Pattern Matching in Unordered Trees
(Extended Abstract)*

Dennis Shasha  Jason Tsong-Li Wang  Kaizhong Zhang  Frank Y. Shih
(shasha@cs.nyu.edu) (jason@cis.njit.edu) (kzhang@csd.uwo.ca) (yshih@cis.njit.edu)
Courant Institute Dept. of CIS Dept. of CIS Dept. of CIS
New York University N. J. I. T. N. J. I. T.
New York, NY 10012 Newark, NJ 07102 Newark, NJ 07102

Abstract

We consider the problem of comparison between unordered trees, i.e., trees for which the order among siblings is unimportant. The criterion for comparison is the distance as measured by a weighted sum of the costs of deletion, insertion and relabel operations on tree nodes. Such comparisons may contribute to pattern recognition efforts in any field (e.g., genetics) where data can naturally be characterized by unordered trees. We first observe that the problem is NP-complete. Then we present an enumerative algorithm and several heuristics leading to approximate solutions. The algorithms are based on probabilistic hill climbing and bipartite matching techniques. The paper evaluates the accuracy and time efficiency of the heuristics by applying them to a set of trees transformed from industrial parts based on a previously proposed morphological model.

1 Introduction

Comparing trees has applications in image processing [18], molecular biology [20], natural language processing [14], and other fields of pattern recognition [4, 8, 12]. A commonly used technique is to compare an unknown pattern (tree) against idealized template trees in order to assign the new pattern to the category to which the majority of its closest template trees belong [7].

For example, various researchers [5] have observed that the secondary structures of RNA (trees) influence translation rates from RNA to proteins. Because different sequences can produce similar secondary structures [6], comparisons among secondary structures are necessary to understanding the comparative functionality of different RNAs.

Much research in this area concerns the comparison of ordered trees (i.e., trees in which the left-to-right order among siblings is fixed) [4, 8, 24, 28, 29]. However, many important problems in genetics (e.g., determining genetic diseases based on ancestry tree patterns) and other fields suggest that comparisons among unordered trees may be of great significance. This paper is concerned with pattern matching in such trees.

Section 2 presents basic definitions and summarizes the arguments showing that the problem of unordered tree matching is NP-complete (details can be found in [30]). Section 3 presents an efficient enumerative algorithm. Section 4 gives a state space model for the heuristic approach we choose, and several approximate algorithms. Section 5 discusses empirical results. Section 6 concludes the paper.

2 Editing Distance Between Trees

2.1 Edit Operations

Unless otherwise stated, all trees we consider in the paper are rooted, labeled, and unordered (i.e., only ancestor-descendant relationships are important; the order among siblings is unimportant). The distance metric used here is a generalization of the editing distance between strings [25] and ordered trees [28]. There are three kinds of edit operations. Changing a node n means changing the label on n. Deleting a node n means making the children of n become the children of the parent of n and then removing n. Inserting is the complement of deleting. This means that inserting n as the child of m will make n the parent of a subset of the current children of m.

We represent an edit operation as a \rightarrow b, where a is either \Delta or a label of a node in tree T and b is either \Delta or a label of a node in tree \bar{T}. We call a \rightarrow b a relabel operation if a \neq \Delta and b \neq \Delta; a delete operation if b = \Delta; and an insert operation if a = \Delta.

Let T be a sequence c_1, \ldots, c_k of edit operations. An S-derivation from tree T to tree \bar{T} is a sequence of trees
Let \( T_0, \ldots, T_k \) be such that \( T = T_0, \hat{T} = T_k, \) and \( T_{i-1} \rightarrow T_i \) via \( \epsilon_i \) for \( 1 \leq i \leq k. \)

Let \( \gamma \) be a cost function which assigns to each edit operation \( a \rightarrow b \) a nonnegative real number \( \gamma(a \rightarrow b). \) We constrain \( \gamma \) to be a distance metric. That is, (i) \( \gamma(a \rightarrow b) \geq 0, \) \( \gamma(a \rightarrow a) = 0; \) (ii) \( \gamma(a \rightarrow b) = \gamma(b \rightarrow a); \) and (iii) \( \gamma(a \rightarrow c) \leq \gamma(a \rightarrow b) + \gamma(b \rightarrow c). \)

We extend \( \gamma \) to the editing operations sequence \( S \) by letting \( \gamma(S) = \sum_{i=1}^{n} \gamma(e_i). \) Formally the distance between \( T \) and \( \hat{T} \) is defined as:

\[
\delta(T, \hat{T}) = \min \{ \gamma(S) \mid S \text{ is a sequence of edit operations transforming } T \text{ to } \hat{T} \}.
\]

\( \delta \) is also a distance metric according to the definition of \( \gamma. \)

2.2 Mappings

The edit operations correspond to a mapping which is a graphical specification of what edit operations apply to each node in the two trees. The mapping in Figure 1 shows a way to transform \( T \) to \( \hat{T}. \) The mapping corresponds to the sequence (delete (node with label \( d \)), insert (node with label \( f \))).

