is a metric. Then, a binary linear program is developed for computing this graph edit distance, and polynomial time methods for determining upper and lower bounds on the solution of the binary program are derived by applying solution methods for standard linear programming and the assignment problem. A recognition problem of comparing a sample input graph to a database of known prototype graphs in the context of a chemical information system is presented as an application of the new method. The costs associated with various edit operations are chosen by using a minimum normalized variance criterion applied to pairwise distances between nearest neighbors in the database of prototypes. The new metric is shown to perform quite well in comparison to existing metrics when applied to a database of chemical graphs.

Index Terms

Graph algorithms, Similarity measures, Structural Pattern Recognition, Graphs and Networks, Linear Programming, Continuation (homotopy) methods

I. Introduction

Attributed graphs provide convenient structures for representing objects when relational properties are of interest. Such representations are frequently useful in applications ranging from computer-aided drug design to machine vision. A familiar machine vision problem is to recognize specific objects within an image. In this case, the image is processed to generate a representative graph based on structural characteristics, such a region adjacency graph or a line adjacency graph, and vertex attributes may be assigned according to characteristics of the region to which each vertex corresponds [1]. This representative graph is then compared to a database of prototype or model graphs in order to identify and classify the object of interest. Face identification [2] and symbol recognition [3] are among the problems in machine vision where graphs have been utilized recently. In this context, a reliable and speedy method for comparing graphs is important. Many heuristics and simplifications have been developed and employed for this purpose in a variety of applications [4].

Comparing graphs in the context of graph database searching has also found significant application in the pharmaceutical and agrochemical industries. The attributed graphs of interest are so called chemical graphs which are derived from chemical structure diagrams. Graphical representations are of great utility here because of the *similar property principle*, which states that molecules with similar structures will exhibit similar chemical properties [5]. Thus databases of these chemical graphs are often searched by comparing with a query graph to aid in the design of new chemicals or medicines [6]. Various techniques have been designed for processing the graphs for structural features and generating bit strings (referred to as *fingerprints*) based on the presence or absence of such features [7]. Since the fingerprints can be rapidly compared, some pre-screening or clustering is often done based on these to eliminate all but the most similar graphs to a given input graph. The remaining graphs may then be compared to the query using a more sophisticated (and more computationally demanding) method. Distance metrics based on the maximum common subgraph are frequently used in this role [8], [9].

Although computing the maximum common subgraph (MCS) is no small task (indeed, it is an NP-Hard problem [10]), several graph distance metrics have been proposed that use the size of the MCS. One such metric is given in [11], with a slight modification presented in [12]. A different MCS-based metric is presented in [13] specifically for application to chemical graphs. An alternate metric that uses the MCS along with the minimum common supergraph has also been proposed [14]. For related structures, such as attributed trees, it is often possible to derive distance metrics based on the maximum common substructure that operate in polynomial time [15]. An intimate relationship between graph comparison and graph (or subgraph) isomorphism is readily apparent in these examples because the MCS defines subgraphs in the two graphs being compared that are isomorphic. Indeed, computing a graph distance metric often requires the computation of some sort of isomorphism (aka matching) between graphs.

An exact graph isomorphism defines a mapping between the nodes of two attributed graphs so that their structures (vertex attributes along with edges) exactly coincide. As one might expect, this is also a challenging computational problem in general although it has not been shown to be NP-Complete [10]. As with subgraph isomorphism, polynomial time algorithms are available for certain restricted classes of graphs [16]. Algorithms for general graph isomorphism that are shown to be quite speedy in practice are given in [17], [18]. Such algorithms typically take advantage of vertex attributes to efficiently prune a search tree constructed for finding an isomorphism. Graphs obtained from real objects are rarely isomorphic, however, so it is useful to consider inexact or error-correcting graph isomorphisms (ECGI) that allow for graphs to nearly (but not exactly) coincide [19]. As the name suggests, the lack of exact isomorphism can be caused by measurement noise or errors in a sample graph when compared to a model graph. On the other hand, when comparing two model graphs one might interpret such 'errors' as capturing the essential differences between the two graphs.

Error-correcting graph matching attempts to compute a mapping between the vertices of two graphs so that they approximately coincide, realizing that the graphs may not be isomorphic. Many suboptimal approaches exist to tackle this problem [20], [21], [22], [23]. The adjacency matrix eigendecomposition approach of [21] gives fast suboptimal results, however it is only applicable to graphs having adjacency matrices with no repeated eigenvalues. Graphs with a low degree of connectivity will often have adjacency matrices with multiple zero eigenvalues.

Heuristics are used in the graduated assignment type methods of [22], [23] to significantly reduce the exponential complexity of the original problem. These methods can be applied to very large graphs; however they require several tuning parameters to which the performance of the algorithm is quite sensitive. Unfortunately, no systematic method for choosing these parameters is provided. The linear programming approach of [20] gives good results in a reasonable amount of time for graphs having the same number of vertices. The authors of [20] use the linear program to minimize a matrix norm similarity metric.

The graph edit distance (GED) is a convenient and logical graph distance metric that arises naturally in the context of error-correcting graph matching [19], [24], [25]. It can also be viewed as an extension of the string edit distance [26]. The basic idea is to define graph edit operations such as insertion or deletion of a node/vertex or relabeling of a vertex along with costs associated with each operation. The graph edit distance between two graphs is then just the cost associated with the least costly series of edit operations needed to make the two graphs isomorphic. The optimal error-correcting graph isomorphism can be defined as the resulting isomorphism after performing this optimal series of edits. Furthermore, it has been shown that the optimal ECGI under a certain graph edit cost function will find the MCS [24]. Enumeration procedures for computing such optimal matchings have been proposed [27], [28], [19]. These procedures are applicable for only small graphs. In [29], [30], [31], probabilistic models of the edit operations are proposed and used to develop MAP estimates of the optimal ECGI. It is not clear in all applications, however, what is the appropriate model to use for the edit probabilities. As with previous metrics, efficient algorithms have been developed for computing edit distances on trees with certain structures [32], [33], [34].

The graph edit distance is parameterized by a set of edit costs. The flexibility provided by these costs can be very useful in the context of a standard recognition problem described earlier of matching a sample input graph to a database of known prototype graphs [35]. If chosen appropriately, the costs can capture the essential features that characterize differences among the prototype graphs. Recently, methods for choosing these costs that are best from a recognition point of view have been presented. In [36], the EM algorithm is applied to assumed Gaussian mixture models for edit events in order to choose costs that enforce similarity (or dissimilarity) between specific pairs of graphs in a training set. An application for matching images based on their shock graphs [37] uses the tree edit distance algorithm in [34] and chooses edit costs based on local shape differences within shock graphs corresponding to similar images. In a chemical graph recognition application, heuristics are used to choose the edit costs of a string edit distance between strings formed from the maximal paths between vertices in the graphs [38]. Related studies have also been done into the effectiveness of weighting the presence or absence of certain substructures differently when comparing fingerprints derived from chemical graphs [39].

In this paper, we provide a formulation of the graph edit distance whereby error-correcting graph matching may be performed by solving a binary linear program (BLP–that is a linear program where all variables must take values from the set $\{0, 1\}$). We first present a general framework for computing the GED between attributed, unweighted graphs by treating them as subgraphs of a larger graph referred to as the edit grid. It is argued that the edit grid need only have as many vertices as the sum of the total number of vertices in the graphs being compared. We show that graph editing is equivalent to altering the state of the edit grid and prove that the GED derived in this way is a metric provided the cost function for individual edit operations is a metric. We then use the adjacency matrix representation to formulate a binary linear program to solve for the GED. Since solving a BLP is NP-Hard [10], we show how to obtain upper and lower bounds for the GED in polynomial time by using solution techniques for standard linear programming and the assignment problem [40]. These bounds may be useful in the event that the problem is so large that solving the BLP is impractical.

We also present a recognition problem [35] that demonstrates the utility of the new method in the context of a chemical information system. Suppose there is a database of prototype chemical graphs to which a sample graph is to be compared as described earlier. The experiment proceeds in two stages: edit cost selection followed by recognition of a perturbed prototype graph via a minimum distance classifier. We provide a method for choosing the edit costs that is purely nonparametric and is based on the assumption (or prior information) that the graphs in the database should be uniformly distributed. The edit costs are chosen as those that minimize the normalized variance of pairwise distances between nearest neighbor prototypes, thereby uniformly distributing them in the metric space of graphs define by the GED. Note that a metric which uniformly distributes nearest neighbors in the database essentially equalizes the probability of classification error with a minimum distance classifier, thereby minimizing the worst case error. This method is similar to the use of spherical packings for error-correcting code design, where the distances between all nearest-neighbor code points are the same [41]. Also, providing such homogeneous sets of graphs is desirable in chemical applications for certain structure-activity experiments [6]. This computation involves matching all pairs of prototypes in a neighborhood and tabulating the edits between them. These are provided as inputs to a single convex program to solve for the optimal edit costs. This is one possible method for choosing edit costs; other methods might certainly be concocted to accommodate whatever prior information about the data at hand is available.

We test our algorithm on a database of 135 chemical graphs derived from a set of similar biochemical molecules [42]. Our GED metric is compared with the MCS based metrics proposed in [13], [11]. Indeed, the similarity of molecules in this database indicates it is a good candidate for our method of edit cost selection. We first compute the optimal edit costs as previously described and show that our metric equipped with these costs more uniformly distributes the prototype graphs than either MCS metric. The recognition problem is investigated next by generating sample graphs through random perturbations on the prototype graphs; thus we consider a scenario where the ECGI is used to 'fix' errors between sample and model graphs. Each sample graph is matched to every prototype in the database in an effort to recognize which prototype was perturbed to create the sample graph, and a classification ambiguity index is computed. The GED metric is found to perform better with respect to certain types of edit and worse with respect to others than the MCS metrics. However, when random edits are applied, the GED typically performs better.

This paper is organized as follows. Section II presents the bulk of the theory. Within Section II, we first present the general framework for computing the GED and prove that it results in a metric provided the edit cost function is a metric. This is followed by the development of the binary linear program to compute the graph edit distance along with a description of polynomial-time solutions for upper and lower bounds on the GED. Finally a description of edit cost selection for a graph recognition problem is given, and it is shown that the resulting problem is a convex program. Section III presents the results of the graph recognition problem applied to a database of chemical graphs, and Section IV provides some concluding remarks.

II. Theory

We first introduce a framework for edits on the set of unweighted, undirected graphs with vertex attributes based on relabeling of graph elements (vertices and edges). Suppose we wish to find the graph edit distance between graphs G_0 and G_1. The graph to be edited G_0 is embedded in a labeled complete graph G_Ω referred to as the 'edit grid.' Vertices and edges in G_Ω may possess the special null label indicating the element is not part of the embedded graph; such an element is referred to as 'virtual' and allows for insertion and deletion edits by simply swapping a null label for a non-null label or vice versa. The state of the edit grid is altered by relabeling its elements until the graph G_1 surfaces somewhere on the grid. Assuming a cost metric on the set of labels (including the null label), we prove the existence of a set of graph edits with minimum cost that occur in one transition of the edit grid state. We also show that the graph edit distance implied by this cost is a metric on the set of graphs.

Next, we consider the adjacency matrix representation in order to develop the binary linear programming formulation of the graph edit optimization as a practical implementation of the general framework. We show how to use this formulation to obtain upper and lower bounds on the graph edit distance in polynomial time.

Finally, we offer a minimum variance method for choosing a cost metric. This metric is appropriate for a graph recognition problem wherein an input graph is compared to a database of prototypes. It is based on the assumption that the prototype graphs should be roughly uniformly distributed in the metric space described by the graph edit distance.

