On the Editing Distance between Undirected Acyclic Graphs and Related Problems*

Kaizhong Zhang† Jason T. L. Wang‡ Dennis Shasha§

December 28, 1994

1 Introduction

Problem We consider the problem of comparing *CUAL* graphs (*C*onnected, *U*ndirected, *A*cyclic graphs with nodes being *L*abeled).[1] Suppose we define the *distance* between two CUAL graphs to be the weighted number (the user chooses the weighting) of edit operations (insert node, delete node and relabel node) to transform one graph to the other. By reduction from exact cover by 3-sets, one can show that finding the distance between two graphs is NP-hard. In view of the hardness of the problem, we propose a constrained distance metric, called the *degree-2 distance*, for graphs by requiring that any node to be inserted (deleted) have no more than 2 neighbors. As will become clear, this constraint is sensible in defining the edit operations on graphs. Further, the measure is a natural extension of the edit distance for strings [22] and Selkow's distance for trees [15].

Main Results We develop algorithms to find the degree-2 distance between a class of limited graphs, including CUAL graphs, planar CUAL graphs, unordered trees and ordered trees. (A planar CUAL graph is one that can be embedded in the plane in such a way that the edges of the embedding intersect only at the nodes of the graph. An unordered tree is a rooted tree in which the order among siblings is unimportant.) Let G_1 and G_2 be two given graphs. Let N_i, $i = 1, 2$, represent the number of nodes in G_i. Let $deg(n)$ denote the number of neighbors of node n (in the rooted tree case, $deg(n)$ is defined as the number of n's children, excluding n's parent); $d_i = \max_{n \in G_i} deg(n)$. Table 1 summarizes the asymptotic time complexities of our algorithms for finding the degree-2 distance between G_1 and G_2 for the general weighting and

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

†Department of Computer Science, The University of Western Ontario, London, Ontario, Canada N6A 5B7 (kzhang@csd.uwo.ca).

‡Department of Computer and Information Science, New Jersey Institute of Technology, University Heights, Newark, New Jersey 07102 (jason@vienna.njit.edu).

§Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, New York 10012 (shasha@cs.nyu.edu).

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

integral weighting edit operations, respectively.

Limited Graph	General Weighting	Integral Weighting
CUAL Graph	$O(N_1 N_2 D^2)$	$O(N_1 N_2 D \sqrt{D} \log D)$
Planar CUAL Graph	$O(N_1 N_2 \log D)$	$O(N_1 N_2 \log D)$
Unordered Tree	$O(N_1 N_2 D)$	$O(N_1 N_2 \sqrt{D} \log D)$
Ordered Tree	$O(N_1 N_2)$	$O(N_1 N_2)$

Table 1. Time complexities of the proposed algorithms for the limited graphs; $D = \min\{d_1, d_2\}$.

Significance of the Work Undirected labeled graphs have long been used to represent two-dimensional (2-D) chemical compounds and molecules in chemical information systems [2, 3]. Figure 1(a) shows two examples 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)).

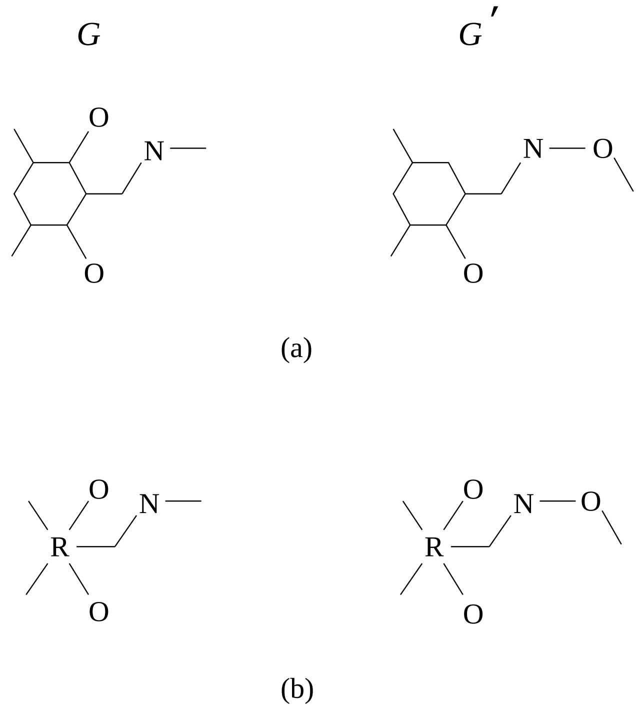

Figure 1. (a) Two examples of chemical compounds [8]. N represents a nitrogen atom and O represents an oxygen atom. Omitted node labels are carbon atoms (C). Hydrogen atoms (H) are not included in the graph representations since their presence or absence can be deduced from the other information. (b) The same compounds can be represented as CUAL graphs, with each ring being represented by a special node label R.