![Fig. 1. A mapping from \( T \) to \( \hat{T}. \) A dotted line from a node \( u \) in \( T \) to a node \( v \) in \( \hat{T} \) indicates that \( u \) should be changed to \( v \) if \( u \neq v, \) or that \( u \) remains unchanged if \( u = v. \) The nodes of \( T \) not touched by a dotted line are to be deleted and the nodes of \( \hat{T} \) not touched are to be inserted. The mapping shows a way to transform \( T \) to \( \hat{T}. \) The only constraint is that ancestor-descendant relationships be preserved.

Suppose we have an ordering for nodes in a tree. Let \( T[i] \) be the \( i \)th node of tree \( T \) in the given ordering. Formally we define a triple \( (M, T, \hat{T}) \) to be a mapping from \( T \) to \( \hat{T}, \) where \( M \) is any set of pairs of integers \( (i, j) \) satisfying the following constraints:

1. \( 1 \leq i \leq |T|, 1 \leq j \leq |\hat{T}|, \) where \(| \cdot | \) represents the number of nodes in the indicated tree;
2. For any pair of \( (i_1, j_1) \) and \( (i_2, j_2) \) in \( M, \)
   - (a) \( i_1 = i_2 \) iff \( j_1 = j_2 \) (one-to-one),
   - (b) \( T[i_1] \) is an ancestor of \( T[i_2] \) iff \( \hat{T}[j_1] \) is an ancestor of \( \hat{T}[j_2] \) (ancestor relationship preserved).

We will use \( M \) instead of \( (M, T, \hat{T}) \) if there is no confusion. Let \( M \) be a mapping from \( T \) to \( \hat{T}. \) Let \( I \) and \( J \) be the sets of nodes, in \( T \) and \( \hat{T}, \) respectively, not touched by any line in \( M. \) Then we can define the cost of \( M: \)

\[
\gamma(M) = \sum_{(i,j) \in M} \gamma(T[i] \rightarrow \hat{T}[j]) + \sum_{i \in I} \gamma(T[i] \rightarrow \Lambda)+ \sum_{j \in J} \gamma(\Lambda \rightarrow \hat{T}[j]).
\]

Mappings can be composed. Let \( M \) be a mapping from \( T \) to \( \hat{T}, \) and let \( M' \) be a mapping from \( T' \) to \( T''. \) Define \( M \circ M' = \{(i,j) \mid \exists k \text{ s.t. } (i,k) \in M \text{ and } (k,j) \in M'\}. \)

Theorem 2.1. [30] (i) \( M \circ M' \) is a mapping; (ii) \( \gamma(M \circ M') \leq \gamma(M) + \gamma(M'). \)

Theorem 2.2. [30] Given \( S, \) a sequence of edit operations from \( T \) to \( \hat{T}, \) there exists a mapping \( M \) from \( T \) to \( \hat{T} \) such that \( \gamma(M) \leq \gamma(S). \) Conversely, for any mapping \( M, \) there exists a sequence of edit operations \( S \) such that \( \gamma(S) = \gamma(M). \)

Theorem 2.2 implies that the best mapping corresponds to the least cost sequence of edit operations. Intuitively, the sequence can be formed from the deletes induced by the mapping followed by the inserts and relabelings. Thus, we obtain

Theorem 2.3. \( \delta(T, \hat{T}) = \min \{ \gamma(M) \mid M \text{ is a mapping from } T \text{ to } \hat{T} \}. \)

Given two trees \( T \) and \( \hat{T}, \) the unordered tree pattern matching problem is to find \( \delta(T, \hat{T}). \) Such a problem may become intractable, as shown by reduction from exact cover by 3-sets [30].

Theorem 2.4. [30] Finding \( \delta(T, \hat{T}) \) is NP-complete when \( T \) and \( \hat{T} \) are binary trees with a label alphabet of size two.

So, finding \( \delta(T, \hat{T}) \) in general is also NP-complete. We now develop enumerative and approximate algorithms for solving the problem.