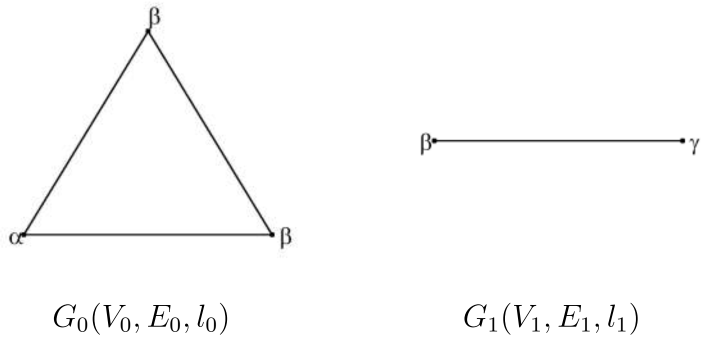


Fig. 1. Example undirected unweighted graphs with vertex attributes. The attribute alphabet is given by $\Sigma = \{\alpha, \beta, \gamma\}$.

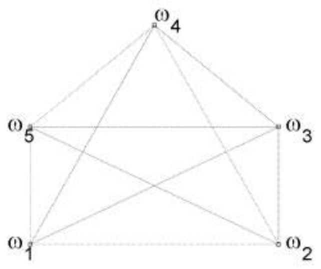


Fig. 2. Example edit grid $G_\Omega(\Omega, \Omega \times \Omega, l_\Omega)$ with five vertices.

A. Editing Graphs and the Graph Edit Distance

Let $G_0(V_0, E_0, l_0)$ be an undirected graph to be edited where V_0 is a finite set of vertices, $E_0 \subseteq V_0 \times V_0$ is a set of unweighted edges, and $l_0 : V_0 \rightarrow \Sigma$ is a labeling function that assigns a label from the alphabet Σ to each vertex. We assume there is at most one edge between any pair of vertices. The vertex labels in Σ capture the attribute information. We define the label $\phi \notin \Sigma$ as ϕ is a special vertex label whose purpose will be introduced shortly. These assumptions are made implicitly for every graph in this paper so that we need not mention them again. Some example graphs are shown in Figure 1.

Let $\Omega = \{\omega_i\}_{i=1}^N$ denote a set of vertices; accordingly $\Omega \times \Omega$ is the set of undirected edges connecting all pairs of vertices in Ω. We refer to the complete graph $G_\Omega(\Omega, \Omega \times \Omega, l_\Omega)$ as the *edit grid*. N, the number of vertices in the edit grid, may be as large as necessary. We will argue later that for computing the graph edit distance between $G_0(V_0, E_0, l_0)$ and $G_1(V_1, E_1, l_1)$ N need be no larger than $|V_0| + |V_1|$. An example edit grid with five vertices is shown in Figure 2.

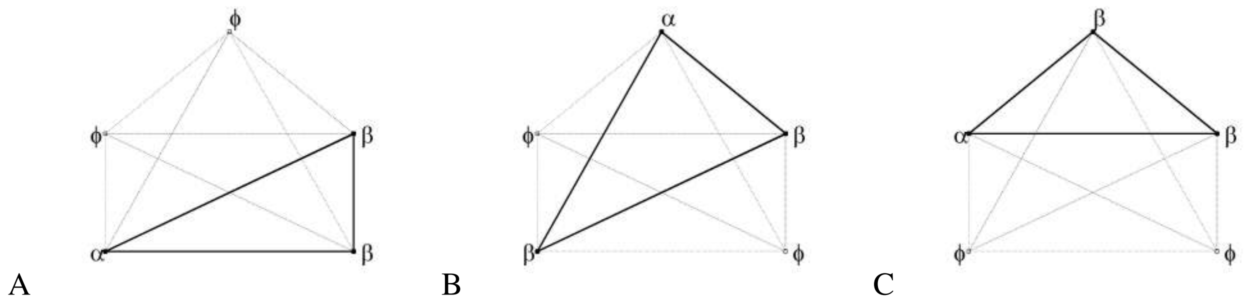


Fig. 3. Isomorphisms of the graph G_0 on the edit grid G_Ω. Vertex labels are noted; dotted lines indicate virtual edges (label 0) while solid lines indicate real edges (label 1). The vertex numbering in Figure 2 is used therefore the standard placement is shown in A.

For the purposes of editing, we let the graph $G_0(V_0, E_0, l_0)$ be situated on the edit grid, i.e. $V_0 \subset \Omega$ and $E_0 \subset \Omega \times \Omega$; equivalently, G_0 is a subgraph of G_Ω. Vertices in V_0 are assigned the appropriate label from Σ determined by the labeling function l_0, that is $l_\Omega(\omega_i) = l_0(v_i)$ for all $\omega_i \in V_0$. Vertices in $\Omega - V_0$ are assigned the vertex null label ϕ so $l_\Omega(\omega_i) = \phi$ for all $\omega_i \in \Omega - V_0$. The null label indicates a virtual vertex that may be made 'real' during editing by changing its label to something in Σ. Since edges are unweighted, they take labels from the set $\{0, 1\}$ where 1 indicates a real edge and 0 (the edge null label) indicates a virtual edge. Accordingly, $l_\Omega(\omega_i, \omega_j) = 1$ for all edges $(\omega_i, \omega_j) \in E_0$ and $l_\Omega(\omega_i, \omega_j) = 0$ for all edges $(\omega_i, \omega_j) \in (\Omega \times \Omega) - E_0$. When the graph G_0 is placed on the first $|V_0|$ vertices of G_Ω (i.e. $\omega_i = v_i$ for $i = 1, 2, \ldots, |V_0|$), we refer to this as the *standard placement*. Some placements of the graph G_0 from Figure 1 on the edit grid of Figure 2 are shown in Figure 3. These are clearly isomorphisms of G_0 on the edit grid.

Here it is appropriate to provide a quick note on indexing notation used throughout this paper. Superscript indices index elements within a vector while subscript indices index the entire vector (such as when it occurs in a sequence). For example, x_5^2 refers to the second element in the x_5 vector (fifth vector in a sequence of $\{x_k\}$). Similar indexing schemes are adopted for matrices: A_1^{34} refers to the $(3, 4)$ element in matrix A_1. Also, a single superscript index on a matrix indexes the entire row, so that A_1^3 denotes the third row of matrix A_1.

Let $\eta \in (\Sigma \cup \phi)^N \times \{0, 1\}^{\frac{1}{2}(N^2 - N)}$ denote the state vector of the edit grid. We assign an ordering to the graph elements (vertices and edges) of the edit grid so that the i^{th} element of η contains the label of the i^{th} element of the edit grid (i.e. $\eta^i = l_\Omega(\rho^i)$ for $\rho^i \in \Omega \cup (\Omega \times \Omega)$). For

i	ρ^i	η_A^i	η_B^i	η_C^i	π_A^i	π_B^i	π_C^i
1	ω_1	α	β	ϕ	1	4	5
2	ω_2	β	ϕ	ϕ	2	1	4
3	ω_3	β	β	β	3	3	3
4	ω_4	ϕ	α	β	4	5	2
5	ω_5	ϕ	ϕ	α	5	2	1
6	(ω_1, ω_2)	1	0	0	6	8	15
7	(ω_1, ω_3)	1	1	0	7	13	14
8	(ω_1, ω_4)	0	1	0	8	15	12
9	(ω_1, ω_5)	0	0	0	9	11	9
10	(ω_2, ω_3)	1	0	0	10	7	13
11	(ω_2, ω_4)	0	0	0	11	9	11
12	(ω_2, ω_5)	0	0	0	12	6	8
13	(ω_3, ω_4)	0	1	1	13	14	10
14	(ω_3, ω_5)	0	0	1	14	10	7
15	(ω_4, ω_5)	0	0	1	15	12	6

TABLE I

ELEMENT ORDERINGS, STATE VECTORS, AND CORRESPONDING STATE VECTOR PERMUTATIONS FOR THE ISOMORPHISMS OF G_0 ON THE EDIT GRID AS SHOWN IN FIGURE 3. THE VECTOR OF EDIT GRID GRAPH ELEMENTS IS DENOTED BY ρ, THE STATE VECTORS ARE DENOTED BY η_A, η_B, AND η_C, AND THE STATE VECTOR PERMUTATIONS ARE DENOTED BY π_A, π_B, AND π_C FOR THE CORRESPONDING ISOMORPHISM IN FIGURE 3. NOTE THAT THE NUMBERING OF THE EDIT GRID VERTICES ω_i SHOWN IN FIGURE 2 IS USED.

example, the element orderings and state vectors for the graphs in Figure 3 are shown in Table I.

We perform a finite sequence of graph edits to transform the graph $G_0(V_0, E_0, l_0)$ situated on the edit grid $G_\Omega(\Omega, \Omega \times \Omega, l_\Omega)$ into the graph $G_1(V_1, E_1, l_1)$ (such that $V_1 \subset \Omega$ and $E_1 \subset \Omega \times \Omega$). Vertex edits consist of insertion, deletion, or relabeling to some other symbol in Σ. Edge edits consist of insertion or deletion. Using the null labels introduced above, we may interpret all graph edits as relabeling of real and virtual elements. For example, changing the label of a virtual edge from 0 to 1 corresponds to the insertion of that edge into the graph $G(V, E, l)$. Similarly, relabeling an edge from 1 to 0 amounts to deleting that edge. Vertex insertion or deletion is a bit more complex in that it also typically involves edge edits; however there is a natural decomposition of the vertex edit that is consistent with this framework. Consider a vertex deletion whereby a vertex is removed from the graph along with all edges adjacent to that vertex. We may delete the vertex by changing its label $\sigma \in \Sigma$ to ϕ and relabeling all edges adjacent to it from 1 to 0. Vertex insertion may involve attaching the new vertex to the existing graph via an edge. Again this process is easily decomposed by relabeling a virtual vertex from ϕ to some desired label $\sigma \in \Sigma$ and changing the label of an appropriate virtual edge from 0 to 1. Thus it suffices to consider the transforming of edge and vertex labels as the fundamental operation for editing.

Edits essentially serve to alter the state of the edit grid. Thus we may specify a sequence of edits by noting the sequence of edit grid state vectors $\{\eta_k\}_{k=0}^{M}$ resulting from these edits. Suppose we wish to transform a graph G_0 into a graph G_1 by performing edit operations. Assume at this point that the initial state of the edit grid η_0 contains G_0 in its standard placement. We must have the final state η_M be such that it describes G_1 situated in some fashion on the edit grid. Thus if Γ_1 is the set of state vectors corresponding to all isomorphisms of G_1 on the edit grid, we must have $\eta_M \in \Gamma_1$. Two different state sequences for transforming the example G_0 into the example G_1 of Figure 1 are shown in Figure 4.

The set of all isomorphisms of a graph G_n on the edit grid, Γ_n, may be defined in terms of the standard placement of G_n denoted by η_n and Π–the set of all permutation mappings describing isomorphisms of the edit grid G_Ω–as in Eq. (1).

$$\Gamma_n = \left\{ \eta \mid \exists \pi \in \Pi \text{ s.t. } \eta^i = \eta_n^{\pi^i} \right\} \tag{1}$$

Note that Π does not contain all possible permutations of the elements of the state vector η because elements of Π must describe an isomorphism of the edit grid. For example, an edit grid with two vertices $\Omega = \{\omega_1, \omega_2\}$ has only two isomorphisms: $\omega_1' = \omega_1$, $\omega_2' = \omega_2$ and $\omega_1' = \omega_2$, $\omega_2' = \omega_1$. Assuming the graph elements are indexed as $\rho = (\omega_1, \omega_2, (\omega_1, \omega_2))$, there are only two permutations of the state vector that comprise Π: $\pi_1 = (1, 2, 3)$ and $\pi_2 = (2, 1, 3)$. Indeed, it will always be the case that $|\Pi| = N!$. The permutations of the state vector corresponding to the isomorphisms of G_0 in Figure 3 are given in Table I.

