International Journal of Foundations of Computer Science

ON THE EDITING DISTANCE BETWEEN UNDIRECTED ACYCLIC GRAPHS

KAIZHONG ZHANG
Department of Computer Science, University of Western Ontario, London, Ontario, Canada N6A 5B7

JASON T. L. WANG
Department of Computer and Information Science, New Jersey Institute of Technology, University Heights, Newark, New Jersey 07102, USA

and

DENNIS SHASHA
Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, New York 10012, USA

Received May 1995
Revised October 1995
Communicated by T. Jiang, Guest Editor

ABSTRACT

We consider the problem of comparing *CUAL* graphs (*C*onnected, *U*ndirected, *A*cyclic graphs with nodes being *L*abeled). This problem is motivated by the study of information retrieval for bio-chemical and molecular databases. Suppose we define the distance between two CUAL graphs G_1 and G_2 to be the weighted number of edit operations (insert node, delete node and relabel node) to transform G_1 to G_2. By reduction from exact cover by 3-sets, one can show that finding the distance between two CUAL graphs is NP-complete. In view of the hardness of the problem, we propose a constrained distance metric, called the degree-2 distance, by requiring that any node to be inserted (deleted) have no more than 2 neighbors. With this metric, we present an efficient algorithm to solve the problem. The algorithm runs in time $O(N_1 N_2 D^2)$ for general weighting edit operations and in time $O(N_1 N_2 D\sqrt{D}\log D)$ for integral weighting edit operations, where N_i, $i = 1, 2$, is the number of nodes in G_i, $D = \min\{d_1, d_2\}$ and d_i is the maximum degree of G_i.

Keywords: Approximate pattern matching, chemical compounds, dynamic programming, editing distance, NP-completeness

1. Introduction

In this paper we study the problem of comparing two labeled undirected acyclic graphs (abbreviated as CUAL graphs).[a] Our specific interest in this problem

[a]Such graphs are also known as labeled free trees. When the context is clear, we refer to CUAL

comes from its applicability in bio-chemical information systems.[1,2] Undirected labeled graphs have been used to represent two-dimensional (2-D) compounds and molecules. For example, Figure 1(a) shows two graph representations of 2-D compounds; each node in the graphs represents an atom and each edge represents a bond. The compounds can be represented alternatively as two CUAL graphs (Figure 1(b)).

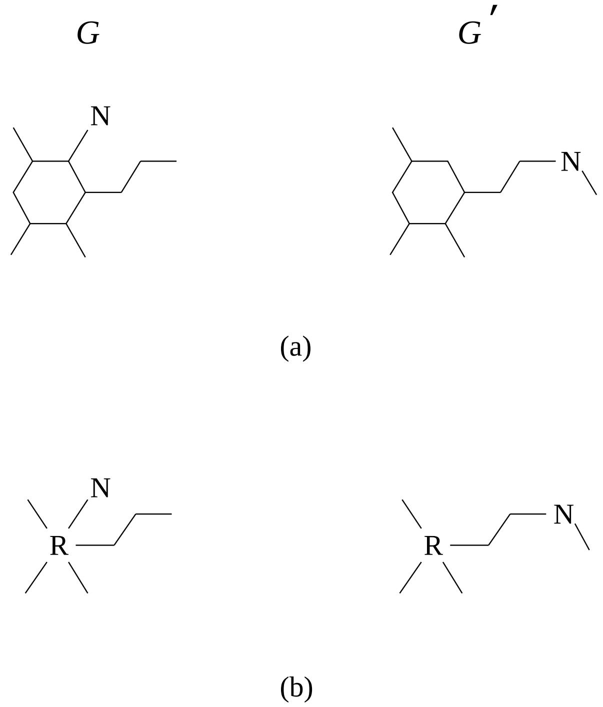

Fig. 1. (a) Two examples of chemical compounds.[7] N represents a nitrogen atom. Omitted node labels are carbon atoms (C). Hydrogen atoms (H) are not included in the graph representations since their presence or absence can be deduced from the other information. (b) The same compounds can be represented as CUAL graphs, with each ring being represented by a special node label R.

One common use of the bio-chemical information systems is to perform a *similarity search*, i.e., to find database compounds that are similar to a query structure.[6,12,13] Many similarity measures have been devised; they are usually calculated by considering the atom, bond, or ring-centered substructural fragments found in common in the query and in a compound.[b] While these measures are often useful, they don't

graphs simply as graphs. Note that, in practice, edges of a graph may have labels. In that case, one can transform a labeled edge between two nodes u and v to a labeled node connecting u and v.

[b]For example, the much used fragment weighting scheme works by considering the number of occurrences of a particular fragment type within a compound.[14] The more frequently a fragment occurs, the greater weight it gains. Thus, a pair of molecules that had several occurrences of a given fragment in common would be considered to be more similar to each other than if they had only a single occurrence in common.

capture many of the interesting topological differences between two compounds, which play a key role in identifying the difference in the compounds' functionalities.

This paper presents a new (dis)similarity measure for graphs. We define the distance between two graphs G_1 and G_2 to be the weighted number of edit operations (insert node, delete node and relabel node) to transform G_1 to G_2. By reduction from exact cover by 3-sets, one can show that finding the distance between two graphs is NP-complete. In view of the hardness of the problem, we propose a constrained distance metric, called the *degree-2 distance*, by requiring that any node to be inserted (deleted) have no more than two neighbors. This metric is a natural extension of the edit distance for strings[10] and Selkow's[9] distance for trees. As our next example shows, it is also a practically useful metric for graphs.