3 An Enumerative Algorithm

One may think that the distance between trees \( T \) and \( \hat{T} \) is the same as the minimum distance over all orderings of \( T \) and \( \hat{T}. \) This is incorrect as shown by the example in Figure 2. Assume all edit operations have unit cost. The distance between \( T \) and \( \hat{T} \) would be 4 (representing the cost of deleting the four \( e \)'s in \( T \) and \( \hat{T}. \) However, the least distance between the ordered trees derived from \( T \) and \( \hat{T} \) would be 6 (representing the cost of relabeling.
\(a \in T\) by \(c \in \hat{T}\) and relabeling \(c \in T\) by \(a \in \hat{T}\). The example shows that even after calculating distances for all ordered trees derived from \(T\) and \(\hat{T}\), one may not find the distance between the two trees. Thus, we are forced to consider a different strategy.

Let \(A\) be an unordered tree. Based on an arbitrarily fixed ordering of \(A\), we can number the nodes of \(A\) according to the left-to-right postorder numbering. Let \(A[i]\) be the \(i\)th node in the numbering. We use \(\text{leaves}(A)\) to represent the number of leaves in \(A\), \(\text{par}(n)\) the parent of node \(n\), and \(\text{deg}(n)\) the number of children of \(n\). Define \(\text{Head}(A) = \{n \mid (n \text{ is the root of } A) \lor (\text{deg(par}(n)) > 1)\}\).

For each \(n \in \text{Head}(A)\), let \(s(n)\) represent the string starting from \(n\) and ending at a node that either is the first node having more than one child or is a leaf. (Note such a string may contain a single node.) By construction, if one replaces each string \(s(n)\) in \(A\) by the node \(n\), one would get a tree in which every node \(v\) is branching (i.e., \(\text{deg}(v) > 1\)).

Our approach will be based on finding the best marking of the trees to be compared. Intuitively, a given marking decorates some of the strings in \(T\) and \(\hat{T}\) with marks (that indicate that they will be reduced to a single node). Formally, a marking \(K\) is represented by a pair \((S_T, S_\hat{T})\) where \(S_T\) is the set of unmarked strings in \(T\) and \(S_\hat{T}\) is the set of unmarked strings in \(\hat{T}\). Let \(K(T)\) represent the resulting tree after deleting nodes on the unmarked strings in \(T\). Define the reduced marked tree of \(T\), denoted \(RK(T)\), as the tree obtained by replacing each string \(s(n)\) in \(K(T)\), \(n \in \text{Head}(K(T))\), by the node \(n\). (Note that \(\text{Head}(K(T))\) may be different from \(\text{Head}(T)\); a string in \(K(T)\) may consist of several strings in \(T\).) Let \(\text{dist}(m, n)\) be the distance between nodes \(m \in RK(T)\) and \(n \in RK(\hat{T})\), and let \(\text{edit}(s(m), s(n))\) be the editing distance between the strings \(s(m) \in K(T)\) and \(s(n) \in K(\hat{T})\). We define

\[\text{dist}(m, n) = \text{edit}(s(m), s(n)).\]

A marking \(K\) is legal if \(RK(T)\) is isomorphic to \(RK(\hat{T})\) (denoted \(RK(T) \cong RK(\hat{T})\)). There may be several different isomorphisms between \(RK(T)\) and \(RK(\hat{T})\). Each such isomorphism is called a complete mapping. The cost of a complete mapping \(CM\), denoted \(\text{cost}(CM)\), is defined as

\[\text{cost}(CM) = \sum_{(i,j) \in CM} \text{dist}(RK(T)[i], RK(\hat{T})[j]).\]

The cost of a legal marking \(K = (S_T, S_\hat{T})\) is the cost of the best complete mapping between \(RK(T)\) and \(RK(\hat{T})\) plus the cost of deleting (nodes on) strings in \(S_T\) plus the cost of inserting (nodes on) strings in \(S_\hat{T}\). Our algorithm will be to enumerate all legal markings and find the one with minimum cost.

### 3.1 Cost of a Legal Marking

In finding the best complete mapping between \(RK(T)\) and \(RK(\hat{T})\) for a legal marking \(K\), we proceed level by level in bottom-up fashion. Let \(RK_{tdist}(i, j)\) be the distance between the subtree rooted at \(RK(T)[i]\) and the subtree rooted at \(RK(\hat{T})[j]\). We compute \(RK_{tdist}(i, j)\) where \(RK(T)[i]\) and \(RK(\hat{T})[j]\) are at the same level. In performing the computation, the following two cases are considered:

**Case 1.**  \(\text{deg}(RK(T)[i]) \neq \text{deg}(RK(\hat{T})[j])\). Then, \(RK_{tdist}(i, j) = \infty\), representing the fact that \((i, j)\) can not be in any complete mapping from \(RK(T)\) to \(RK(\hat{T})\).