We define a cost function $c : (\Sigma \cup \phi)^2 \cup \{0, 1\}^2 \to \Re_+$ that assigns a nonnegative cost to each graph edit. The cost of an edit grid state transition denoted as $C(\eta_{k-1}, \eta_k)$ is simply the sum of

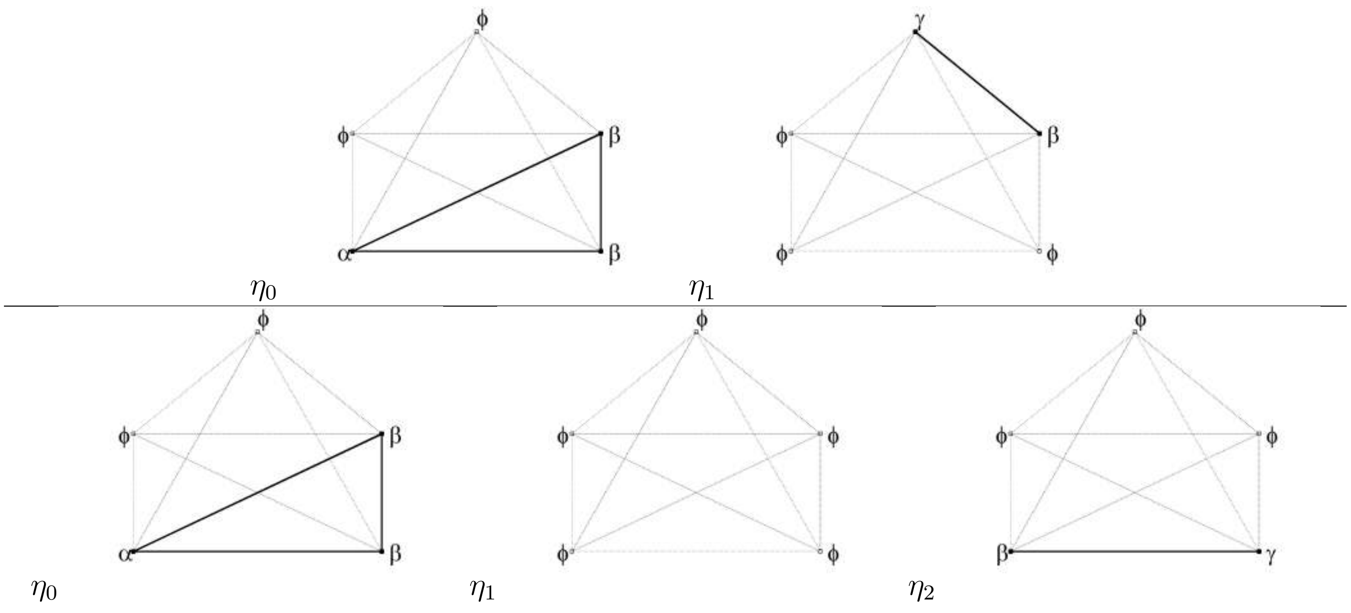


Fig. 4. Two different edit grid state sequences for transforming G_0 of Figure 1 into G_1. The upper sequence requires only one state transition, while the lower sequence requires two. In both cases, the initial state is the standard placement of G_0, and the final state represents an isomorphism of G_1 on the edit grid. Using the vertex numbering scheme of Fig. 2, the following non-trivial edits are made in the single transition of the upper sequence: $(\omega_1, \alpha \to \phi)$, $(\omega_2, \beta \to \phi)$, $(\omega_4, \phi \to \gamma)$, $((\omega_1, \omega_2), 1 \to 0)$, $((\omega_1, \omega_3), 1 \to 0)$, $((\omega_2, \omega_3), 1 \to 0)$, and $((\omega_3, \omega_4), 0 \to 1)$. In the first transition of the lower sequence we have $(\omega_1, \alpha \to \phi)$, $(\omega_2, \beta \to \phi)$, $(\omega_3, \beta \to \phi)$, $((\omega_1, \omega_2), 1 \to 0)$, $((\omega_1, \omega_3), 1 \to 0)$, $((\omega_2, \omega_3), 1 \to 0)$, and in the second transition of the lower sequence: $(\omega_1, \phi \to \beta)$, $(\omega_2, \phi \to \gamma)$, $((\omega_1, \omega_2), 0 \to 1)$. For a metric cost function c, the cost of the upper sequence is given by $c(\alpha, \phi) + c(\beta, \phi) + c(\phi, \gamma) + 4c(0, 1)$, and the cost of the lower sequence is $c(\alpha, \phi) + 3c(\beta, \phi) + c(\phi, \gamma) + 4c(0, 1)$.

all edits separating the two states, i.e.

$$C(\eta_{k-1}, \eta_k) = \sum_{i=1}^{I} c(\eta_{k-1}^i, \eta_k^i) \tag{2}$$

where $I = N + \frac{1}{2}(N^2 - N)$ is the total number of graph elements (vertices and edges) in the edit grid. Similarly, the cost of a sequence of state transitions is simply the sum of the costs of individual transitions. We consider only cost functions that are metrics on the set of vertex and edge labels as characterized by Definition 1.

Definition 1 *A cost function $c : (\Sigma \cup \phi)^2 \cup \{0, 1\}^2 \to \Re_+$ is a metric if the following conditions hold for all $(x, y) \in (\Sigma \cup \phi)^2 \cup \{0, 1\}^2$:*

1) *Positive definiteness: $c(x, y) = 0$ if and only if $x = y$.*
2) *Symmetry: $c(x, y) = c(y, x)$.*
3) *Triangle inequality: $c(x, y) \leq c(x, z) + c(z, y)$ for all (x, z) and (z, y) in $(\Sigma \cup \phi)^2 \cup \{0, 1\}^2$.*

Assuming a metric cost function, the cost of the upper sequence in Figure 4 is $c(\alpha,\phi)+c(\beta,\phi)+c(\phi,\gamma)+4c(0,1)$, and the cost of the lower sequence is $c(\alpha,\phi)+3c(\beta,\phi)+c(\phi,\gamma)+4c(0,1)$. With such a cost function c, we have the following simple result.

Proposition 1 *If $c : (\Sigma \cup \phi)^2 \cup \{0,1\}^2 \rightarrow \Re_+$ is a metric on the set of labels then C as defined in Eq. (2) is a metric on the edit grid state space.*

Proof: Expand C in terms of c using the definition in Eq. (2) and apply the metric properties of c to trivially obtain the corresponding properties for C. ■

We thus have the following useful Lemma:

Lemma 1 *Let c be a metric and $\{\eta_k\}_{k=0}^{M}$ be a sequence of edit grid state vectors. Then for all $M \geq 1$, $C(\eta_0,\eta_M) \leq \sum_{k=1}^{M} C(\eta_{k-1},\eta_k)$.*

Proof: The $M = 1$ case is trivial and the $M = 2$ case follows from the triangle inequality. Assume the claim holds for some value M and proceed by induction:

$$\begin{aligned} C(\eta_0,\eta_{M+1}) &\leq C(\eta_0,\eta_M) + C(\eta_M,\eta_{M+1}) \\ &\leq \textstyle\sum_{k=1}^{M} C(\eta_{k-1},\eta_k) + C(\eta_M,\eta_{M+1}) \\ &= \textstyle\sum_{k=1}^{M+1} C(\eta_{k-1},\eta_k) \end{aligned} \tag{3}$$

where the first line in Eq. (3) follows from the triangle inequality and the second line uses the induction hypothesis. ■

We now define the graph edit distance with respect to a cost function c as

$$d_c(G_0,G_1) = \min_{\{\eta_k\}_{k=1}^{M} | \eta_M \in \Gamma_1} \sum_{k=1}^{M} C(\eta_{k-1},\eta_k) \tag{4}$$

where η_0 is the standard placement of G_0 on the edit grid, Γ_1 is the set of state vectors corresponding to isomorphisms of G_1 on the edit grid as in Eq. (1), and M is the maximum number of allowed state transitions (this can be as large as desired, but we are only concerned with finite graphs implying M will be finite). There is no loss of generality by fixing the number of terms in the summation of Eq. (4), since for $M' < M$ transitions we can simply repeat the final state $\eta_{M'}$ so that $\eta_k = \eta_{M'}$ for $k = M'+1, M'+2, \ldots M$. Note that since there is a finite number of state vector sequences $\{\eta_k\}_{k=1}^M$ such that $\eta_M \in \Gamma_1$, the graph edit distance as defined in Eq. (4) always exists. It essentially finds a state transition sequence of minimum cost that transforms G_0 into G_1 (the minimizing sequence need not be unique). Since our cost function is a metric, it seems logical that we should be able to achieve the minimum cost with only one edit grid state transition. The following theorem shows this is indeed the case.

Theorem 1 *For a given graph edit cost function, $c : (\Sigma \cup \phi)^2 \cup \{0,1\}^2 \rightarrow \Re_+$ that is a metric, there exists a single state transition $(\eta_0, \bar{\eta}_1)$ such that $d_c(G_0, G_1) = C(\eta_0, \bar{\eta}_1)$ where η_0 is the standard placement of G_0 and $\bar{\eta}_1 \in \Gamma_1$.*

Proof: Assume the initial state η_0 describes G_0 in its standard placement on the edit grid, and suppose $\{\tilde{\eta}_k\}_{k=1}^M$ solves the graph edit minimization in Eq. (4)–this optimal sequence always exists as argued earlier. Define $\bar{\eta}_1 = \tilde{\eta}_M$, then we have

$$\begin{aligned} d_c(G_0, G_1) &= C(\eta_0, \tilde{\eta}_1) + \textstyle\sum_{k=2}^M C(\tilde{\eta}_{k-1}, \tilde{\eta}_k) \\ &\geq C(\eta_0, \tilde{\eta}_M) \\ &= C(\eta_0, \bar{\eta}_1) \end{aligned} \tag{5}$$

where the second line follows from the first by applying Lemma 1. Now define the state sequence $\{\hat{\eta}_k\}_{k=1}^M$ so that $\hat{\eta}_k = \bar{\eta}_1$ for all k. Then the positive definiteness of the metric C gives

$$\begin{aligned} C(\eta_0, \bar{\eta}_1) &= C(\eta_0, \hat{\eta}_1) + \textstyle\sum_{k=2}^M C(\hat{\eta}_{k-1}, \hat{\eta}_k) \\ &\geq \min_{\{\eta_k\}_{k=1}^M | \eta_M \in \Gamma_1} \textstyle\sum_{k=1}^M C(\eta_{k-1}, \eta_k) \\ &= d_c(G_0, G_1) \end{aligned} \tag{6}$$

The second line follows because $\hat{\eta}_M = \bar{\eta}_1 = \tilde{\eta}_M \in \Gamma_1$ and the proof is complete. ■

Theorem 1 allows the graph edit distance in Eq. (4) to be re-expressed equivalently as

$$d_c(G_0, G_1) = \min_{\bar{\eta}_1 \in \Gamma_1} C(\eta_0, \bar{\eta}_1) = \min_{\pi \in \Pi} \sum_{i=1}^{I} c(\eta_0^i, \eta_1^{\pi^i}) \tag{7}$$

where the second equality follows from the definition of Γ_1 in Eq. (1) and η_0 and η_1 are the standard placements of G_0 and G_1 respectively. Note that this one-state-transition result is crucial for our binary linear programming formulation, and it hinges on the metric properties of the cost function. Under a cost that is not a metric, there is no guarantee that the graph edit distance can be computed with just one edit grid state transition. One might consider multiple state transitions in a greedy algorithm for computing the graph edit distance in this case.