Example 1 Consider the atom-centered fragment f and the three compounds G_1, G_2, G_3 in Figure 2. According to the aforementioned fragment weighting scheme, G_1 is closer to G_3 than to G_2 because f occurs twice in both of the G_1 and G_3, whereas it occurs only once in G_2. On the other hand, according to the proposed degree-2 distance metric, G_1 is closer to G_2 than to G_3.[c] Visually, G_1 is closer to G_2, consistent with the degree-2 metric; therefore we argue that this is a plausible metric. □

Thus our work provides a complementary measure capable of reflecting the structural differences between chemical compounds (except that our graphs must be acyclic, so rings must be reduced to single nodes). We believe the presented techniques can also contribute to comparison and search of 2-D and 3-D (macro)molecules in protein and DNA structures.[8]

1.1. The Main Results

Let G_1 and G_2 be two given graphs. We first show that finding the editing distance between G_1 and G_2 is NP-complete. We then develop an algorithm to find the degree-2 distance between the two graphs that runs in time $O(N_1 N_2 D^2)$ for general weighting edit operations and in time $O(N_1 N_2 D \sqrt{D} \log D)$ for integral weighting edit operations. Here, N_i, $i = 1, 2$, represents the number of nodes in G_i; $deg(n)$ denotes the number of neighbors of node n; $d_i = \max_{n \in G_i} deg(n)$ and $D = \min\{d_1, d_2\}$. A subroutine of this algorithm computes the degree-2 distance between two unordered trees in time $O(N_1 N_2 D)$ for general weighting edit operations and in time $O(N_1 N_2 \sqrt{D} \log D)$ for integral weighting edit operations. (An unordered tree is a rooted tree in which the order among siblings is unimportant. For such trees, $deg(n)$ is defined as the number of n's children, excluding n's parent.)

The rest of the paper is organized as follows. Section 2 introduces some notation and definitions used in the paper and shows the NP-completeness result. Section 3 presents the algorithm for the unordered tree case. This algorithm is then used as a subroutine to compute the degree-2 distance between graphs (Section 4). We conclude the paper in Section 5.

---

[c] Assume all edit operations have unit cost. The degree-2 distance between G_1 and G_2 is 1, obtained by relabeling the leftmost C in G_1 to N in G_2, whereas the distance between G_1 and G_3 is 7, obtained by inserting the seven Rs into G_3.

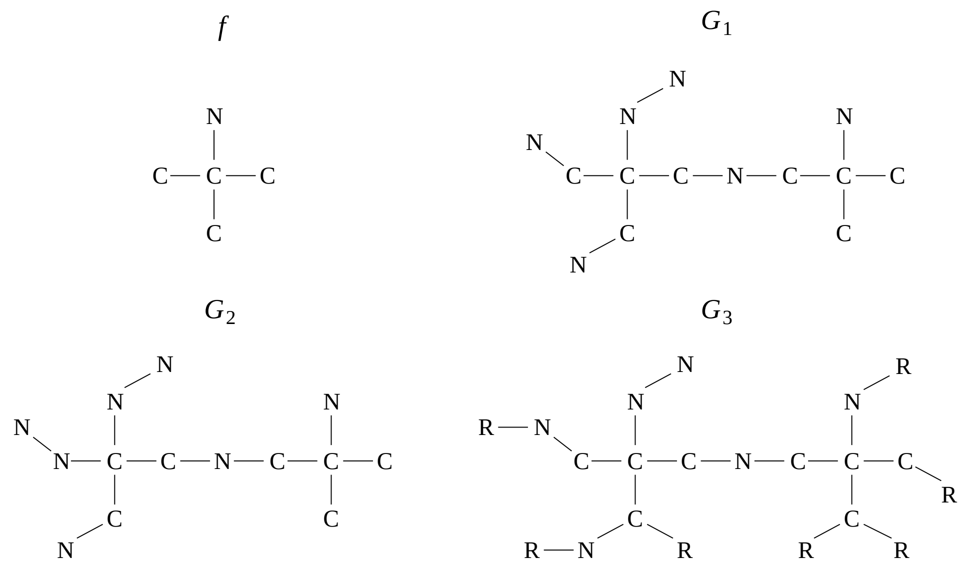

Fig. 2. An augmented atom (fragment) and three chemical compounds.

2. Preliminaries

2.1. Edit Operations on Graphs

There are three kinds of edit operations on graphs: relabel, delete and insert a node. Relabeling node n means changing the label on n. Deleting a node n means making the neighbors of n (except an arbitrarily specified neighbor n') become the neighbors of n' and then removing n. (This amounts to contraction of the edge between n and n'[3] and making the resulting node have label of n'.) Insert is the complement of delete. This means that inserting n as a neighbor of n' makes a subset of the current neighbors of n' become the neighbors of n. We represent an edit operation as a pair $(u, v) \neq (\Lambda, \Lambda)$, sometimes written $u \to v$. We call $u \to v$ a relabeling operation if $u \neq \Lambda$ and $v \neq \Lambda$; a delete operation if $v = \Lambda$; and an insert operation if $u = \Lambda$. Let G_2 be the graph that results from the application of an edit operation $u \to v$ to graph G_1; this is written $G_1 \Rightarrow G_2$ via $u \to v$.

Let S be a sequence $s_1, s_2, \ldots, s_k$ of edit operations. S transforms graph G to graph G' if there is a sequence of graphs $G_0, G_1, \ldots, G_k$ such that $G = G_0, G' = G_k$ and $G_{i-1} \Rightarrow G_i$ via s_i for $1 \leq i \leq k$. Let γ be a cost function that assigns to each edit operation $u \to v$ a nonnegative real number $\gamma(u \to v)$. We require γ to be a metric. By extension, the cost of the sequence S, denoted $\gamma(S)$, is simply the sum of costs of the constituent edit operations. The *distance* from G to G', denoted $\Delta(G, G')$, is the minimum cost of all sequences of edit operations taking G to G'.

Theorem 1 *Finding $\Delta(G, G')$ is NP-complete.*

Proof. The proof that this problem can be solved by a nondeterministic polynomial time algorithm is straightforward: given a distance value, just guess a sequence of edit operations for transforming G to G' and see whether the cost incurred is less than the distance value. To show that the problem is NP-hard, we transform the exact cover by 3-sets problem[5] to it.

Exact Cover by 3-Sets (X3C)

Instance: A finite set S with $|S| = 3k$ and a collection $\mathcal{C}$ of 3-element subsets of S.