**Case 2.** \(\text{deg}(RK(T)[i]) = \text{deg}(RK(\hat{T})[j])\). By the ancestor relationship preserving condition, \((i, j) \in CM\) for all complete mappings \(CM\) from \(RK(T)\) to \(RK(\hat{T})\). To find the best mapping between the children of \(RK(T)[i]\) and the children of \(RK(\hat{T})[j]\), we construct a weighted bipartite graph \(BG\) as follows.

Let \(U = \{i_1, \ldots, i_n\}\) and \(V = \{j_1, \ldots, j_n\}\) be the two disjoint sets of nodes from the two trees. Specifically, \(i_h\) and \(j_h, 1 \leq h \leq n\), are the children of \(RK(T)[i]\) and \(RK(\hat{T})[j]\) respectively. Assign the cost for each edge \((u, v)\) where \(u \in U\) and \(v \in V\) based on the formula

\[\text{cost}((u, v)) = \begin{cases} \text{RK}_{tdist}(u, v) & \text{if } \text{deg}(u) = \text{deg}(v) \\ \infty & \text{otherwise} \end{cases}\]

Thus, the problem of determining \(RK_{tdist}(i, j)\) becomes that of finding the optimal complete matching in

---

1. By "the ordered trees derived from \(T\) and \(\hat{T}\)," we mean these ordered trees that are obtained by permuting some siblings in \(T\) or \(\hat{T}\). The distance between two ordered trees \(A\) and \(B\) is the cost of the best ordered mapping \(M_o\) from \(A\) to \(B\), where \(M_o\) only satisfies the map conditions stated in Section 2.2, but preserves the sibling relationship, i.e., for any pair of \((i_1, j_1)\) and \((i_2, j_2)\) in \(M_o\), \(A[i_1]\) is to the left of \(A[i_2]\) iff \(B[j_1]\) is to the left of \(B[j_2]\) [28].

2. Two trees \(A\) and \(B\) are isomorphic if there exists a one-to-one mapping such that every parent-child pair in \(A\) maps to a parent-child pair in \(B\) [8].
We can solve this problem by utilizing Kuhn’s cubic time algorithm [11, 15].

As a consequence of the above analysis, we have the following result.

Theorem 3.1. If \( \text{deg}(RK(T)[i]) \neq \text{deg}(RK(\tilde{T})[j]) \), then \( RK_{\text{tdist}}(i,j) = \infty \); otherwise \( RK_{\text{tdist}}(i,j) = n_{\text{dist}}(RK(T)[i], RK(\tilde{T})[j]) + \sum_{(u,v) \in \text{Ma}} \text{cost}((u,v)) \), where \( \text{Ma} \) is the optimal complete matching in the constructed bipartite graph.

Figure 3 summarizes the algorithm.

\[
\text{for } k := 0 \text{ to height}(RK(T)) \text{ do} \\
\text{for each pair } (i,j) \text{ where } RK(T)[i] \text{ and } RK(\tilde{T})[j] \text{ are at level } k \text{ do} \\
\text{if } \text{deg}(RK(T)[i]) \neq \text{deg}(RK(\tilde{T})[j]) \text{ then} \\
\text{RK}_{\text{tdist}}(i,j) := \infty; \\
\text{else} \\
\text{calculate } RK_{\text{tdist}}(i,j) \text{ as in Theorem 3.1;} \\
\text{end for} \\
\text{end for} \\
\]

Fig. 3. Algorithm Reduced-Marked-Tree-Match.

Theorem 3.2. Algorithm Reduced-Marked-Tree-Match correctly computes the cost of the best complete mapping from \( RK(T) \) to \( RK(\tilde{T}) \) (i.e., the distance between \( RK(T) \) and \( RK(\tilde{T}) \)).

Proof. By Theorem 3.1 and induction on the level \( k \).

Theorem 3.3. The time complexity of the algorithm is \( O(|T| \times |\tilde{T}| + \min(\text{leaves}(T), \text{leaves}(\tilde{T})) \times \text{leaves}(T) \times \text{leaves}(\tilde{T})) \).