If π solves the minimization in Eq. (7), then we have

$$\begin{aligned} d_c(G_0, G_1) &= \textstyle\sum_{i=1}^{I} c(\eta_0^i, \eta_1^{\pi^i}) \\ &= \textstyle\sum_{i|\rho^i \in V_0 \cup V_1 \cup E_0 \cup E_1} c(\eta_0^i, \eta_1^{\pi^i}) \end{aligned} \tag{8}$$

Thus only elements of the edit grid comprising either G_0 or G_1 contribute to the sum; all other elements have the null label in both states and therefore have zero cost. The most terms contribute to the summation in Eq. (8) in the case where V_0 and V_1 are disjoint. This suggests we need an edit grid with no more than $N = |V_0| + |V_1|$ vertices in order to compute the graph edit distance in Eq. (7).

B. A Metric for Graphs

In addition to justifying the binary linear programming formulation to follow, Theorem 1 also provides a simple means for showing that the graph edit distance (when derived from a metric cost) is a metric itself on the set of undirected, unweighted graphs with vertex attributes (denoted by Ξ). We need a preliminary lemma however. We have assumed for simplicity that the graph edit minimization always starts with the graph G_0 in its standard placement on the edit grid. It seems that this should not be necessary; i.e. that the graph edit distance should be the same regardless of where G_0 is on the edit grid. The following lemma establishes this.

Lemma 2 *If $\bar{\eta}_0 \in \Gamma_0$ where Γ_0 is as defined in Eq. (1), then $d_c(G_0, G_1) = \min\limits_{\tilde{\eta}_1 \in \Gamma_1} C(\bar{\eta}_0, \tilde{\eta}_1) = \min\limits_{\pi \in \Pi} \sum_{i=1}^{I} c(\bar{\eta}_0^i, \eta_1^{\pi^i})$.*

Proof: Let $\bar{\eta}_0^i = \eta_0^{\bar{\pi}^i}$ for the standard placement η_0 then we have

$$\begin{aligned} d_c(G_0, G_1) &= \min_{\pi \in \Pi} \textstyle\sum_{i=1}^{I} c(\eta_0^i, \eta_1^{\pi^i}) \\ &= \min_{\pi \in \Pi} \textstyle\sum_{j=1}^{I} c(\eta_0^{\bar{\pi}^j}, \eta_1^{\pi^{\bar{\pi}^j}}) \\ &= \min_{\tilde{\pi} \in \Pi} \textstyle\sum_{j=1}^{I} c(\bar{\eta}_0^j, \eta_1^{\tilde{\pi}^j}) \end{aligned} \tag{9}$$

where the second line follows by reordering the sum with index change $i = \bar{\pi}^j$, and the third by noting that applying $\bar{\pi}$ to any permutation π in Π results in another permutation $\tilde{\pi}$ in Π. ■

We now prove that the graph edit distance is a metric.

Theorem 2 *If the cost function* $c : (\Sigma \cup \phi)^2 \cup \{0,1\}^2 \rightarrow \Re_+$ *is a metric, then the associated graph edit distance* $d_c : \Xi^2 \rightarrow \Re_+$ *is a metric.*

Proof: Let G_0, G_1, and G_2 all be graphs in Ξ.

1) Positive definiteness: Apply Theorem 1 to give $d_c(G_0, G_1) = C(\eta_0, \bar{\eta}_1)$. Clearly d_c is nonnegative because it is a sum of nonnegative edit costs. Now since C is a metric, $C(\eta_0, \bar{\eta}_1) = 0$ if and only if $\eta_0 = \bar{\eta}_1$. This occurs if and only if G_0 is isomorphic to G_1. In other words the standard placement of G_0, η_0, also describes an isomorphism of G_1 on the edit grid, i.e. $\eta_0 \in \Gamma_1$.
2) Symmetry: Theorem 1 gives

$$\begin{aligned} d_c(G_0, G_1) &= C(\eta_0, \bar{\eta}_1) \\ &= C(\bar{\eta}_1, \eta_0) \\ &\geq \min_{\tilde{\eta}_0 \in \Gamma_0} C(\bar{\eta}_1, \tilde{\eta}_0) \\ &= d_c(G_1, G_0) \end{aligned} \tag{10}$$

where symmetry of the metric C gives the second line, and Lemma 2 gives the fourth line from the third. The reverse inequality, $d_c(G_1, G_0) \geq d_c(G_0, G_1)$, is established via an identical argument.
3) Triangle Inequality: The symmetry property gives $d_c(G_0, G_2) = d_c(G_2, G_0)$. Theorem 1 gives $d_c(G_2, G_0) = C(\eta_2, \bar{\eta}_0)$ and $d_c(G_2, G_1) = C(\eta_2, \bar{\eta}_1)$ where η_2 is the standard placement of G_2. But by symmetry of C we have $C(\eta_2, \bar{\eta}_0) = C(\bar{\eta}_0, \eta_2)$ therefore

$$\begin{aligned} d_c(G_0, G_2) + d_c(G_2, G_1) &= C(\bar{\eta}_0, \eta_2) + C(\eta_2, \bar{\eta}_1) \\ &\geq C(\bar{\eta}_0, \bar{\eta}_1) \\ &\geq \min_{\tilde{\eta}_1 \in \Gamma_1} C(\bar{\eta}_0, \tilde{\eta}_1) \\ &= d_c(G_0, G_1) \end{aligned} \tag{11}$$

where the second line follows from the triangle inequality for C, and Lemma 2 gives the fourth line from the third.

■

C. Binary Linear Program for the Graph Edit Distance

In order to develop a binary linear program for computing the graph edit distance, we organize the labels in the edit grid state vector using the adjacency matrix representation. The elements of the adjacency matrix consist of edge labels, and we associate a vertex label with each row(column) of the matrix (the matrix is symmetric since the graphs of interest are undirected). We adopt the ordering scheme in Table I, so that if $A_k \in \{0,1\}^{N\times N}$ is the adjacency matrix corresponding to edit grid state vector η_k then the vertex label η_k^i is associated with the i^{th} row(column) of A_k for $i = 1, 2, \ldots, N$ (i.e. $l(A_k^i) = \eta_k^i$), and the upper half of the matrix is given by $A_k^{ij} = \eta_k^{iN+j-\frac{i^2+i}{2}}$ for $i < j \leq N$–the lower half follows similarly from symmetry. All zeros lie on the diagonal of A_k. For example, the adjacency matrix representations of the isomorphisms in Figure 3 (with state vectors in Table (I) are given by

$$A_A = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{matrix} \alpha \\ \beta \\ \beta \\ \phi \\ \phi \end{matrix} \quad A_B = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{matrix} \beta \\ \phi \\ \beta \\ \alpha \\ \phi \end{matrix} \quad A_C = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix} \begin{matrix} \phi \\ \phi \\ \beta \\ \beta \\ \alpha \end{matrix} \tag{12}$$

where the vertex labels associated with each row/column are printed to the right of the corresponding row.

The adjacency matrix representation also allows a convenient means for expressing an isomorphism permutation $\pi \in \Pi$. We can represent the first N elements of π as a permutation matrix $P \in \{0,1\}^{N\times N}$–recall that the remaining $\frac{1}{2}(N^2 - N)$ elements of π correspond to reordering of the edges which is determined by the vertex permutation in the first N elements. The elements of the permutation matrix P corresponding to π are given by $P^{ij} = \delta(\pi^i, j)$ for $i, j = 1, 2, \ldots, N$

where $\delta : \Re^2 \to \{0,1\}$ is the Kronecker delta function.

We now write the graph edit distance in this framework. First, partition the re-expression of Eq. (7) into summations over state vector entries corresponding to vertex and edge elements (assuming an ordering like the one in Table I is used).

$$\begin{aligned} d_c(G_0,G_1) &= \min_{\pi\in\Pi} \textstyle\sum_{i=1}^{N} c(\eta_0^i,\eta_1^{\pi^i}) + \sum_{i=N+1}^{I} c(\eta_0^i,\eta_1^{\pi^i}) \\ &= \min_{\pi\in\Pi} \textstyle\sum_{i=1}^{N}\sum_{j=1}^{N} c(\eta_0^i,\eta_1^j)\delta(\pi^i,j) + c(0,1)\sum_{i=N+1}^{I}(1-\delta(\eta_0^i,\eta_1^{\pi^i})) \end{aligned} \tag{13}$$

where the second term in the second line follows because c is a metric.

Let A_n be the adjacency matrix corresponding to η_n (the standard placement vector of G_n), $l(A_n^i)$ be the label assigned to the i^{th} row/column of A_n as previously described, and B be the set of all permutation matrices on $\Re^{N\times N}$ given by

$$B = \left\{ X \in \{0,1\}^{N\times N} \mid \sum_j X^{kj} = \sum_i X^{ik} = 1 \;\forall k \right\} \tag{14}$$

Eq. (13) may then be rewritten as

$$d_c(G_0,G_1) = \min_{P\in B} \sum_{i=1}^{N}\sum_{j=1}^{N} c\left(l(A_0^i), l(A_1^j)\right) P^{ij} + \frac{1}{2}c(0,1)\left|A_0 - PA_1P^T\right|^{ij} \tag{15}$$

Note that since A_n corresponds to the standard placement, only the first $|V_n|$ rows/columns will have non-ϕ labels, and only the upper $|V_n|\times|V_n|$ block of A_n will have nonzero elements. Also, following the prior argument, we use $N = |V_0| + |V_1|$. In order to make the optimization in Eq. (15) linear, we follow the strategy in [20] by introducing the matrices $\tilde{S}$, $\tilde{T}$. The graph edit distance $d_c(G_0,G_1)$ is then the optimal value of the following problem.

$$\begin{aligned} \min_{P,\tilde{S},\tilde{T}\in\{0,1\}^{N\times N}} \quad & \textstyle\sum_{i=1}^{N}\sum_{j=1}^{N} c\left(l(A_0^i), l(A_1^j)\right) P^{ij} + \frac{1}{2}c(0,1)\left(\tilde{S}P + \tilde{T}P\right)^{ij} \\ \text{such that} \quad & (A_0 - PA_1P^T + \tilde{S} - \tilde{T})^{ij} = 0 \;\forall i,j \\ & \textstyle\sum_i P^{ik} = \sum_j P^{kj} = 1 \;\forall k \end{aligned} \tag{16}$$

where we introduce an extra P in the second term of the objective function, which does not affect the result because it simply reorders the terms in the sum. Finally, we make the change of variables $S = \tilde{S}P$, $T = \tilde{T}P$ and right multiply the constraint equation by P to obtain the following binary linear program (BLP) for $d_c(G_0, G_1)$.

$$\begin{array}{lll}\underset{P,S,T\in\{0,1\}^{N\times N}}{\min} & \sum_{i=1}^{N}\sum_{j=1}^{N} c\left(l(A_0^i), l(A_1^j)\right) P^{ij} + \frac{1}{2}c(0,1)\,(S+T)^{ij} & \\ \text{such that} & (A_0P - PA_1 + S - T)^{ij} = 0\ \forall i,j & (17) \\ & \sum_i P^{ik} = \sum_j P^{kj} = 1\ \forall k & \end{array}$$

Note that Eq. (17) an equivalent representation of the GED minimization, so that Theorem 1 assures the existence of an optimal solution. More explicitly, feasibility of this BLP is seen by taking P as the identity matrix, S as the nonnegative part of $A_1 - A_0$, and T as the nonnegative part of $A_0 - A_1$. One might compare Eq. (17) to the linear programming approach for graph matching in [20]. Although [20] seeks to minimize the difference in adjacency matrix norms for graphs with the same number of vertices, this is a sort of generalization for the graph edit distance on attributed graphs. The optimal permutation matrix that solves Eq. (17), P_*, can be used to determine the optimal edit operations whose cost is the graph edit distance as follows: simply form the permuted adjacency matrix $\bar{A}_1 = P_* A_1 P_*^T$ remembering to also permute row/column labels, then compare the row/column labels of A_0 to those of $\bar{A}_1$ to determine the optimal vertex relabelings, similarly compare elements of A_0 and $\bar{A}_1$ to determine optimal edge relabelings.