Question: Does $\mathcal{C}$ contain an exact cover for S, that is, a subcollection $\mathcal{C}' \subseteq \mathcal{C}$ such that every element of S occurs in exactly one member of $\mathcal{C}'$?

Given an instance of the X3C problem, let the set $S = \{s_1, s_2, \ldots, s_m\}$, where $m = 3k$. Let $\mathcal{C} = \{C_1, C_2, \ldots, C_n\}$. Here each $C_i = \{t_{i_1}, t_{i_2}, t_{i_3}\}$ where $t_{i_j} \in S$, $1 \leq j \leq 3$. Without loss of generality, assume $n > k$. We construct two graphs as shown in Figure 3. This construction can be done in polynomial time.

Claim. If $\Delta(G, G') = k + 3(n-k)$, then $\mathcal{C}$ contains an exact cover for S.

Proof of Claim. Assume we use l_1 deletes, l_2 inserts and l_3 relabelings to change G to G'. Since changing node labels does not change the number of nodes in the graphs, $|G| - l_1 + l_2 = |G'|$. Thus, $l_1 - l_2 = |G| - |G'| = 4n+1-(4(n-k)+3k+1) = k$. Since only inserts and relabelings can create new node labels, $l_2 + l_3 \geq 3(n-k)$. Thus $l_1 + l_2 + l_3 = (l_1 - l_2) + (l_2 + l_3) + l_2 \geq k + 3(n-k) + l_2$. So, $\Delta(G, G') \geq k + 3(n-k) + l_2$. If $\Delta(G, G') = k + 3(n-k)$, then $l_2 = 0$. Since $l_1 = |G| - |G'| = k$, $l_3 = 3(n-k)$.

Now, because (i) G has k more nodes than G', and (ii) G has k more nodes labeled *set* than G', we have to use the l_1 deletes to remove the nodes labeled *set*. On the other hand, because G' has $3(n-k)$ nodes labeled Λ, we have to use the l_3 relabeling operations to change the labels. Thus after applying the l_1 deletes and l_3 relabelings, the rest of G must be equivalent to G'. Therefore $\mathcal{C}$ contains an exact cover for S. □

Next, given an exact cover for S, it is easy to see that $\Delta(G, G') = k + 3(n-k)$. Therefore $\Delta(G, G') = k + 3(n-k)$ iff $\mathcal{C}$ contains an exact cover for S. Thus, finding $\Delta(G, G')$ is NP-hard. □

2.2. Degree-2 Distance

In view of the hardness of the problem, we propose to impose the following *degree-2* constraint on the edit operations: a node n can be deleted (inserted) only when $deg(n) \leq 2$.[d] Intuitively one can delete either a leaf or a node n with two neighbors; in the latter case, after deleting n, we simply connect its two neighbors together. When inserting n between two nodes n' and n'', we remove the edge between n' and n'' and make n the neighbor of both n' and n''. These constrained edits will be referred to as the degree-2 edit operations; they are natural in manipulating nodes and edges in the updated graphs. We define the degree-2 distance between graph G and graph G', denoted $\delta(G, G')$, to be the minimum cost of all sequences of the degree-2 edit operations transforming G to G'. Clearly δ is a metric.

Remark. The degree-2 constraint is *complete* in the sense that for any two given CUAL graphs G_1 and G_2, one can always transform G_1 to G_2 by applying degree-2 edit operations to them as follows. First delete all leaves of G_1 (nodes with degree

[d] Thus, to delete a node n with $deg(n) > 2$, one has to first delete some of its neighbors to make its degree less than or equal to 2 before removing it.

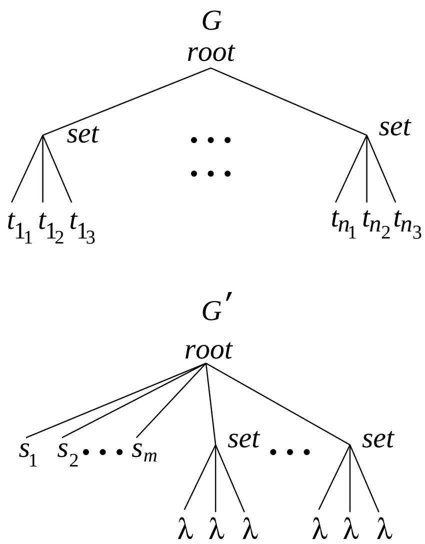

Fig. 3. Two graphs G and G' constructed based on an instance of the X3C problem.

1). These deletions will result in some new leaves. Continue deleting the new leaves until no node is left in G_1. Then insert a node n of G_2 into the empty graph. Then insert the neighbors of n. Next, insert new neighbors to those neighbors. Each such insertion is a degree-2 edit operation. Continue these insertions until G_2 is obtained.

2.3. Mappings

The degree-2 edit operations correspond to a *mapping*, which is a graphical specification of what edit operations apply to each node in the two graphs. For example, the mapping in Figure 4 shows a way to transform the CUAL graph G to the CUAL graph G' given in Figure 1. It corresponds to the sequence (delete (node with label N), insert (node with label N)).

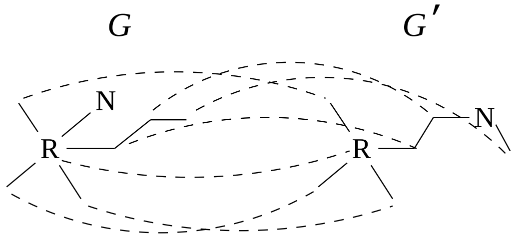

Fig. 4. A mapping from G to G'. Nodes in G not touched by a mapping line are to be deleted; nodes in G' not touched by a mapping line are to be inserted. The mapping shows a way to transform G to G'.

To formalize the notion of mappings, we need some definitions. Let u, v, w be three nodes in a graph G; let $[u, v]$ denote the path between node u and node v inclusively. Define the *center* of the three nodes u, v, w, denoted $center(u, v, w)$, to be the intersection node of the three paths $[u, v]$, $[v, w]$ and $[w, u]$. Figure 5 illustrates the definition.