There are two common uses of the chemical information systems. The first, referred to as a *structure search*, is to recognize if a compound has been included in the data file previously, and if not, to register it in the file [12]. Here the root subroutine is based on graph isomorphism algorithms. The second, referred to as a *similarity search*, is to find compounds that are similar to a query structure [7, 23, 24]. Many similarity measures have been devised; they are usually based on the number of atom, bond, or ring-centered substructural fragments found in common in the query and in a compound. While these measures are often useful, they don't capture many of the interesting topological differences between two compounds, which play a key role in identifying the difference in the compounds' functionalities. 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 [14].

Comparison to Past Research This paper generalizes the work on the edit distance between strings [6, 11, 13, 16, 20, 21, 25] and trees [19, 28, 29]. Various kinds of constrained and generalized edit distance on strings and trees have been developed [1, 9, 10, 17, 27]. Our degree-2 distance, when applied to unordered trees, is a restricted form of the constrained distance previously reported in [27]. When applied to ordered trees, the degree-2 distance is a generalized measure of the constrained distance originated from Selkow [17], though our algorithm has the same asymptotic complexity as Selkow's algorithm. (Selkow's distance measure requires that all node deletions and insertions occur at leaves i.e., with degree 1.) In this extended abstract, we present algorithms for computing the degree-2 distance between two CUAL graphs and unordered trees. (The result for the latter is used as a subroutine to calculate the former.) The entire set of algorithms is in the full paper and is available from the authors. We have chosen to present these two results, because we believe they are the most useful of our results to date for approximate graph matching.

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' [4] 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-hard.*
Proof. Similar to the NP-completeness proof given in [26]. □

2.2 Degree-2 Distance

In view of the hardness of the problem, we propose to impose the following constraint on the edit operations: a node n can be deleted (inserted) only when $deg(n) \leq 2$.[2] 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.

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 2 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 O), insert (node with label O)).

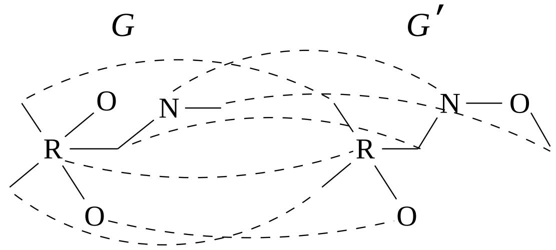

Figure 2. A mapping from G to G'. Nodes in G not touched by a mapping line are to be deleted; nodes in G' not touched by a mapping line are to be inserted. The mapping shows a way to transform G to G'.

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. 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 3 illustrates the definition.

Figure 3. Illustrations of the center, which is represented by the bullet $\bullet$.

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]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.

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.

Lemma 1. *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 $\gamma(M) \leq \gamma(S)$. Conversely, for any mapping M, there exists a sequence of degree-2 edit operations S such that $\gamma(S) = \gamma(M)$.*

Hence, $\delta(G, G') = \min\{\gamma(M) | M$ is a mapping from G to $G'\}$. As an example, consider again the graphs G and G' in Figure 2. The mapping in the figure is a minimum cost mapping and $\delta(G, G') = 2$.

3 Algorithms

We first present the algorithm for finding the degree-2 distance between two rooted unordered trees and then extend the algorithm to handle graphs.

3.1 The Algorithm for 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 mapping M defined in § 2.3 is exactly the same as the edit distance mapping between the unordered trees with the following constraint: 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.[3] 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 2. *For all $1 \leq i \leq N_1$ and $1 \leq j \leq N_2$,*
(i) $\delta(\emptyset, \emptyset) = 0$;
(ii) $\delta(T_1[i], \emptyset) = \delta(F_1[i], \emptyset) + \gamma(t_1[i] \to \Lambda)$;
(iii) $\delta(F_1[i], \emptyset) = \sum_{k=1}^{n_i} \delta(T_1[i_k], \emptyset)$;
(iv) $\delta(\emptyset, T_2[j]) = \delta(\emptyset, F_2[j]) + \gamma(\Lambda \to t_2[j])$;
(v) $\delta(\emptyset, F_2[j]) = \sum_{k=1}^{n_j} \delta(\emptyset, T_2[j_k])$.

[3] An edit distance mapping M_e between two rooted unordered trees satisfies the node one-to-one relationship and preserves the ancestor relationship, i.e., supposing u is mapped to v and x is mapped to y in M_e, u is an ancestor of x iff v is an ancestor of y [18].