D. Bounding the Graph Edit Distance in Polynomial Time

Unfortunately, binary linear programming in general is NP-Hard [40], so for large problems the graph edit distance as given by Eq. (17) may be too hard to compute. However, we can obtain upper $u_{d_c}(G_0, G_1)$ and lower $l_{d_c}(G_0, G_1)$ bounds for the graph edit distance $d_c(G_0, G_1)$ in polynomial time. The lower bound is obtained by relaxing the constraints $P, S, T \in \{0,1\}^{N\times N}$ on the variables in Eq. (17) to $P, S, T \in [0,1]^{N\times N}$. This results in the linear programming relaxation given in Eq. (18).

$$\begin{array}{ll}\underset{P,S,T}{\min} & \sum_{i=1}^{N}\sum_{j=1}^{N} c\left(l(A_0^i), l(A_1^j)\right) P^{ij} + \frac{1}{2}c(0,1)\,(S+T)^{ij} \\ \text{s.t.} & (A_0P - PA_1 + S - T)^{ij} = 0\ \forall i,j \\ & \sum_i P^{ik} = \sum_j P^{kj} = 1\ \forall k \\ & 0 \le P^{ij} \le 1,\ \ 0 \le S^{ij} \le 1,\ \ 0 \le T^{ij} \le 1\ \forall i,j \end{array} \qquad (18)$$

For n variables, a linear program can be solved in $O(n^{3.5})$ time using an interior point method [43]. Thus the lower bound can be computed in $O(N^7)$ time since the linear program in Eq. (18) has $O(N^2)$ variables. If $l_{d_c}(G_0, G_1)$ is the optimal value of Eq. (18) then $l_{d_c}(G_0, G_1) \leq d_c(G_0, G_1)$ because the feasible region of the problem in Eq. (17) is a subset of the feasible region of the problem in Eq. (18). It follows that Eq. (18) is always feasible because Eq. (17) is always feasible as argued earlier. Thus the Weierstrass theorem assures existence of an optimal value since we are minimizing a linear functional over a nonempty compact set [44]. Notice that since the optimal matrix P_* that solves Eq. (18) is only guaranteed to be doubly stochastic, not necessarily a permutation matrix, there may not be a set of edit operations that achieves the lower bound $l_{d_c}(G_0, G_1)$. However in the event that P_* is a permutation matrix, such a set can be constructed as described in the previous section, and the optimal value of Eq. (18) is in fact the graph edit distance.

The upper bound is obtained in polynomial time by solving the assignment problem with only the vertex edit term. The assignment problem is given by

$$\begin{array}{ll} \displaystyle\min_{P \in \{0,1\}^{N \times N}} & \sum_{i=1}^{N} \sum_{j=1}^{N} c\left(l(A_0^i), l(A_1^j)\right) P^{ij} \\ \text{such that} & \sum_i P^{ik} = \sum_j P^{kj} = 1 \ \forall k \end{array} \tag{19}$$

Note that the optimal value of Eq. (19) always exists since there is a finite number of permutations and the identity always serves as a feasible permutation. Indeed, the Hungarian method may be used to solve it in $O(N^3)$ time [40]. If P_* is an optimal solution of Eq. (19), we compute S_* as the nonnegative part of $P_* A_1 - A_0 P_*$ and T_* as the nonnegative part of $A_0 P_* - P_* A_1$ so that (P_*, S_*, T_*) is in the feasible region of the problem in Eq. (17). $u_{d_c}(G_0, G_1)$ is then computed by evaluating the objective function in Eq. (17) at (P_*, S_*, T_*). It follows that $d_c(G_0, G_1) \leq u_{d_c}(G_0, G_1)$. Since P_* is a permutation matrix for the solution to Eq. (19), a set of edit operations whose cost is the upper bound to the graph edit distance can always be determined.

E. Selecting a Cost Metric for Uniform Distribution

We have assumed that the cost metric that characterizes the graph edit distance is available, however, in a given application it may not be clear what is the 'best' cost metric to use. We propose an empirical method for selecting a metric based on prior information suitable for a recognition problem. Suppose there is a set of prototype graphs $\{G_i\}_{i=1}^N$, and we classify a sample graph G_0 by selecting the prototype that is closest to it with respect to a graph distance metric. Prior information might suggest that the prototypes should be roughly uniformly distributed in the metric space of graphs defined by the graph edit distance. We can then choose an optimal metric with respect to this objective. Such a criterion will also have the effect of minimizing the worst case classification error, since it equalizes the probability of error under the minimum distance classifier.

Note that for a set of points uniformly distributed in some space, all nearest neighbor distances are the same. To uniformly distribute the prototypes, we first determine all pairwise distances using a cost metric that assigns unity to all edits, i.e. $c(0,1) = 1$, $c\left(l(A_0^i), l(A_1^j)\right) = 0$ if $l(A_0^i) = l(A_1^j)$ and $c\left(l(A_0^i), l(A_1^j)\right) = 1$ otherwise. The resulting edit operations under the unit cost matching are then fixed, and we optimize the normalized variance of pairwise distances over the set of cost metrics.

To carry out this optimization, we must tabulate the edits necessary to match each graph with its q nearest neighbors under a unit cost function. If the graphs are not too large, we may compute a permutation matrix that actually solves the graph edit distance minimization in Eq. (17). If this is not the case a permutation matrix that solves the assignment problem in Eq. (19) with unit weights will serve as a reasonable approximation. We consider each nearest neighbor pair only once. For example if G_i has G_j as one of its q nearest neighbors and G_j has G_i as one of its q nearest neighbors, then the edits necessary to match G_i to G_j are tabulated only once.

After determining the unit cost matching between all prototype pairs, we order all distinct edits that occur in any prototype matching and tabulate the vectors $\{H_j\}_{j=1}^K$. H_j^i indicates the number of times the i^{th} edit occurs to match the j^{th} pair of nearest neighbor prototypes (under a unit cost function) and K is the number of distinct nearest neighbor pairs. If c is a vector containing the corresponding edit costs, then the graph edit distance between the j^{th} pair is given by $d_c(G_{j0}, G_{j1}) = H_j^T c$. For example, consider as prototypes the standard placements of G_0 and G_1 as shown in the lower left and lower right respectively of Figure 4. If we order the edits as

$(\{\alpha,\beta\},\{\alpha,\gamma\},\{\alpha,\phi\},\{\beta,\gamma\},\{\beta,\phi\},\{\gamma,\phi\},\{1,0\})$ then the vector of counts H corresponding to this matching is $(1,0,0,1,1,0,2)$. Indeed, the dimension of each H_j vector will always be $\frac{1}{2}(|\Sigma|^2+|\Sigma|)+1$ for a vertex label set Σ and edge label set $\{0,1\}$.

Using this notation, the scaled variance of pairwise distances is then given by

$$K\sigma_d^2 = c^T\left[\sum_{i=1}^{K}\left(H_i-\frac{1}{K}\sum_{j=1}^{K}H_j\right)\left(H_i-\frac{1}{K}\sum_{j=1}^{K}H_j\right)^T\right]c \equiv c^TQc \tag{20}$$

We also require the cost function to be a metric. Positive definiteness may be enforced by selecting some minimum positive cost for all edits $a>0$. Symmetry is enforced implicitly by binning symmetric edits together in the count vector H and assigning the same cost to both edits. Finally, we must include linear inequalities of the form $c^i+c^j-c^k\geq 0$ to assure the triangle inequality holds for all sets of three vertex labels. There are $\frac{1}{2}|\Sigma|(|\Sigma|^2-1)$ of these; we define the matrix F such that $Fc\geq 0$ summarizes the triangle inequalities. The variance should be normalized before optimizing so that the result does not depend on the scale of the costs (determined by a). If we normalize by the sum of the costs, a convex program results where any local optimum is also a global optimum [45]. The optimal costs are then given by the convex program:

$$\begin{aligned} \min_c \quad & \tfrac{c^TQc}{e^Tc} \\ \text{s.t.} \quad & Fc\geq 0 \\ & c^i\geq a \ \ \forall i \end{aligned} \tag{21}$$

where e is a vector of ones. Note that the problem is always feasible because if we take $c^i=a$ for all i, then all necessary triangle inequalities are satisfied. Furthermore, the choice of a is irrelevant, provided $a>0$; we only need that the costs c^i (all of which correspond to nontrivial edits) be uniformly bounded below away from zero so that a metric results. For any given a we may choose $a'=\kappa a$ for some $\kappa>0$, and the change of variables $c'=\kappa c$ results in the original optimization problem in Eq. (21). The following proposition establishes the convexity of the problem. A barrier method will solve the convex program in polynomial time [45].

Proposition 2 *The optimization problem in Eq. (21) is convex.*

Proof: All inequalities are linear, so we need only show that the objective function is convex. The objective function is defined over the positive orthant, which is a convex set, so the function is convex if and only if its Hessian is positive semidefinite [45]. Let $\Psi(c) = \frac{c^T Q c}{e^T c}$ be the objective function of the problem. The Hessian quadratic form with an arbitrary vector v may be factored as

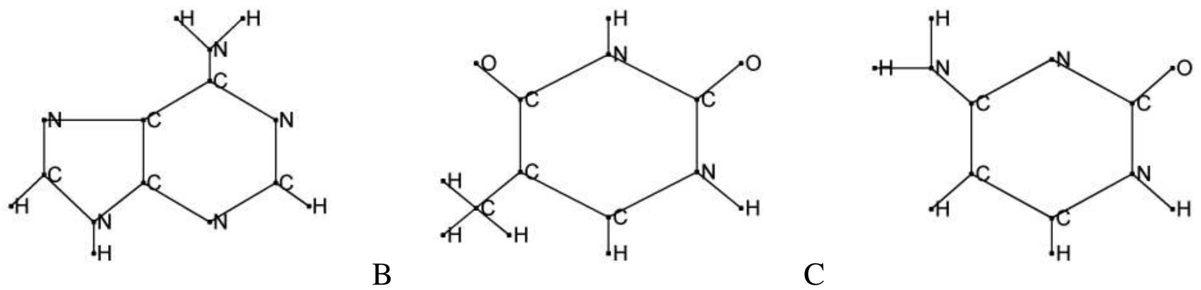
$$\begin{aligned} v^T(\nabla^2\Psi(c))v &= v^T \left(\tfrac{2}{(e^T c)^3} \left[(c^T Q c) e e^T + (e^T c)^2 Q - (e^T c)(e c^T Q + Q c e^T) \right] \right) v \\ &= \tfrac{2}{(e^T c)^3} \left\| (e^T v) Q^{\frac{1}{2}} c - (e^T c) Q^{\frac{1}{2}} v \right\|^2 \geq 0 \end{aligned} \quad (22)$$

where $Q^{\frac{1}{2}}$ is the matrix square root of Q which exists because Q as defined in Eq. (20) is obviously symmetric positive semidefinite. The inequality follows because c is defined over the positive orthant so $(e^T c)^3 > 0$; thus $\nabla^2\Psi(c) \succeq 0$. ■

III. Chemical Graph Recognition

As an application, we use the graph edit distance to recognize chemical graphs in the context of a chemical information system. We selected our database of 135 similar molecules from the Klotho Biochemical Compounds Declarative Database, which consists of small molecules useful in describing mechanisms of biochemical reactions [42]. Only molecules with 18 or fewer atoms were used so that we could compute exact distance measures in reasonable time. Attributed undirected graphs were generated from the 135 molecules (referred to as *chemical graphs*), then the optimal edit costs were computed to uniformly distribute them in the graph metric space. Finally, we compared the recognition ability of the graph edit distance with optimal costs and unit costs to that of two maximum common subgraph based distance metrics by using randomly perturbed prototype graphs from the database.