Thus, the cost of a legal marking \( K = (S_T, S_{\tilde{T}}) \) can be computed in the same time, and

\[
\text{cost}(K) = \sum_{s \in S_T} s_{\text{dist}}(s, \emptyset) + \sum_{i \in S_{\tilde{T}}} s_{\text{dist}}(\emptyset, i) + RK_{\text{tdist}}(|RK(T)|, |RK(\tilde{T})|).
\]

3.2 The Algorithm

We are now ready to give the exhaustive search algorithm.

find Head(T), Head(\tilde{T}), and strings \( s(u), s(v) \) for \( u \in \text{Head}(T) \) and \( v \in \text{Head}(\tilde{T}) \);

mincost := \infty;

repeat

let \( K \) be a marking;

find Head(K(T)), Head(K(\tilde{T})), and \( s(m), s(n) \) for \( m \in \text{Head}(K(T)) \) and \( n \in \text{Head}(K(\tilde{T})) \);

if \( RK(T) \cong RK(\tilde{T}) \) (i.e., \( K \) is legal) then

if \( \text{cost}(K) < \text{mincost} \) then

\( \text{minK} := K, \text{mincost} := \text{cost}(\text{minK}); \)

until try out all markings;

return(\text{minK}, \text{cost}(\text{minK}));

Fig. 4. Algorithm Exhaustive-Search.

Theorem 3.4. Given a legal marking \( K \) of \( T \) and \( \tilde{T} \), there exists a mapping \( M \) from \( T \) to \( \tilde{T} \) such that \( \gamma(M) = \text{cost}(K) \).

Theorem 3.5. For any mapping \( M \) from \( T \) to \( \tilde{T} \), there exists a legal marking \( K \) of \( T \) and \( \tilde{T} \) such that \( \text{cost}(K) \leq \gamma(M) \).

Theorem 3.6. Algorithm Exhaustive-Search correctly computes \( \delta(T, \tilde{T}) \).

Proof. By Theorem 3.4 and Theorem 2.3, \( \delta(T, \tilde{T}) \leq \text{cost}(\text{minK}). \) By Theorem 3.5, \( \text{cost}(\text{minK}) \leq \gamma(M) \).

Hence \( \text{cost}(\text{minK}) = \delta(T, \tilde{T}), \) and the result follows.

Theorem 3.7. The time complexity of the algorithm is \( O(4^{\text{leaves}(T)} \times \text{leaves}(\tilde{T})) \times \min(\text{leaves}(T), \text{leaves}(\tilde{T})) \times \text{leaves}(T) \times \text{leaves}(\tilde{T})) \) [21], which is still polynomial time. The algorithm runs in exponential time when both \( \text{leaves}(T) \) and \( \text{leaves}(\tilde{T}) \) are proportional to the size of their respective trees.

The following section discusses the heuristic approach we choose for this general case.

4 Heuristic Algorithms

4.1 Terminology

We consider each legal marking as a state and transform the problem of computing \( \delta(T, \tilde{T}) \) to a state space searching problem. Our heuristics perform random walks in the state space via a series of moves. A state \( K' = (S_T, S_{\tilde{T}}) \) is reachable from \( K = (S_T, S_{\tilde{T}}) \) in one move, denoted \( K \Rightarrow K' \), if one of the following (or both) cases hold:
Theorem 4.1. Given a state \( K = (S_T, S_T^+) \), the number of its neighbors is bounded by \( O(\text{leaves}(T) \times \text{leaves}(\hat{T})) \).

The following theorem shows that the state space is connected, i.e., any two states are reachable from each other.

Theorem 4.2. Given two states \( K = (S_T, S_T^+) \) and \( K' = (S_T', S_T'^+) \), there exists a sequence of states \( K_0, K_1, \ldots, K_k \) such that \( K_0 = K, K_k = K' \), and \( K_{i-1} \Rightarrow K_i \) for \( 1 \leq i \leq k \).

A move is called uphill (downhill, respectively) if the cost of the source state is lower (higher, respectively) than the cost of the destination state. A state is a local minimum if in all paths starting at that state any downhill move comes after at least one uphill move. A state is a global minimum if it has the lowest cost among all states.

By Theorem 3.6, \( \delta(T, \hat{T}) \) is the cost of a global minimum in the state space. We will present three approximate algorithms for finding the minimum: Iterative Improvement [13], Simulated Annealing [10] and Two Phase Heuristic [9].

4.2 Iterative Improvement

Iterative Improvement (II) is a greedy algorithm (Figure 5) that has been widely used to solve combinatorial optimization problems [13]. The inner loop of II is called a local optimization. A local optimization starts at a random state and improves the solution by repeatedly accepting random downhill moves until it reaches a local minimum. II repeats these local optimizations, each starting at a new random state, until a stopping condition is met, at which point it returns the local minimum state. (Commonly, a stopping condition is simply a certain number of local optimizations.)