Fig. 5. Illustrations of the center, which is represented by the bullet •.

Let $g[i]$ represent the ith node of graph G according to some ordering (e.g., a depth-first search order). Formally, a mapping from G to G' is a triple (M, G, G') (or simply M when the context is clear), where M is any set of pairs of integers (i, j) satisfying the following conditions:

1. $1 \leq i \leq |G|$, $1 \leq j \leq |G'|$.
2. For any two pairs (i_1, j_1) and (i_2, j_2) in M, $i_1 = i_2$ iff $j_1 = j_2$ (one-to-one).
3. For any three pairs (i_1, j_1), (i_2, j_2) and (i_3, j_3) in M, (i^*, j^*) is also in M where $g[i^*] = center(g[i_1], g[i_2], g[i_3])$ and $g'[j^*] = center(g'[j_1], g'[j_2], g'[j_3])$ (center relationship preservation).

The cost of M, denoted $\gamma(M)$, is the cost of deleting nodes of G not touched by a mapping line plus the cost of inserting nodes of G' not touched by a mapping line plus the cost of relabeling nodes in those pairs related by mapping lines with different labels.

Mappings can be composed. Let M_1 be a mapping from G to G' and let M_2 be a mapping from G' to G''. Define the composition of M_1 and M_2, denoted $M_1 \circ M_2$, as

$$M_1 \circ M_2 = \{(i, j) \mid \exists\ k \text{ s. t. } (i, k) \in M_1 \text{ and } (k, j) \in M_2\}.$$

The following lemmas establish the relationship between mappings and degree-2 edit operations.

Lemma 1 *(i) $M_1 \circ M_2$ is a mapping from G to G''. (ii) $\gamma(M_1 \circ M_2) \leq \gamma(M_1) + \gamma(M_2)$.*

Proof. (i) follows from the definition of mappings. For (ii), let I be the set of nodes in G not touched by mapping lines, and let J be the set of nodes in G'' not touched by mapping lines. For any $g[i] \in G$ and $g''[j] \in G''$, three cases may occur: $(i, j) \in M_1 \circ M_2$, $g[i] \in I$ or $g''[j] \in J$. Each case corresponds to a degree-2 edit operation $u \to v$ where u and v may be node labels or may be Λ. In all the cases, the triangle inequality on the distance metric δ ensures that $\gamma(u \to v) \leq \gamma(u \to w) + \gamma(w \to v)$. □

Lemma 2 *Given S, a sequence $s_1, s_2, \ldots, s_k$ of degree-2 edit operations from G to G', there exists a mapping M from G to G' such that $\gamma(M) \leq \gamma(S)$.*

Proof. The lemma is proved by induction on k. The base case ($k = 1$) holds, since any single degree-2 edit operation preserves the center relationship in the mapping. For the general case ($k \geq 2$), let S_1 be the sequence $s_1, \ldots, s_{k-1}$ of degree-2 edit operations. By induction hypothesis, there exists a mapping M_1 such that $\gamma(M_1) \leq \gamma(S_1)$. Let M_2 be the mapping for s_k. By Lemma 1, we have $\gamma(M)$ $= \gamma(M_1 \circ M_2) \leq \gamma(M_1) + \gamma(M_2) \leq \gamma(S)$. □

Lemma 3 *For any mapping M from G to G', there exists a sequence of degree-2 edit operations S such that $\gamma(S) = \gamma(M)$.*

Proof. From the mapping conditions, if (i_1, j_1), (i_2, j_2) and (i_3, j_3) are in M, then (i^*, j^*) is also in M where $g[i^*] = center(g[i_1], g[i_2], g[i_3])$ and $g'[j^*]$ $= center(g'[j_1], g'[j_2], g'[j_3])$. Let $T_1 = \{g[i] \mid \exists\ (i_1, j_1) \in M$ and $(i_2, j_2) \in M$ s.t. $g[i] \in [g[i_1], g[i_2]]\}$ and let $T_2 = \{g'[j] \mid \exists\ (i_1, j_1) \in M$ and $(i_2, j_2) \in M$ s.t. $g'[j] \in [g'[j_1], g'[j_2]]\}$. It's easy to see that T_1 and T_2 are connected. In fact, they both are trees.

From the above definitions, every node of G touched by a mapping line must be in T_1. Furthermore, any node $u \in T_1$ not touched by a mapping line must have exactly 2 neighbors where both of the neighbors are in T_1. (Otherwise u would be the intersection node of at least three paths, each of which would contain at least one node in M, implying that u itself would be in M.) For the same reason, any node $u \in T_1$ with $deg(u) > 2$ must be in M.

As a result, we can construct the desired sequence S of degree-2 edit operations as follows. First, delete the nodes $u \in G$ and $u \notin T_1$, starting from nodes with degree 1. After performing these deletes, G becomes T_1. Then, delete the nodes of G that are in T_1 but not in M. These deletes are valid since all the deleted nodes have degree 2. Then, perform all the relabelings indicated by the mapping M. Then, insert the nodes of G' that are in T_2 but not in M, resulting in the tree T_2. Again the inserts are valid since all the inserted nodes have degree 2. Finally insert all the other nodes of G'. Obviously $\gamma(S) = \gamma(M)$, which completes the proof. □

From Lemmas 2 and 3, we obtain

Theorem 2 $\delta(G, G') = \min\{\gamma(M) | M$ *is a mapping from G to G'*$\}$.