We first generated chemical graphs from the molecular structure diagrams by associating atoms with vertices and bonds with edges. Each vertex was labeled by the chemical symbol of the element to which that vertex corresponded. Our vertex label alphabet was thus given by $\Sigma = \{H, C, O, N, Cl, P, S, Br, Si\}$. Vertices in the graph were connected by an edge if and only if their corresponding atoms were bonded (single bonds, double bonds, etc. were treated equally). For example, the chemical graphs derived from the molecules adenine, thymine, and



A B C

Fig. 5. Chemical graphs derived from the familiar molecules from DNA: adenine (A), thymine (B), and cytosine (C).

cytosine are shown in Figure 5.

In order to compute the optimal edit costs, we treated all 135 molecules as nearest neighbors (so that the number of nearest neighbor pairs K is given by $\frac{1}{2}(135^2 - 135) = 9045$). Indeed, all molecules in the database are of similar structure and function. In the context of a larger chemical database consisting of thousands or millions of molecules, one might suppose our 135 molecules are the result of some clustering [46] or pre-screening procedure [7] performed using a quickly computed similarity measure in order to isolate only the most likely matches to a given input. For example, one might use the lower bound obtained by the LP relaxation in Eq. (18) as a pre-screening criterion. We then wish to homogenize the most likely matches with respect to the graph edit distance using the optimal edit costs.

We used the permutation matrices that solve the binary linear program in Eq. (17) with unit costs to tabulate the edit operation counts in the vectors $\{H_j\}$ necessary for optimizing the cost metric. The publicly available `lp_solve` program was used to solve the integer program; it implements the simplex method in a branch-and-bound algorithm [47]. The optimal edit costs were then computed by solving the convex program in Eq. (21) with $a = 0.1$ using a barrier method. The optimal edit costs for vertex relabelings are shown in Figure 6–the optimal edge edit cost was computed to be $c(0, 1) = 0.1$. The associated edit counts tabulated over all pairs in the database matched with unity cost function are shown in Figure 7. The most frequently inserted/deleted atom types in matching the prototypes were H, C, and O, while the most frequent relabelings were $O \leftrightarrow H$ and $O \leftrightarrow N$. Note that there is roughly an inverse relationship between the number of times a particular edit occurs and its optimal cost, as one might expect.

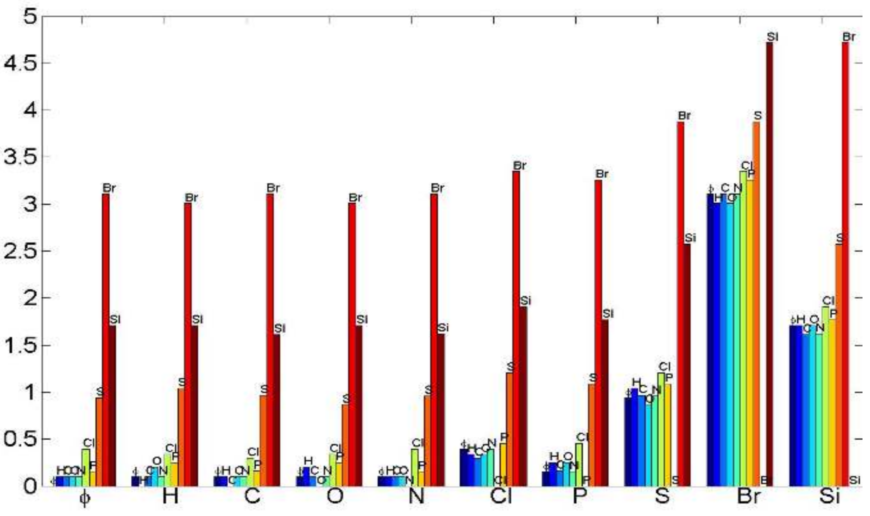


Fig. 6. Optimal edit costs resulting from the convex program in Eq. (21). There is a label associated with each group of bars. Within the group, the edit cost of changing the group label to an individual label corresponds to the height of the bar below that individual label. The optimal edge edit cost $c(0, 1)$ was 0.1.

This does not hold exactly, however, because the edit costs must also satisfy the necessary triangle inequalities.

The maximum common subgraph (MCS) is frequently used as a similarity measure for chemical graphs [8]. Also, some graph metrics have been devised based on the MCS that are appropriate for comparison to our graph edit based metric [13], [11], [12]. There are some variations in the literature on what is meant by 'maximum common subgraph.' The differences amount to whether the vertices or the edges are the defining feature of the subgraph, resulting in a 'maximum common induced subgraph (MCIS)' or a 'maximum common edge subgraph (MCES)' respectively [9]. The MCIS is used in [48], while the MCES is used in [49], [50]. We will use the MCIS, which satisfies the MCS definition given in [24]. A slightly modified version of the distance metric proposed in [13] appropriate for the MCIS is given by

$$d_{mcs1}(G_0, G_1) = |V_0| + |V_1| - 2|V_{01}| \tag{23}$$

where $G_{01}(V_{01}, E_{01}, l_{01})$ is the MCS (MCIS) of graphs $G_0(V_0, E_0, l_0)$ and $G_1(V_1, E_1, l_1)$. Note that we are being somewhat careless with language–although we say 'the' MCS, it need not be

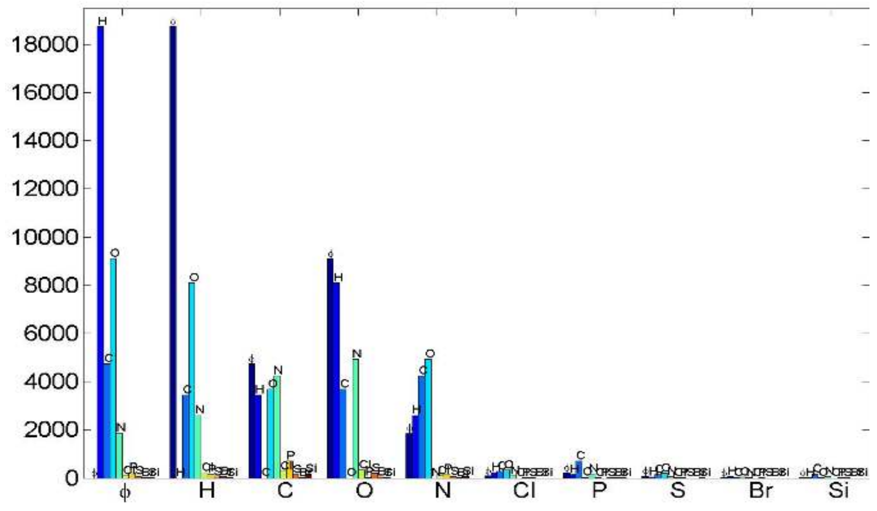


Fig. 7. Total number of occurrences of each type of vertex edit tabulated over all pairs of database graphs matched with unit cost function. There is a label associated with each group of bars. Within the group, the edit cost of changing the group label to an individual label corresponds to the height of the bar below that individual label. We see that H, C, and O are the most frequently inserted/deleted atom types, and the most frequent relabelings are $O \leftrightarrow H$ and $O \leftrightarrow N$. Note that edits occurring more frequently are typically assigned a lower cost (Figure 6). There were 60974 total edge edits (not shown).

unique. In addition to the MCS metric of Eq. (23), we also compared recognition performance to the following metric that is proposed in [11].

$$d_{mcs2}(G_0, G_1) = 1 - \frac{|V_{01}|}{\max(|V_0|, |V_1|)} \tag{24}$$

It has been shown that computing the MCS of graphs $G_0(V_0, E_0, l_0)$ and $G_1(V_1, E_1, l_1)$ is equivalent to computing the maximum clique in a modular product graph $G_p(V_p, E_p)$ [51]. In general finding the maximum clique is NP-Hard, so the worst case complexity is equivalent to binary linear programming [10]. The modular product graph is defined by the sets

$$\begin{aligned}
V_p &= \{(v_0, v_1) \mid v_0 \in V_0, v_1 \in V_1, l_0(v_0) = l_1(v_1)\} \\
E_+ &= \{[(v_0, v_1), (u_0, u_1)] \mid v_0 \neq u_0, v_1 \neq u_1, (v_0, u_0) \in E_0, (v_1, u_1) \in E_1\} \\
E_- &= \{[(v_0, v_1), (u_0, u_1)] \mid v_0 \neq u_0, v_1 \neq u_1, (v_0, u_0) \notin E_0, (v_1, u_1) \notin E_1\} \\
E_p &= E_+ \cup E_-
\end{aligned} \tag{25}$$

We computed the MCS by using the algorithm in [52] to find the maximum clique in the modular

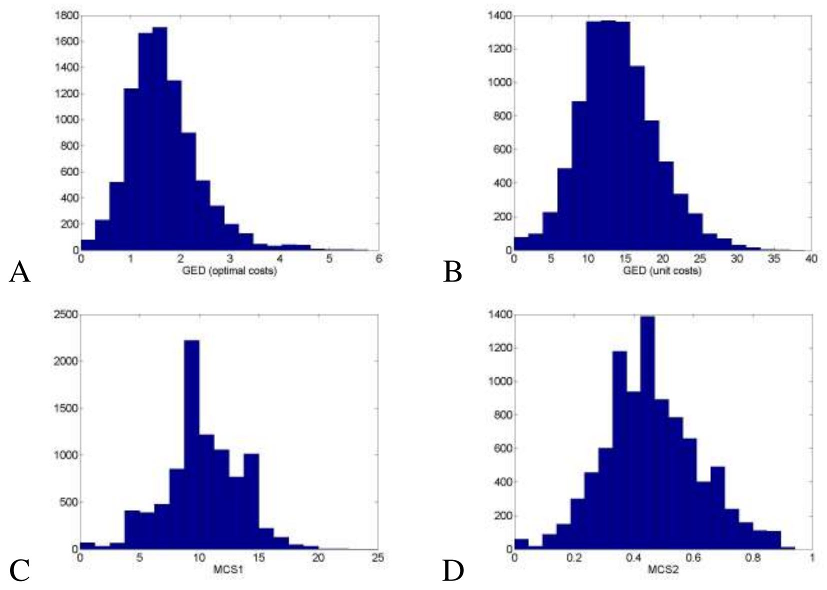


Fig. 8. Pairwise distance histograms between all 9045 pairs of 135 prototype graphs in the database. Distances computed with the GEDo are shown in A, those computed with the GEDu are in B, those computed with the MCS1 metric are shown in C, and those computed with the MCS2 metric are in D. Ideally, all pairwise distances would be the same. Since the GEDo distances are more concentrated, the GEDo more uniformly distributes the prototype graphs. This should result in less ambiguity in the graph recognition phase, whereby the distance between a sample graph and each prototype graph is computed.

product graph.

We calculated all 9045 pairwise distances between prototype graphs in the database using both the graph edit distance with optimal costs (GEDo) in Figure 6 and unit costs (GEDu), along with the two MCS distance metrics (MCS1 and MCS2) given in Eqs. (23) and (24) respectively. Histograms of the resulting pairwise distances are shown in Figure 8. Note that the GEDo pairwise distances are more concentrated around a single value than either of the MCS distances or the GEDu; this indicates the GEDo more uniformly distributes the prototypes in the graph metric space.

The ability of the four metrics to recognize input graphs as one of the prototype graphs in the database was tested next. An error-correcting graph isomorphism is indeed appropriate here, since each input graph was generated by applying a predetermined number of edits M (where $M \in \{1, 2, 3, 4, 5, 6\}$) to a randomly chosen prototype graph. The edits applied fell into one of the following five categories:

1) *edge edit*: M edges edits are selected with insertion and deletion having equal probability.

Once the M edit operations are selected, pairs of vertices between which edges should be either inserted or deleted are selected at random.