Lemma 3. *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 k 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. $\square$

In calculating $\delta(F_1[i], F_2[j])$, notice that if two nodes $t_1[x_1]$ and $t_1[x_2]$ of $T_1[i_s]$ are in M, then by the mapping conditions there must exist an integer t such that the two nodes connected to $t_1[x_1]$ and $t_1[x_2]$, respectively, by the mapping lines of M are in $T_2[j_t]$. 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 $\omega((t_1[i_s], t_2[j_t]))$, $1 \leq s \leq n_i$ and $1 \leq t \leq n_j$, 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 $n_i \leq n_j$. 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. Let Ma be the maximum weighted matching in BG.

Lemma 4.

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

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 in [5]. Figure 4 summarizes the 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 2 (ii) (iii);
for j := 1 to N2 do
    compute δ(∅, F2[j]) and δ(∅, T2[j]) as in Lemma 2 (iv) (v);
for i := 1 to N1 do
    for j := 1 to N2 do
        compute δ(F1[i], F2[j]) as in Lemma 4;
        compute δ(T1[i], T2[j]) as in Lemma 3;
```

Figure 4. Algorithm for computing $\delta(T_1, T_2)$ for two unordered trees T_1 and T_2.

Time Complexity The complexity of computing $\delta(T_1[i], T_2[j])$ is, by Lemma 3, 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 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 [5]. 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 $n_i n_i$. 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 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 $\log V = \log n_i$. Thus 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, the complexity is bounded by

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

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

$$\sum_{i=1}^{N_1} \sum_{j=1}^{N_2} O(n_i n_j \min\{n_i, n_j\})$$

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

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

3.2 The Algorithm for CUAL Graphs

Let G_1 and G_2 be two CUAL graphs. By the definition, if (i, j) is in a minimum cost mapping from G_1 to G_2, then we can assign $g_1[i]$ as the root of G_1 and assign $g_2[j]$ as the root of G_2, resulting in two rooted unordered trees. By applying Algorithm A to the two trees, we can find $\delta(G_1, G_2)$. This naive algorithm 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. Let us choose an arbitrary node, say r, in G_1 and assign r as the root of G_1. Thus the graph G_1 can be considered as a rooted unordered tree. 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. The distance is the minimum of the following two cases: 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 $T_1^r[x_2]$.[4]

Case 1. In this case, $\delta(G_1, G_2)$ can be obtained by trying each node of G_2, in turn, as the root and running a modified version of Algorithm A in each trial. We can show that this case requires at most $O(N_1\ N_2\ (\min\{d_1, d_2\})^2)$ time when the edit operations have general cost and $O(N_1\ N_2 \min\{d_1, d_2\} \sqrt{\min\{d_1, d_2\}} \log(\min\{d_1, d_2\}))$ time when the edit operations have integral cost.

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 5). The best mapping from $T_1^r[x_1]$ to $T_2^{v_2}[y_1]$ and the best mapping from $T_1^r[x_2]$ to $T_2^{v_1}[y_2]$ can be obtained during the computation of Case 1 above. We can show that this case requires at most $O(N_1 N_2)$ time.

Thus, the algorithm for calculating $\delta(G_1, G_2)$ for two CUAL graphs G_1 and G_2, referred to as Algorithm B, runs in time $O(N_1 N_2\ (\min\{d_1, d_2\})^2)$ when the edit operations have general cost and in time $O(N_1 N_2 \min\{d_1, d_2\} \sqrt{\min\{d_1, d_2\}} \log(\min\{d_1, d_2\}))$ when the edit operations have integral cost. 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 $O(N_1 N_2)$.

[4] 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.

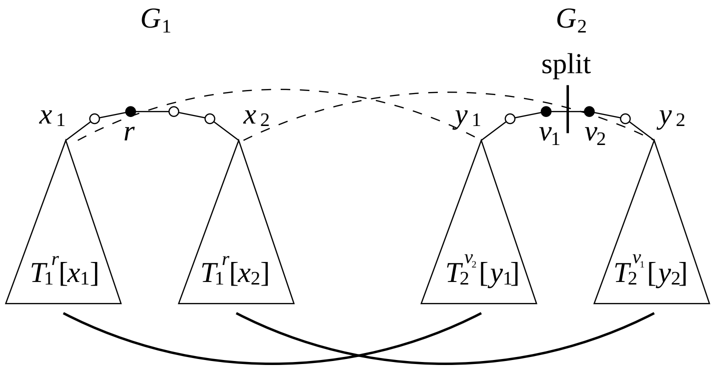

Figure 5.

4 Conclusion

Using these simple, efficient algorithms, a user can submit a query structure and obtain those data structures approximately matching the query. To our knowledge, this work gives the first polynomial time algorithm ever presented to solve the edit distance problem between undirected acyclic graphs.

Acknowledgements

We thank Jim Kaminski and Karen Pysniak of the Schering-Plough Research Institute for very helpful discussions concerning this work.