4.3 Simulated Annealing

Simulated Annealing (SA) is a Monte Carlo optimization technique originally proposed by Kirkpatrick, Gelatt and Vecchi for complex problems that involve many degrees of freedom [10]. It has been successfully applied to many areas, including VLSI design [16, 17], pattern recognition [1], combinatorial graph theory [3] and query optimization [9]. In contrast to II, simulated annealing performs not only downhill moves, but uphill moves with some probability, trying to avoid being caught in a high cost local minimum. Figure 6 shows the algorithm.

\[
\begin{align*}
\text{mincost} & := \infty; \\
\text{while not (stopping-condition)} \text{ do} & \\
& \begin{align*}
K & := \text{random state}; \\
& \text{while not (local minimum)} \text{ do} \\
& \begin{align*}
K' & := \text{randomly pick a neighbor state of } K; \\
& \text{if } \text{cost}(K') < \text{cost}(K) \text{ then } K := K'; \\
& \text{end} \\
& \text{if } \text{cost}(K) < \text{mincost} \text{ then} \\
& \text{minK} := K, \text{mincost} := \text{cost}(\text{minK}); \\
& \text{end}; \\
& \text{return}(\text{minK}, \text{cost}(\text{minK})); \\
& \end{align*} \\
& \text{end}; \\
& \text{return}(\text{minK}, \text{cost}(\text{minK})); \\
& \end{align*}
\]

Fig. 5. The iterative improvement algorithm for calculating \( \delta(T, \hat{T}) \).

\[
\begin{align*}
K & := \text{random state}; \\
\text{Temp} & := \text{initial temperature}; \\
\text{minK} & := K; \\
\text{while not (frozen)} \text{ do} & \\
& \begin{align*}
& \text{while not (equilibrium)} \text{ do} \\
& \begin{align*}
K' & := \text{randomly pick a neighbor state of } K; \\
& \Delta(C) := \text{cost}(K') - \text{cost}(K); \\
& \text{if } \Delta(C) < 0 \text{ then } K := K'; \\
& \text{end} \\
& \text{if } \Delta(C) \geq 0 \text{ then} \\
& \text{K} := K' \text{ with probability } e^{-\Delta(C)/\text{Temp}}, \\
& \text{if } \text{cost}(K) < \text{cost}(\text{minK}) \text{ then } \text{minK} := K; \\
& \text{end} \\
\text{Temp} & := \text{reduce}(\text{Temp}); \\
& \text{end} \\
& \text{return}(\text{minK}, \text{cost}(\text{minK})); \\
& \end{align*} \\
& \end{align*}
\]

Fig. 6. The simulated annealing algorithm for calculating \( \delta(T, \hat{T}) \).

The algorithm proceeds in a way that is supposed to be analogous to the process of the annealing of crystals. The inner loop of SA is often called a stage. Each stage is performed under a fixed value of a parameter Temp, called temperature. This parameter controls the probability of accepting uphill moves. The probability is \( e^{-\Delta(C)/\text{Temp}} \), where \( \Delta(C) \) is the difference between the cost of the new state and that of the original one. Thus, the lower the temperature, the smaller the probability of accepting an uphill move. Each stage ends when the algorithm reaches an equilibrium. (We will give our criterion for equilibrium below. Commonly it is simply a certain number of iterations that may or may not be a function of temperature.) After reaching equilibrium at a given temperature, the algorithm reduces the temperature according to some function and another stage begins. Thus, the temperature is lowered as time passes. The algorithm terminates when it is considered to be frozen, i.e., when the temperature is equal to zero. (Romeo and Sangiovanni-Vincentelli show theoretically that, when certain conditions are satisfied,
the algorithm is able to converge to the global minimum [16].

4.4 Two Phase Heuristic

The Two Phase Heuristic (2PH) was originally proposed by Ioannidis for selecting the best execution plans in optimizing complex queries [9]. The algorithm is a combination of both iterative improvement and simulated annealing. It consists of two phases. In phase one, IT is run for a small period of time, i.e., a few local optimisations are performed. The output of that phase, which is the best local minimum found, is the initial state of the next phase. In phase two, SA is run with a low initial temperature. Thus, the algorithm chooses a local minimum and then searches the area around it. Due to the low initial temperature, it is unable to climb up very high hills, though still is able to move in and out of local minimum.