Example 2 Consider again the graphs G and G' in Figure 4. The mapping in the figure is a minimum cost mapping and $\delta(G, G') = 2$. □

3. The Algorithm for Rooted Unordered Trees

For notational convenience, in this subsection we use T rather than G to represent a rooted unordered tree. Let $t[i]$ denote the ith node of T according to the depth-first search order. $T[i]$ represents the subtree rooted at $t[i]$ and $F[i]$ represents the forest obtained by deleting $t[i]$ from $T[i]$. Let T_1 and T_2 be two rooted unordered trees. We use $t_1[i_1], t_1[i_2], \ldots, t_1[i_{n_i}]$ to represent the children of $t_1[i]$ in $T_1[i]$ and use $t_2[j_1], t_2[j_2], \ldots, t_2[j_{n_j}]$ to represent the children of $t_2[j]$ in $T_2[j]$. When applied to the rooted unordered trees T_1 and T_2, the degree-2 constraint on the edit operations becomes the following: a node can be deleted (inserted) only when it has fewer than 2 children. The mapping between two rooted unordered trees T_1 and T_2 is the same as defined in § 2.3 except that condition 3) in that definition should be replaced by the following:

for any two pairs (i_1, j_1) and (i_2, j_2) in M, (i^*, j^*) is also in M where $t_1[i^*] = lca(t_1[i_1],\ t_1[i_2])$, $t_2[j^*] = lca(t_2[j_1],\ t_2[j_2])$ and $lca(.)$ represents the least common ancestor of the indicated nodes.

With this notion in mind, it's easy to develop a dynamic programming algorithm for rooted unordered trees. We now present several lemmas, which will be the basis of our algorithm.

Lemma 4 *For all* $1 \leq i \leq N_1$ *and* $1 \leq j \leq N_2$,
(i) $\delta(\emptyset, \emptyset) = 0$;
(ii) $\delta(T_1[i], \emptyset) = \delta(F_1[i], \emptyset) + \gamma(t_1[i] \rightarrow \Lambda)$;
(iii) $\delta(F_1[i], \emptyset) = \sum_{k=1}^{n_i} \delta(T_1[i_k], \emptyset)$;
(iv) $\delta(\emptyset, T_2[j]) = \delta(\emptyset, F_2[j]) + \gamma(\Lambda \rightarrow t_2[j])$;
(v) $\delta(\emptyset, F_2[j]) = \sum_{k=1}^{n_j} \delta(\emptyset, T_2[j_k])$.

Proof. Immediate from definitions. □

Lemma 5 *For all* $1 \leq i \leq N_1$ *and* $1 \leq j \leq N_2$,

$$\delta(T_1[i], T_2[j]) = \mathbf{min} \begin{cases} \delta(T_1[i], \emptyset) + \min_{1 \leq s \leq n_i} \{\delta(T_1[i_s], T_2[j]) - \delta(T_1[i_s], \emptyset)\} \\ \delta(\emptyset, T_2[j]) + \min_{1 \leq t \leq n_j} \{\delta(T_1[i], T_2[j_t]) - \delta(\emptyset, T_2[j_t])\} \\ \delta(F_1[i], F_2[j]) + \gamma(t_1[i] \rightarrow t_2[j]) \end{cases}$$

Proof. Let M be a minimum-cost mapping from $T_1[i]$ to $T_2[j]$. There are four cases to be considered:

Case 1. $i \notin M$ and $j \in M$. Let (z, j) be in M. Thus $t_1[z]$ must be a node in $F_1[i]$. Let $t_1[i_s]$ be the child of $t_1[i]$ on the path from $t_1[z]$ to $t_1[i]$. Thus $\delta(T_1[i], T_2[j]) = \delta(T_1[i_s], T_2[j]) + \delta(T_1[i_1], \emptyset) + \ldots + \delta(T_1[i_{s-1}], \emptyset) + \delta(T_1[i_{s+1}], \emptyset) + \ldots + \delta(T_1[i_{n_i}], \emptyset) + \gamma(t_1[i] \rightarrow \Lambda)$. Since $\delta(T_1[i], \emptyset) = \gamma(t_1[i] \rightarrow \Lambda) + \sum_{k=1}^{n_i} \delta(T_1[i_k],\ \emptyset)$, we can rewrite the right hand side of the formula as $\delta(T_1[i], \emptyset) + \delta(T_1[i_s], T_2[j]) - \delta(T_1[i_s], \emptyset)$. The range of s is from 1 to n_i; therefore we take the minimum of the corresponding costs.

Case 2. $i \in M$ and $j \notin M$. This is analogous to Case 1.

Case 3. $i \in M$ and $j \in M$. By the mapping conditions, (i, j) must be in M. Thus $\delta(T_1[i], T_2[j]) = \delta(F_1[i], F_2[j]) + \gamma(t_1[i] \rightarrow t_2[j])$.

Case 4. $i \notin M$ and $j \notin M$. We would have $\delta(T_1[i], T_2[j]) = \delta(F_1[i], F_2[j]) + \gamma(t_1[i] \rightarrow \Lambda) + \gamma(\Lambda \rightarrow t_2[j])$. Since $\gamma(t_1[i] \rightarrow t_2[j]) \leq \gamma(t_1[i] \rightarrow \Lambda) + \gamma(\Lambda \rightarrow t_2[j])$ (the triangle inequality), we need not include this case in our formula. □

In calculating $\delta(F_1[i], F_2[j])$, we first make the following observation.

Lemma 6 *Suppose* $(i, j) \in M$. *If two nodes* u_1 *and* u_2 *of* $T_1[i_s]$ *are in* M *for some* $1 \leq s \leq n_i$, *then there must exist an integer* t, $1 \leq t \leq n_j$, *such that the two nodes connected to* u_1 *and* u_2, *respectively, by the mapping lines of* M *are in* $T_2[j_t]$.