2) *vertex deletion*: M vertices are selected to be deleted. First a label to be deleted is chosen with deletion probabilities given by normalizing the edit counts over the ϕ-group in Figure 7. Among the vertices having the chosen label, one is selected at random to be deleted along with all edges connected to it.
3) *vertex insertion*: M vertices are inserted. First a label to be inserted is chosen with insertion probabilities given by normalizing the edit counts over the ϕ-group in Figure 7. A vertex with the chosen label is then connected by a single edge to an existing vertex in the graph chosen at random.
4) *vertex relabeling*: M vertices are selected to be relabeled. First a pair of labels is chosen with probabilities given by normalizing the edit counts in Figure 7 over all non-ϕ edits. Among the vertices having a label that matches one in the pair, one is selected at random and its label is changed to the complementary label in the pair.
5) *random*: The M edits to be performed are randomly chosen from the above four categories with each having equal probability.

Note that in performing vertex edits, we used the edit counts in Figure 7 as a guide so that the edits made would represent likely errors, say, in transcribing the chemical formula of one of the prototype graphs. Also, no regard was given to physical laws governing bonding, therefore some input graphs may not be physically realizable molecules. Examples of the different edit types applied to the adenine molecule are shown in Figure 9.

For each of the five edit categories, we generated ten input graphs from randomly chosen prototype graphs for each value of M (number of edits) ranging from 1 to 6; this resulted in $6 \times 10 \times 5 = 300$ sample input graphs. We then attempted to recognize the input graph by computing the distance (using GEDo, GEDu, MCS1, and MCS2) between the input graph and each of the 135 prototypes. There were $300 \times 135 = 40,500$ distinct input graph/prototype pairs matched using each of the four metrics to determine the corresponding graph distances. Due to the large number of matchings considered and the exponential complexity of the algorithms

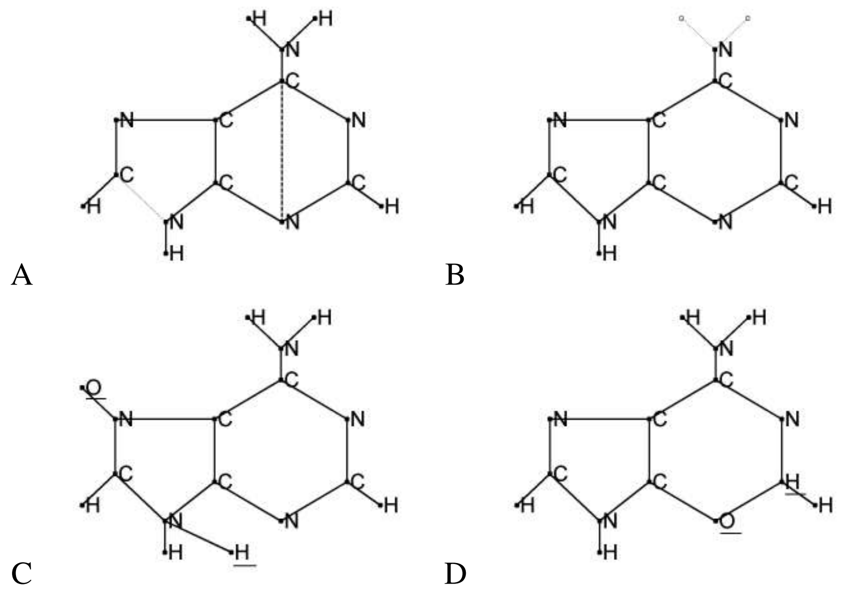


Fig. 9. Example edits applied to the adenine chemical graph shown in Figure 5A. Two edge edits (one insertion, one deletion) are shown in A with the thick dashed line representing the inserted edge and the thin dashed line is the deleted edge. Two vertex deletions (represented by dotted lines and open boxes) are shown in B. C shows two vertex insertions (underlined), and D shows two vertex relabelings (underlined).

tested, we allowed a maximum of 45 seconds for any distance computation. If an optimal solution was not found within the allotted time, the best feasible suboptimal solution available was used. Running on Pentium 4, 2GHz processors, the average time required to solve the binary linear program necessary for GEDo or GEDu with `lp_solve` [47] was about 1.3 seconds, while the average time required to compute the maximum common subgraph using the maximum clique algorithm of [52] was about 0.1 seconds. Although the MCS routine is about ten times faster here, these times will vary depending on the particular algorithm/implementation one chooses for binary linear programming and maximum common subgraph detection.

We say an input graph is correctly recognized if it is closest (with respect to the appropriate distance metric) to the prototype graph from which it was generated. A 'classifier ratio' (CR) as given in Eq. (26) was computed for each input graph in order to gauge the level of ambiguity associated with the classification.

$$CR = \frac{d_*}{d_o} \tag{26}$$

Where d_* is the graph edit distance between the sample graph and the prototype from which it was generated, and d_o is the distance between the sample and the nearest incorrect prototype

Metric	1) Edge Edit	2) Vertex Delete	3) Vertex Insert	4) Vertex Relabel	5) Random
GEDo	1.00 , 0.65	0.33 , 0.79	0.65 , 0.71	0.88 , 0.71	0.75 , 0.76
GEDu	0.98 , 0.70	0.25 , 0.85	0.45 , 0.87	0.95 , 0.69	0.65 , 0.80
MCS1	0.67 , 0.83	0.75 , 0.81	0.97 , 0.72	0.63 , 0.87	0.67 , 0.79
MCS2	0.78 , 0.77	0.52 , 0.89	0.55 , 0.87	0.73 , 0.82	0.68 , 0.85

TABLE II

PROPORTION OF GRAPHS CORRECTLY RECOGNIZED AND AVERAGE CLASSIFIER RATIO FOR EACH EDIT TYPE CATEGORY AVERAGED OVER ALL GRAPHS IN THAT CATEGORY. THESE ARE COMPUTED BY MARGINALIZING THE PLOTS IN FIGURE 10 OVER THE HORIZONTAL AXIS (NUMBER OF EDITS, M). THE FIRST NUMBER IN EACH PAIR IS THE PROPORTION CORRECTLY RECOGNIZED AND THE SECOND NUMBER IS THE AVERAGE CLASSIFIER RATIO (PC, CR). THE GED METRICS PERFORM BETTER IN THE CASE OF EDGE EDITS, VERTEX RELABELINGS, AND RANDOM EDITS (1, 4, AND 5); INDEED, THE GEDO METRIC CORRECTLY RECOGNIZES AT LEAST 75% OF GRAPHS IN THESE CATEGORIES. ONLY THE MCS1 METRIC PERFORMS WELL IN THE CASE OF VERTEX DELETIONS AND INSERTIONS (2 AND 3) WITH AT LEAST 75% CORRECT RECOGNITION IN BOTH CASES. THE GEDO METRIC (GED WITH OPTIMAL COSTS) HAS THE LOWEST AVERAGE CR IN ALL CATEGORIES BUT ONE, INDICATING REDUCED CLASSIFICATION AMBIGUITY.

('incorrect' in that the sample was not generated from this prototype). The lower CR is the less ambiguous the classification.

The proportion of graphs correctly recognized by each metric along with average classifier ratio associated with that metric for the five edit categories are shown in Figure 10. The classifier ratios were averaged only over those graphs that were correctly classified. The marginal values associated with these distributions averaged over the number of edits M are given in Table II. Note that the GED metrics had superior performance in the edge edit, vertex relabeling, and random edit categories; the GEDo metric correctly recognizes at least 75% of graphs in these categories. The GED metrics were particularly successful in the edge edit category with all graphs correctly recognized by the GEDo metric, which also gave a consistently lower classifier ratio. The MCS1 metric was most robust in the case of vertex deletions and insertions (having at least 75% correct recognition); indeed both GED metrics had significant trouble when three or more vertices are deleted and trail off similarly in the case of vertex insertions. Undoubtedly, the changes on the prototype graph caused by inserting/deleting three or more vertices were so drastic that a different prototype was actually closer with respect to the GED to the sample graph produced. The GED metrics remained strong for up to five vertex relabelings, however, while the proportion correct for either MCS metric in this case decreased after three. In Table

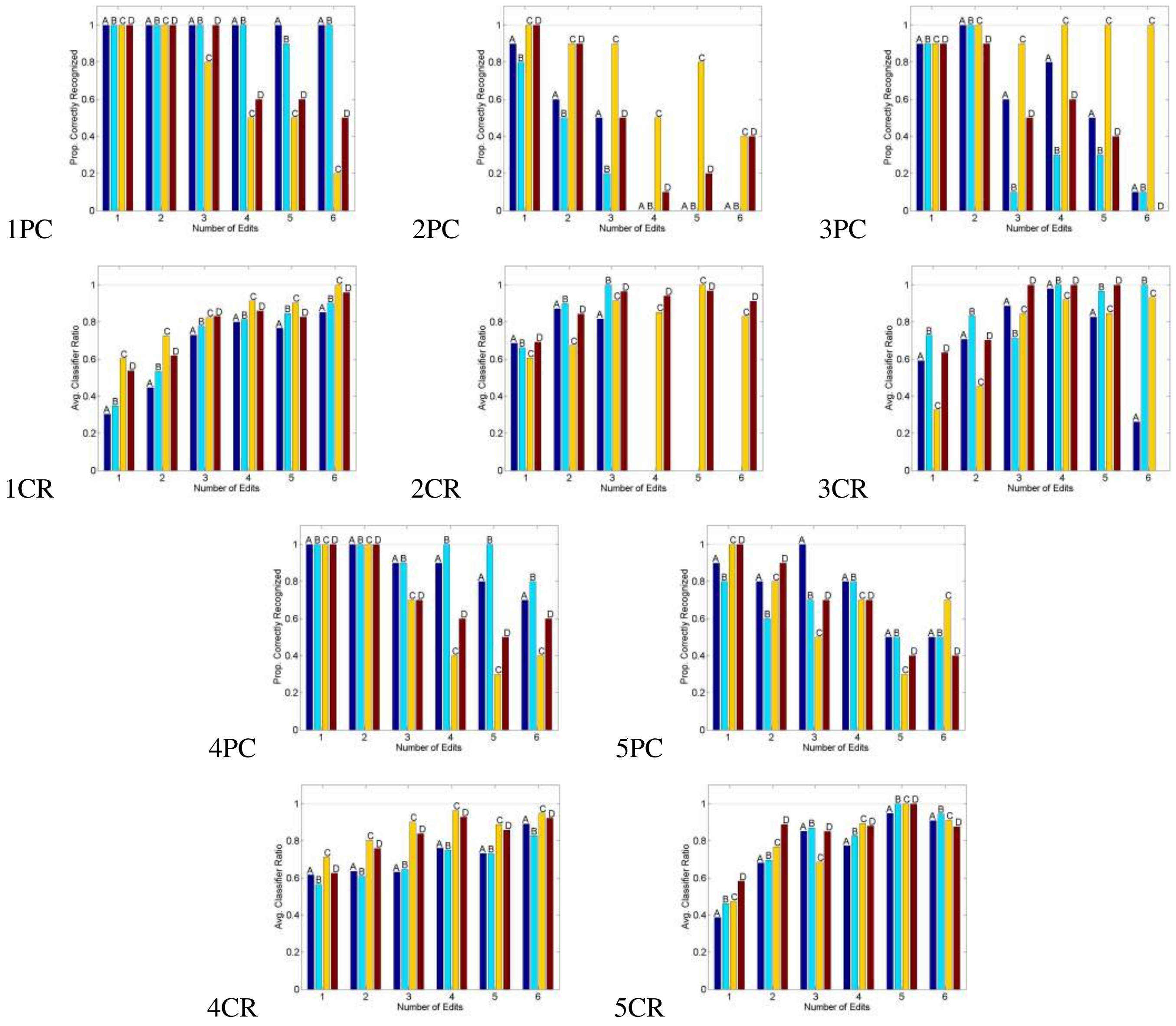


Fig. 10. Proportion of graphs correctly recognized (PC) and average classifier ratios (CR) for the five different edit categories: 1) edge edit, 2) vertex deletion, 3) vertex insertion, 4) vertex relabeling, and 5) random. Within each plot, the letter above a bar denotes the metric used: A) GEDo (graph edit distance with optimal costs), B) GEDu (graph edit distance with unit costs), C) MCS1 (max common subgraph metric of Eq. (23)), and D) MCS2 (max common subgraph metric of Eq. (24)). Each set of four bars corresponds to a different number of edits M, indicated along the horizontal axis. Typically, as the number of edits increases, the proportion correctly recognized drops while the ambiguity of classification (as measured by the CR) rises. Note that the GED metrics perform better in the case of edge edits, vertex relabelings, and random edits (1, 4, and 5). The MCS metrics perform better in the case of vertex deletions and insertions (2 and 3). Marginal values of these distributions (averaged over M) are given in Table II.