4.5 Sorting Heuristic for Selection of Initial State

The algorithms described in the previous subsections start by randomly picking a state. Sometimes it might be beneficial to start with a good initial guess. We now describe a heuristic for guessing a good initial state.

The heuristic sorts the nodes at each level of $T$ and $\hat{T}$ based on the number of their descendants. Let $T_i$ and $\hat{T}_i$ be the resulting (ordered) trees. Thus, for each pair of nodes $m$, $n$ in $T_i$ (or $\hat{T}_i$), $m$ is a left sibling of $n$ if the number of descendants of $m$ is less than or equal to that of $n$. The distance between the two sorted ordered trees is computed using the optimal algorithm in [28]. Let $M_k$ be the mapping yielding the distance between $T_i$ and $\hat{T}_i$. Let $S_T$ and $S_{\hat{T}}$ be the set of strings in $T_i$ and $\hat{T}_i$, respectively, not touched by any line in $M_k$. Then $(S_T, S_{\hat{T}})$ becomes the initial state for the algorithms described earlier.

4.6 Application Specific Parameters

Several parameters of the hill climbing algorithms are application specific. They can be tuned to improve performance and/or output quality. Below, we first discuss the parameters in IT, and then those in SA and 2PH.

Local minimum

We define a state to be the $r$-local minimum ($r$ stands for relative) if after all its neighbors are tested, none of them has a lower cost. Note that the $r$-local minimum may have neighboring states with the same cost.

Initial temperature

The initial temperature $Temp_0$ has to be high enough so that the system accepts many uphill moves when it starts. The $Temp_0$ used here is equal to $c_T \times cost(K_0)$, where $c_T$ is a constant and $K_0$ is the initial state. This formula was chosen based on Ioannidis' experience [9], though he was working in different domains.

Freezing criterion

Many freezing criteria have been proposed in the literature. Most of them are a combination of tests which verify that the system is at a low temperature and has converged to a satisfactory final state. The freezing criterion used here is a combination of the ones given in [3, 9] and consists of two parts. First, the temperature must be below 1; second, $\min K$ must remain unchanged for 4 stages.

Equilibrium criterion

All the previous work on simulated annealing assumes that every stage consists of a specific number of iterations through the inner loop. We set the number to $epoch_factor \times epoch$ (both the terminology and formulation are from [3, 9]). The $epoch_factor$ is a constant, denoted $c_e$, and $epoch = (\lceil T \rceil + \lceil \hat{T} \rceil)$. $^3$

Choosing the next move

At any point in the execution, the next state is chosen from the set of the current state's neighbors according to some probability $P$. The $P$ chosen here is the same as that in [9], namely each neighbor of the current state has equal probability to be chosen as the next state. I.e.,

$$P(\text{move from } K \text{ to } K') = \begin{cases} \frac{\lceil N(K) \rceil}{\sum_{K' \in N(K)}} & \text{if } K' \text{ is } K's \text{ neighbor} \\ 0 & \text{otherwise} \end{cases}$$

where $\lceil N(K) \rceil$ represents the number of the neighbors of state $K$.

Reducing the temperature $Temp$

There are many cooling schedules developed for the simulated annealing process. The most commonly used one is to reduce the temperature according to the formula $Temp_{\text{new}} = \alpha(\text{Temp}_{\text{old}}) \times \text{Temp}_{\text{old}}$. The function $\alpha$ takes values between 0 and 1. The literature suggests two quite different strategies for computing $\alpha$. Romeo et al. [16] suggested that $\alpha$ range over time (i.e., it depends on $\text{Temp}_{\text{old}}$). It is smaller in the beginning (cooling the system fast), then it rises up to higher values (slowing down the cooling process), and eventually it becomes small again to drive the system down to a minimum without any uphill moves. On the other hand, Aragon et al. [3] suggested that $\alpha$ remain constant at a relatively high value (in the range of 0.9 to 0.95). We give results of experiments with both strategies (Section 5).

Tables 1, 2, 3 summarize the choice of the parameters for IT, SA, 2PH (some of which are self-explanatory).

---

$^3$We have also experimented with different epoch values (e.g., $\lceil T \rceil \times \lceil \hat{T} \rceil$), but found that when $c_e$ is sufficiently large, the performance of SA is insensitive to the parameter, provided its value is greater than the current setting (Section 5).
5 Performance Analysis

The heuristic algorithms proposed in the previous sections were implemented in the C language on a SPARC workstation running UNIX 4.1.1. A series of experiments were carried out to evaluate their effectiveness. The evaluation was based on elapsed CPU time and the quality of the calculation (nearness to exhaustive search). For many of the experiments, the parameters' values in simulated annealing and two phase heuristics were set as follows: \( C_T = 10; \ c_r = 10; \ \alpha = 0.5 \) (cf. Tables 2 & 3). An uphill move is accepted if the corresponding probability is greater than or equal to 0.5 (cf. Figure 6).