Proof. Suppose not. Suppose there are h and k such that the node v_1 connected to u_1 by the mapping line is in $T_2[j_h]$ and the node v_2 connected to u_2 by the mapping line is in $T_2[j_k]$. Note that $u_3 = lca(u_1, u_2)$ is in $T_1[i_s]$ and $lca(v_1, v_2) = t_2[j]$. By the mapping conditions, u_3 and $t_2[j]$ must be connected by a mapping line in M, contradicting the fact that $(i, j) \in M$. □

Thus, a subtree rooted at some child of $t_1[i]$ must be mapped to a subtree rooted at some child of $t_2[j]$. We try to find a best mapping between the children of $t_1[i]$ and the children of $t_2[j]$ by constructing a weighted bipartite graph BG as follows. Let $U = \{t_1[i_1], \ldots, t_1[i_{n_i}]\}$ and $V = \{t_2[j_1], \ldots, t_2[j_{n_j}]\}$. Assign the weight for each edge $(t_1[i_s], t_2[j_t])$, denoted $\omega((t_1[i_s], t_2[j_t]))$, $1 \le s \le n_i$ and $1 \le t \le n_j$, based on the formula

$$\omega((t_1[i_s], t_2[j_t])) = \delta(T_1[i_s], \emptyset) + \delta(\emptyset, T_2[j_t]) - \delta(T_1[i_s], T_2[j_t]).$$

Without loss of generality, assume $n_i \le n_j$. To better bound the complexity of our algorithm, for each node $u \in U$, we only pick the top n_i highest weighted edges touching on u and store these edges as well as their end nodes in BG. Thus BG has at most $n_i n_i$ edges and at most $n_i + n_i n_i$ nodes. Letting Ma be the maximum weighted matching in BG, we obtain

Lemma 7

$$\delta(F_1[i], F_2[j]) = \sum_{s=1}^{n_i} \delta(T_1[i_s], \emptyset) + \sum_{t=1}^{n_j} \delta(\emptyset, T_2[j_t]) - \sum_{(u,v) \in Ma} \omega((u, v)).$$

Thus the problem of calculating $\delta(F_1[i], F_2[j])$ becomes that of finding the maximum weighted matching in BG. One can solve the problem by using Gabow and Tarjan's algorithm.[4] Figure 6 summarizes our algorithm.

Algorithm A
Input: Unordered trees T_1 and T_2.
Output: $\delta(T_1[i], T_2[j])$ where $1 \le i \le N_1$ and $1 \le j \le N_2$;
$\delta(T_1[N_1], T_2[N_2]) = \delta(T_1, T_2)$.

```
δ(∅, ∅) := 0;
for i := 1 to N1 do
    compute δ(F1[i], ∅) and δ(T1[i], ∅) as in Lemma 4 (ii) (iii);
for j := 1 to N2 do
    compute δ(∅, F2[j]) and δ(∅, T2[j]) as in Lemma 4 (iv) (v);
for i := 1 to N1 do
    for j := 1 to N2 do
        compute δ(F1[i], F2[j]) as in Lemma 7;
        compute δ(T1[i], T2[j]) as in Lemma 5;
```

Fig. 6. Algorithm for computing $\delta(T_1, T_2)$ for two unordered trees T_1 and T_2.

Theorem 3 *The time complexity of Algorithm A is $O(N_1 N_2 D)$ for general weighting edit operations and $O(N_1 N_2 \sqrt{D} \log D)$ for integral weighting edit operations.*

Proof. By Lemma 5, the complexity of computing $\delta(T_1[i], T_2[j])$ is bounded by $O(n_i + n_j)$. In constructing BG for calculating $\delta(F_1[i], F_2[j])$, for each node $u \in U$, it takes $O(n_j)$ time to calculate the weights of the edges touching on u and to pick the n_i edges with the highest weights. Thus, it takes a total of $O(n_i n_j)$ time to construct BG. Let V be the number of nodes in BG and let E be the number of edges in BG. The complexity of finding the maximum weighted matching in BG is $O(\min\{n_i, n_j\}(E + V \log V))$ when the edges have general weights and is $O(\sqrt{V}E\log(VW))$ when the edges have integral weights where W is the maximum weight.[4] Without loss of generality, assume $n_i \le n_j$. Then V is at most $n_i + n_i n_i$ and E is at most $n_i n_i$. Thus for the general weighting case, the complexity of computing $\delta(T_1[i], T_2[j])$ for any pair of i and j is bounded by $O(n_i n_j + n_i(n_i n_i + V \log V))$. $V = \min\{n_i + n_j, n_i + n_i^2\}$, and therefore $\log V = \log n_i$. Hence the complexity is bounded by

$$\begin{aligned}
& O(n_i n_j + (\min\{n_i, n_j\})^3 + ((\min\{n_i, n_j\})^2 + \min\{n_i, n_j\} \max\{n_i, n_j\}) \log n_i) \\
= \; & O(n_i n_j \log(\min\{n_i, n_j\}) + (\min\{n_i, n_j\})^3) \\
= \; & O(n_i n_j \min\{n_i, n_j\})
\end{aligned}$$

For the integral weighting case, suppose the node label alphabet for unordered trees is finite. Then the maximum edit cost is finite. As a consequence, the maximum weight W in BG is bounded by cV for some constant c. Thus, the complexity is bounded by

$$\begin{aligned}
& O(n_i n_j + \sqrt{n_i + n_j}\, n_i n_i \log(n_i + n_i^2)) \\
= \; & O(n_i n_j + \sqrt{2n_j}\, n_i n_i \log n_i) \\
= \; & O(n_i n_j + n_i n_j n_i / \sqrt{n_j} \log n_i) \\
= \; & O(n_i n_j + n_i n_j \sqrt{n_i} \sqrt{n_i / n_j} \log n_i) \\
= \; & O(n_i n_j + n_i n_j \sqrt{n_i} \log n_i) \\
= \; & O(n_i n_j \sqrt{\min\{n_i, n_j\}} \log(\min\{n_i, n_j\}))
\end{aligned}$$

Therefore for the general weighting case, the complexity of Algorithm A is

$$\begin{aligned}
& \sum_{i=1}^{N_1} \sum_{j=1}^{N_2} O(n_i n_j \min\{n_i, n_j\}) \\
\le \; & \sum_{i=1}^{N_1} \sum_{j=1}^{N_2} O(n_i n_j \min\{d_1, d_2\}) \\
\le \; & O(\min\{d_1, d_2\} \sum_{i=1}^{N_1} n_i \sum_{j=1}^{N_2} n_j) \\
\le \; & O(N_1 N_2 \min\{d_1, d_2\}) \\
= \; & O(N_1 N_2 D)
\end{aligned}$$

Likewise, for the integral weighting case, the complexity of Algorithm A is $O(N_1 N_2 \sqrt{\min\{d_1, d_2\}} \log(\min\{d_1, d_2\})) = O(N_1 N_2 \sqrt{D} \log D)$. □