References

[1] K. Abrahamson. Generalized string matching. *SIAM J. Comput.*, 16:1039–1051, 1987.

[2] 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.

[3] J. E. Ash and E. Hyde, editors. *Chemical Information Systems.* Ellis Horwood, Chichester, England, 1975.

[4] H. N. Gabow, Z. Galil, and T. H. Spencer. Efficient implementation of graph algorithms using contraction. In *Proceedings of the 25th Annual IEEE Symposium on Foundations of Computer Science*, pages 347–357, 1984.

[5] H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for network problems. *SIAM J. Comput.*, 18(5):1013–1036, 1989.

[6] Z. Galil and K. Park. An improved algorithm for approximate string matching. *SIAM J. Comput.*, 19(6):989–999, Dec. 1990.

[7] M. A. Johnson and G. M. Maggiora, editors. *Concepts and Applications of Molecular Similarity.* Wiley, New York, 1990.

[8] 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*, 34:533–541, 1991.

[9] R. L. Kashyap and B. J. Oommen. An effective algorithm for string correction using a generalized edit distance – I. Description of the algorithm and its optimality. *Information Sci.*, 23(2):123–142, 1981.

[10] P. Kilpelainen and H. Mannila. Query primitives for tree-structured data. In M. Crochemore and D. Gusfield, editors, *Combinatorial Pattern Matching, Lecture Notes in Computer Science, 807*, pages 213–225. Springer-Verlag, 1994.

[11] G. M. Landau and U. Vishkin. Fast parallel and serial approximate string matching. *Journal of Algorithms*, 10(2):157–169, 1989.

[12] Y. C. Martin, M. G. Bures, and P. Willett. Searching databases of three-dimensional structures. In K. B. Lipkowitz and D. D. Boyd, editors, *Reviews in Computational Chemistry*, pages 213–263. VCH Publishers, New York, 1990.

[13] W. J. Masek and M. S. Paterson. A faster algorithm for computing string editing distances. *J. Comput. System Sci.*, 20:18–31, 1980.

[14] 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*, 212:151–166, 1989.

[15] A. S. Noetzel and S. M. Selkow. An analysis of the general tree-editing problem. In D. Sankoff and J. B. Kruskal, editors, *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, pages 237–252. Addison-Wesley, Reading, MA, 1983.

[16] P. A. Pevzner and M. S. Waterman. A fast filtration algorithm for the substring matching problem. In A. Apostolico, M. Crochemore, Z. Galil, and U. Manber, editors, *Combinatorial Pattern Matching, Lecture Notes in Computer Science, 684*, pages 197–214. Springer-Verlag, 1993.

[17] S. M. Selkow. The tree-to-tree editing problem. *Information Processing Letters*, 6(6):184–186, Dec. 1977.

[18] D. Shasha, J. T. L. Wang, K. Zhang, and F. Y. Shih. Exact and approximate algorithms for unordered tree matching. *IEEE Transactions on Systems, Man and Cybernetics*, 24(4):668–678, April 1994.

[19] K.-C. Tai. The tree-to-tree correction problem. *J. ACM*, 26(3):422–433, 1979.

[20] E. Ukkonen. Finding approximate patterns in strings. *Journal of Algorithms*, 6:132–137, 1985.

[21] E. Ukkonen. Approximate string-matching over suffix trees. In A. Apostolico, M. Crochemore, Z. Galil, and U. Manber, editors, *Proc. of the 4th Annual Symposium on Combinatorial Pattern Matching*, pages 228–242. Lecture Notes in Computer Science, 684, Springer-Verlag, 1993.

[22] R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, Jan. 1974.

[23] W. E. Warr. *Chemical Structures.* Springer-Verlag, Berlin, 1988.

[24] P. Willett. *Similarity and Clustering Methods in Chemical Information Systems.* Research Studies Press, Letchworth, 1987.

[25] S. Wu and U. Manber. Fast text searching allowing errors. *Communications of the ACM*, 35(10):83–91, Oct. 1992.

[26] K. Zhang. *The Editing Distance between Trees: Algorithms and Applications.* PhD thesis, Courant Institute of Mathematical Sciences, New York University, 1989.

[27] K. Zhang. A new editing based distance between unordered labeled trees. In A. Apostolico, M. Crochemore, Z. Galil, and U. Manber, editors, *Combinatorial Pattern Matching, Lecture Notes in Computer Science, 684*, pages 254–265. Springer-Verlag, 1993; journal version is to appear in *Algorithmica.*

[28] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18(6):1245–1262, Dec. 1989.

[29] K. Zhang, D. Shasha, and J. T. L. Wang. Approximate tree matching in the presence of variable length don't cares. *Journal of Algorithms*, 16(1):33–66, Jan. 1994.