II, we see that the optimal costs were indeed effective in reducing classification ambiguity as measured by the CR since the GEDo metric has the lowest average CR in all categories but one.

IV. Conclusion

This paper develops a linear formulation of the graph edit distance for attributed graphs. We prove that the derived GED is a metric and show how to compute it using a binary linear program. Upper and lower bounds for the GED that can be computed in polynomial time are also given. A chemical graph recognition problem is presented as an application of the graph matching formalism. The edit costs are chosen using a normalized minimum variance criterion based on the prior information that the database graphs should be uniformly distributed in the graph metric space defined by the GED. This method is shown to give a metric that more uniformly distributes a database of 135 chemical graphs with similar structure than comparable maximum common subgraph based metrics. In recognizing chemical graphs generated by perturbing graphs in the database, the GED metrics with optimal costs and unit costs are shown to correctly recognize which prototype was perturbed more often than the MCS metrics in the case of edge edits and vertex relabelings. The MCS metrics perform better in the case of vertex insertions and deletions. When random edits are applied, the GED metrics are generally the best. Also, the GED with optimized edit costs is shown to have its intended effect of reducing the level of ambiguity associated with the chemical graph recognitions.

Unfortunately, the complexity of binary linear programming makes computing the GED between large graphs difficult using this method. However, the polynomial-time upper and lower bounds may be readily employed in this case. Also, these could be used in pre-screening on large chemical databases. For example, pre-screening may be done by rejecting all molecules whose LP lower bound to the query exceeds a given value. Although we have developed a metric for unweighted graphs, it can be directly extended to graphs with edge weights provided the cost of editing these edges is proportional to the absolute difference in the weights with positive proportionality constant k. Indeed, one could proceed from Eq. (17) with weighted adjacency matrices A_0, A_1 used instead and $c(0,1)$ replaced by k. However, Eq. (17) would become a mixed integer program since, depending on the weights, S and T may not be binary matrices. Incorporating edge weights would yield a method applicable to 3-D structure searching of chemical graphs where weights are assigned to the graph edges based on the length of the bond they represent [6], along with other applications of weighted graphs. We anticipate the results of this paper are applicable in any setting where it is necessary to compare graphical models.

Acknowledgements

This work was partially supported by a Dept. of EECS Graduate Fellowship to the first author and by the National Science Foundation under ITR contract CCR-0325571.

References

[1] T. Pavlidis, *Structural Pattern Recognition*. New York: Springer-Verlag, 1977.

[2] L. Jianzhuang and L. Tsui, "Graph-based method for face identification from a single 2d line drawing," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 10, pp. 1106–1119, 2000.

[3] J. Llados, E. Marti, and J. Villanueva, "Symbol recognition by error-tolerant subgraph matching between region adjacency graphs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 10, pp. 1137–1143, 2001.

[4] D. Shasha, J. Wang, and R. Giugno, "Algorithmics and applications of tree and graph searching," in *Proc. 21st ACM SIGMOD-SIGACT-SIGART*, Madison, WI, June 2005.

[5] M. Johnson and G. Maggiora, Eds., *Concepts and Applications of Molecular Similarity*. New York: John Wiley and Sons, 1990.

[6] G. Downs and P. Willett, "Similarity searching in databases of chemical structures," in *Reviews in Computational Chemistry, Vol. 7*, K. Lipkowitz and D. Boyd, Eds. New York: VCH Publishers, Inc., 1996, pp. 1–66.

[7] J. Raymond and P. Willett, "Effectiveness of graph-based and fingerprint-based similarity measures for virtual screening of 2d chemical structure databases," *J. Computer-Aided Molecular Design*, vol. 16, pp. 59–71, 2002.

[8] P. Willett, "Matching of chemical and biological structures using subgraph and maximal common subgraph isomorphism algorithms," *IMA Vol. Math. Appl.*, vol. 108, pp. 11–38, 1999.

[9] J. Raymond and P. Willett, "Maximum common subgraph isomorphism algorithms for the matching of chemical structures," *J. Computer-Aided Molecular Design*, vol. 16, pp. 521–533, 2002.

[10] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA: W.H. Freeman, 1979.

[11] H. Bunke and K. Shearer, "A graph distance metric based on the maximal common subgraph," *Pattern Recognition Letters*, vol. 19, pp. 255–259, 1998.

[12] W. Wallis, P. Shoubridge, M. Kraetz, and D. Ray, "Graph distances using graph union," *Pattern Recognition Letters*, vol. 22, pp. 701–704, 2001.

[13] M. Johnson, M. Naim, V. Nicholson, and C. Tsai, "Unique mathematical features of the substructure metric approach to quantitative molecular similarity analysis," in *Graph Theory and Topology in Chemistry*, R. King and D. Rouvray, Eds., Mar. 1987, pp. 219–225.

[14] M.-L. Fernandez and G. Valiente, "A graph distance metric combining maximum common subgraph and minimum common supergraph," *Pattern Recognition Letters*, vol. 22, pp. 753–758, 2001.

[15] A. Torsello, D. Hidovic-Rowe, and M. Pelillo, "Polynomial-time metrics for attributed trees," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 7, pp. 1087–1099, July 2005.

[16] M. Gori, M. Maggini, and L. Sarti, "Exact and approximate graph matching using random walks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 7, pp. 1100–1111, July 2005.

[17] B. McKay, "Practical graph isomorphism," *Congressus Numerantium*, vol. 30, pp. 45–87, 1981.

[18] L. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub)graph isomorphism algorithm for matching large graphs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 10, pp. 1367–1372, Oct. 2004.

[19] W. Tsai and K. Fu, "Error-correcting isomorphisms of attributed relational graphs for pattern recognition," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, pp. 757–768, 1979.

[20] H. Almohamad and S. Duffuaa, "A linear programming approach for the weighted graph matching problem," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 5, pp. 522–525, May 1993.

[21] S. Umeyama, "An eigendecomposition approach to weighted graph matching problems," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 10, no. 5, pp. 695–703, Sept. 1988.

[22] S. Gold and A. Rangarajan, "A graduated assignment algorithm for graph matching," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, no. 4, pp. 377–387, Apr. 1996.

[23] B. van Wyk and M. van Wyk, "A pocs-based graph matching algorithm," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 11, pp. 1526–1530, Nov. 2004.

[24] H. Bunke, "Error correcting graph matching: on the influence of the underlying cost function," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 9, pp. 917–922, Sept. 1999.

[25] ——, "Recent developments in graph matching," *Proc. 15th Intl. Conf. on Pattern Recognition*, vol. 2, pp. 117–124, Sept. 2000.

[26] R. Wagner and M. Fischer, "The string-to-string correction problem," *Journal of the Association for Computing Machinery*, vol. 21, no. 1, pp. 168–173, 1974.

[27] A. Hlaoui and S. Wang, "A new algorithm for inexact graph matching," *Proc. 16th Intl. Conf. on Pattern Recognition*, vol. 4, pp. 180–183, 2002.

[28] B. Messmer and H. Bunke, "Error-correcting graph isomorphism using decision trees," *Int. Journal of Pattern Recognition and Art. Intelligence*, vol. 12, pp. 721–742, 1998.

[29] R. Myers, R. Wilson, and E. Hancock, "Bayesian graph edit distance," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 6, pp. 628–635, June 2000.

[30] A. Robles-Kelly and E. Hancock, "Graph edit distance from spectral seriation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 3, pp. 365–378, Mar. 2005.

[31] P. Bergamini, L. Cinque, A. Cross, and E. Hancock, "Efficient alignment and correspondence using edit distance," *SSPR/SPR*, pp. 246–255, 2000.

[32] K. Zhang, "A constrained edit distance between unordered labeled trees," *Algorithmica*, vol. 15, no. 6, pp. 205–222, 1996.

[33] Z. Wang and K. Zhang, "Alignment between two rna structures," *MFCS*, pp. 690–702, 2001.

[34] P. Klein, S. Tirthapura, D. Sharvit, and B. Kimia, "A tree-edit distance algorithm for comparing simple, closed shapes," *Proc. ACM-SIAM Symp. Disc. Algorithms*, pp. 696–704, 2000.

[35] M. Pavel, *Fundamentals of Pattern Recognition*. New York: Marcel Dekker, 1989.

[36] M. Neuhaus and H. Bunke, "A probabilistic approach to learning costs for graph edit distance," *Proc. 17th Intl. Conf. on Pattern Recognition*, vol. 3, pp. 389–393, 2004.

[37] T. Sebastian, P. Klein, and B. Kimia, "Recognition of shapes by editing their shock graphs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 5, pp. 550–571, May 2004.

[38] G. Harper, G. Bravi, S. Pickett, J. Hussain, and D. Green, "The reduced graph descriptor in virtual screening and data-driven clustering of high-throughput screening data," *J. Chem. Inf. Comput. Sci.*, vol. 44, pp. 2145–2156, 2004.

[39] P. Willett and V. Winterman, "A comparison of some measures for the determination of inter-molecular stuctural similarity," *Quant. Struct.-Act. Relat.*, vol. 5, 1986.

[40] C. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Englewood Cliffs, NJ: Prentice Hall, Inc., 1982.

[41] J. Conway and N. Sloane, *Sphere Packings, Lattices and Groups*. New York: Springer-Verlag, 1988.

[42] B. Dunford-Shore, W. Sulaman, B. Feng, F. Fabrizio, J. Holcomb, W. Wise, and T. Kazic, "Klotho: Biochemical compounds declarative database," http://www.biocheminfo.org/klotho/, 2002.

[43] R. Saigal, *Linear Programming: A Modern Integrated Analysis*. Boston: Kluwer Academic Publishers, 1995.

[44] D. Luenberger, *Optimization by Vector Space Methods*. New York: John Wiley and Sons, Inc., 1969.

[45] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York: Cambridge University Press, 2004.

[46] P. Willett, *Clustering in Chemical Information Systems*. Letchworth: Research Studies Press, 1987.

[47] M. Berkelaar, K. Eikland, and P. Notebaert, "lp_solve: open source (mixed-integer) linear programming system," http://groups.yahoo.com/group/lp_solve, May 2004.

[48] M. Cone, R. Venkataraghavan, and F. McLafferty, "Molecular structure comparison program for the identification of maximal common substructures," *J. American Chem. Society*, vol. 99, no. 23, pp. 7668–7671, Nov. 1977.

[49] J. Raymond, E. Gardiner, and P. Willett, "Rascal: calculation of graph similarity using maximum common edge subgraphs," *The Computer Journal*, vol. 45, no. 6, pp. 631–644, 2002.

[50] T. Hagadone, "Molecular substructure similarity searching: efficient retrieval in two-dimensional structure databases," *J. Chem. Inf. Comput. Sci.*, vol. 32, pp. 515–521, 1992.

[51] G. Levi, "A note on the derivation of maximal common subgraphs of two directed or undirected graphs," *Calcolo*, vol. 9, pp. 341–352, 1972.

[52] P. Ostergard, "A fast algorithm for the maximum clique problem," *Discrete Appl. Math.*, vol. 120, pp. 197–207, 2002.