4. The Algorithm for CUAL Graphs

Let G_1 and G_2 be two CUAL graphs. By definition, if (i, j) is in a minimum cost mapping M from G_1 to G_2, then we can assign $g_1[i]$ to be the root of G_1 and assign

$g_2[j]$ to be the root of G_2, resulting in two rooted unordered trees, denoted as T_1 and T_2. It is clear that M is a mapping (as defined in § 3) between these two rooted unordered trees since for any two pairs (i_1, j_1) and (i_2, j_2) in M, $lca(t_1[i_1], t_1[i_2])$ $=center(g_1[i_1], g_1[i_2], g_1[i])$ and $lca(t_2[j_1], t_2[j_2])$ $=center(g_2[j_1], g_2[j_2], g_2[j])$. On the other hand, given any mapping M between T_1 and T_2, M is also a mapping (as defined in § 2.3) between G_1 and G_2. Therefore by applying Algorithm A to the two trees, we can find $\delta(G_1, G_2)$. Naively trying every possible pair of trees results in an algorithm that runs in time $O(N_1^2 N_2^2 \min\{d_1, d_2\})$ when the edit operations have general cost, and in time $O(N_1^2 N_2^2 \sqrt{\min\{d_1, d_2\}} \log(\min\{d_1, d_2\}))$ when the edit operations have integral cost.

A more careful analysis leads to a faster algorithm, referred to as Algorithm B. Let us choose an arbitrary node, say r, in G_1 and assign r to be the root of G_1. Thus the graph G_1 can be considered as a rooted unordered tree, denoted as T_1^r. For any node u in G_1, we use $T_1^r[u]$ to represent the unordered tree rooted at u with respect to G_1's root r. Let M be a minimum cost mapping from G_1 to G_2. There are two cases to be considered: in the rooted G_1, (i) there exists a node x such that x is touched by a line of M and all nodes touched by lines of M are in the tree $T_1^r[x]$; (ii) there exist two nodes x_1 and x_2 such that both x_1 and x_2 are touched by lines of M and all nodes touched by lines of M are either in the tree $T_1^r[x_1]$ or in the tree $T_1^r[x_2]$.[e]

Case 1. In this case, we choose an arbitrary node v in G_2 and assign v to be the root of G_2. Call the resulting tree T_2^v. Compute $\delta(T_1^r[r], T_2^v[v])$ using Algorithm A and keep all the intermediate results. Let the neighbors of v be $y_1, y_2, \ldots, y_n$. Then compute $\delta(T_1^r[u], T_2^{y_j}[v])$ for all $u \in T_1^r$ and for $1 \leq j \leq n$. Next we assign y_1 to be the new root of G_2 and repeat the above computation. The order in which we assign the nodes in G_2 to be its root is based on the preorder traversal of T_2^v.

Case 2. Let y_1, y_2 be in G_2 such that $(x_1, y_1) \in M$ and $(x_2, y_2) \in M$. Then we can find an arbitrary edge (v_1, v_2) on the path connecting y_1 and y_2 and split G_2 at the edge into two rooted unordered trees $T_2^{v_2}[v_1]$ and $T_2^{v_1}[v_2]$. Each of $T_1^r[x_1]$, $T_1^r[x_2]$, $T_2^{v_2}[y_1]$, $T_2^{v_1}[y_2]$ is a rooted unordered tree (see Figure 7). The restriction of M on $T_1^r[x_1]$ and $T_2^{v_2}[y_1]$ and on $T_1^r[x_2]$ and $T_2^{v_1}[y_2]$ are mappings between rooted unordered trees and therefore can be obtained during the computation of Case 1. Note that in this case we can assume that $v_1 = y_1$ and $v_2 = y_2$ since if there is a node v on the path between y_1 and y_2, we can construct a new minimum cost mapping M_1 between G_1 and G_2 by adding (r, v) to M. This M_1 is a mapping between two rooted unordered trees, which has been calculated in Case 1.

The above enumerates all possible situations and therefore the distance is the minimum of the two cases.

Theorem 4 *The time complexity of Algorithm B is* $O(N_1 N_2\, D^2)$ *for general weighting edit operations and* $O(N_1 N_2 D \sqrt{D} \log D)$ *for integral weighting edit operations.*

---

[e]Note that there cannot be more than two nodes. If that were true (say there were three nodes x_1, x_2, x_3), the center of the three nodes would be their least common ancestor. By the mapping conditions, this ancestor would also be in the mapping, contradicting the fact that all the nodes touched by mapping lines are in the trees rooted at x_1, x_2, x_3.

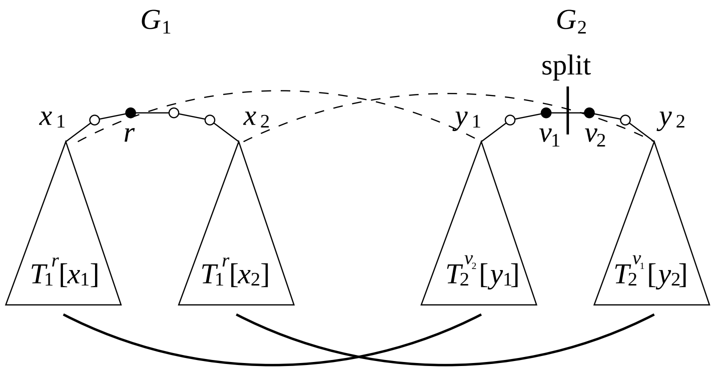

Fig. 7. Illustration of Case 2 in computing $\delta(G_1, G_2)$ for two CUAL graphs G_1 and G_2.

Proof. The computation of $\delta(T_1^r[u], T_2^{y_j}[v])$ for all $u \in T_1^r$ and for $1 \le j \le n$ in Case 1 is dominated by the computation of $\delta(F_1^r[u], F_2^{y_j}[v])$, which in turn is bounded by the computation of the maximum weighted bipartite matching. Let the neighbors of u be $x_1, x_2, \ldots, x_m$. ($y_1, y_2, \ldots, y_n$ are neighbors of v.) If $deg(v) < deg(u)$ (i.e., $n < m$), then we can just apply Gabow and Tarjan's algorithm $deg(v)$ times. On the other hand, if $deg(v) \ge deg(u)$, then in the maximum weighted matching Ma between $x_1, x_2, \ldots, x_m$ and $y_1, y_2, \ldots, y_n$, there exist $deg(v) - deg(u)$ subtrees (among $T_2^v[y_1], T_2^v[y_2], \ldots, T_2^v[y_n]$) that are not in the matching Ma. Suppose y_j is not in Ma for some $1 \le j \le n$. Then we do not need to compute the maximum weighted matching for $F_2^{y_j}[v]$. (If y_j is in Ma, we have to compute the maximum weighted matching.) The number of such computations is therefore $deg(u)$. Thus, we only need to apply Gabow and Tarjan's algorithm $\min(deg(u), deg(v))$ times. Therefore for the general weighting case, the time complexity is

$$\begin{aligned}
& \sum_{i=1}^{N_1}\sum_{j=1}^{N_2} O(n_i n_j \min\{n_i, n_j\}) \min\{n_i, n_j\} \\
\le \; & \sum_{i=1}^{N_1}\sum_{j=1}^{N_2} O(n_i n_j (\min\{d_1, d_2\})^2) \\
\le \; & O(N_1 N_2 (\min\{d_1, d_2\})^2) \\
= \; & O(N_1 N_2 D^2)
\end{aligned}$$

Likewise, for the integral weighting case, the complexity is $O(N_1 N_2 \, (\min\{d_1, d_2\}) \sqrt{\min\{d_1, d_2\}} \log(\min\,\{d_1,\, d_2\})) = O(N_1 N_2 \; D \; \sqrt{D} \, \log D)$.

For Case 2, since the best mapping from $T_1^r[x_1]$ to $T_2^{y_2}[y_1]$ and the best mapping from $T_1^r[x_2]$ to $T_2^{y_1}[y_2]$ can be obtained during the computation of Case 1, this case requires at most $O(N_1 N_2)$ time. Therefore the total time required by Algorithm B is as asserted in the theorem. $\square$

Note that the gap between the running times of Algorithm A and Algorithm B is $\min\{d_1, d_2\}$. If one of the CUAL graphs has a bounded degree, then the running time of both algorithms is $O(N_1 N_2)$.

5. Conclusion

Using this simple, efficient algorithm, a user can submit a query graph and obtain those data graphs approximately matching the query. To our knowledge, this work gives the first polynomial time algorithm ever presented to solve the editing distance problem between undirected acyclic graphs. We are currently implementing the algorithm and integrating it into our pattern matching toolkit.[11]

Acknowledgements

The authors thank Professor Dave Kristol and Dawn Luo for inspirational discussions. We also thank our industrial collaborators Jim Kaminski and Karen Pysniak (Schering-Plough Corporation) and Anthony Noriega (FMC, Princeton) for their constructive suggestions on an early version of this paper.

This work was supported, in part, by the Natural Sciences and Engineering Research Council of Canada under Grant OGP0046373, by the National Science Foundation under Grants IRI-9224601 and IRI-9224602, by the Office of Naval Research under Grant N00014-92-J-1719, and by a grant from the AT&T Foundation.

References

1. J. E. Ash, P. A. Chubb, S. E. Ward, S. M. Welford and P. Willett, *Communication, Storage and Retrieval of Chemical Information* (Ellis Horwood, Chichester, England, 1985).
2. J. E. Ash and E. Hyde, eds., *Chemical Information Systems* (Ellis Horwood, Chichester, England, 1975).
3. H. N. Gabow, Z. Galil and T. H. Spencer, "Efficient implementation of graph algorithms using contraction," *Proc. 25th Annual IEEE Symposium on Foundations of Computer Science*, 1984, pp. 347–357.
4. H. N. Gabow and R. E. Tarjan, "Faster scaling algorithms for network problems," *SIAM J. Comput.* **18** (1989) 1013–1036.
5. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (W. H. Freeman and Company, New York, 1979).
6. M. A. Johnson and G. M. Maggiora, eds., *Concepts and Applications of Molecular Similarity* (Wiley, New York, 1990).
7. J. Kaminski, B. Wallmark, C. Briving and B.-M. Andersson, "Antiulcer agents. 5. inhibition of gastric H^+/K^+-ATPase by substituted imidazo[1,2-*a*]pyridines and related analogues and its implication in modeling the high affinity potassium ion binding site of the gastric proton pump enzyme," *Journal of Medicinal Chemistry* **34** (1991) 533–541.
8. E. M. Mitchell, P. J. Artymiuk, D. W. Rice and P. Willett, "Use of techniques derived from graph theory to compare secondary structure motifs in proteins," *Journal of Molecular Biology* **212** (1989) 151–166.
9. A. S. Noetzel and S. M. Selkow, "An analysis of the general tree-editing problem," in *Time Warps, String Edits, and Macromolecules: The Theory and Practice of*

Sequence Comparison, eds. D. Sankoff and J. B. Kruskal (Addison-Wesley, Reading, MA, 1983) pp. 237–252.

10. R. A. Wagner and M. J. Fischer, "The string-to-string correction problem," *J. ACM* **21** (1974) 168–173.
11. J. T. L. Wang, K. Zhang, K. Jeong and D. Shasha, "A system for approximate tree matching," *IEEE Transactions on Knowledge and Data Engineering* **6** (1994) 559–571.
12. W. E. Warr, *Chemical Structures* (Springer-Verlag, Berlin, 1988).
13. P. Willett, *Similarity and Clustering Methods in Chemical Information Systems* (Research Studies Press, Letchworth, 1987).
14. P. Willett, "Algorithms for the calculation of similarity in chemical structure databases," in *Concepts and Applications of Molecular Similarity*, eds. M. A. Johnson and G. M. Maggiora (Wiley, New York, 1990) pp. 43–61.