A number of industrial parts were collected from the Center for Manufacturing Systems at NJIT, and converted to unordered trees. The conversion is based on the mathematical morphology model presented in [22, 23]. The model employs a grammar of shapes whereby each object is decomposed into primitive shapes. Figure 7 shows how we represent a three-dimensional object, and Figure 8 gives the corresponding tree (see Appendix). Because the composition (shape) operators are commutative, comparing two shapes is an unordered tree comparison problem.

### Table 1. Application specific parameters for II.

<table>
<thead>
<tr>
<th>parameter</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial state ( K_0 )</td>
<td>random</td>
</tr>
<tr>
<td>stopping-condition</td>
<td>equal time to SA</td>
</tr>
<tr>
<td>local minimum</td>
<td>r-local minimum</td>
</tr>
<tr>
<td>next state</td>
<td>random neighbor</td>
</tr>
</tbody>
</table>

### Table 2. Application specific parameters for SA.

<table>
<thead>
<tr>
<th>parameter</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial state ( K_0 )</td>
<td>random</td>
</tr>
<tr>
<td>initial temp ( Temp_0 )</td>
<td>( c_T \times \text{cost}(K_0) )</td>
</tr>
<tr>
<td>frozen</td>
<td>( \text{Temp} &lt; 1 ) and ( \text{minK} ) unchanged for 4 stages</td>
</tr>
<tr>
<td>equilibrium</td>
<td>( c_r \times (</td>
</tr>
<tr>
<td>next state</td>
<td>random neighbor</td>
</tr>
<tr>
<td>temp reduction</td>
<td>( \text{Temp}<em>{\text{new}} = \alpha \times \text{Temp}</em>{\text{old}} )</td>
</tr>
</tbody>
</table>

### Table 3. Application specific parameters for 2PH.

<table>
<thead>
<tr>
<th>parameter</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial state (II phase)</td>
<td>random</td>
</tr>
<tr>
<td>stopping-condition (II phase)</td>
<td>5 local optimizations</td>
</tr>
<tr>
<td>initial state ( K_0 ) (SA phase)</td>
<td>( \text{minK} ) of II phase</td>
</tr>
<tr>
<td>initial temp ( Temp_0 ) (SA phase)</td>
<td>( 3 \times \text{cost}(K_0) )</td>
</tr>
</tbody>
</table>

5.1 Experimental Results

We ran trees obtained from various parts and compared the relative behavior of the heuristic algorithms and their performance relative to optimal values. The sizes of the constructed trees ranged from 1 to 40. Twenty pairs of trees were tested for each algorithm and the average of the distance values produced by the algorithms was plotted. Due to space limitations, we only present the results for algorithms using the sorting heuristic, and omit those not using it, as they lead to similar conclusions.

Figure 9 shows the results for the output quality, and Figure 10 shows those for running time (in seconds). It can be seen that all the heuristics perform well for small trees (those having less than 10 nodes). In running these trees, the enumerative algorithm is even faster than the hill climbing algorithms. When trees become large, the two phase heuristic (SH2PH) outperforms the other heuristics. Note that iterative improvement (SHII) deteriorates significantly as the tree sizes increase. When trees are large, serious plateau problems occur, where there is mostly a flat area separating valleys [27].\(^5\) In such a situation, the local improvement operation may become useless.

---

\(^4\)Each object is considered as a set of points. The shape operators used here include morphological dilation \((\delta)\) [19, 23], set union \((\cup)\), intersection \((\cap)\), and complement \((\sim)\), all of which are commutative.

\(^5\)A state is on a plateau if it has no lower cost neighbor and yet it can reach lower cost states without uphill moves.
When one of the trees has a bounded number of leaves. For the more general case, we have presented several approximate algorithms. All the algorithms perform random walks via a series of moves in the search space. To improve the performance of these algorithms, we have developed a sorting heuristic for guessing an initial match.

When trees are large, the sorting heuristic helps both running time and output quality, while Ioannidis' Two Phase Heuristic works well to find the best distance measurement from the initial match.

The proposed algorithms have been integrated into the tool for tree pattern matching we presented in TAI 91 [26]. Currently we are studying the extension of DAG matching, whose edges may or may not be labeled.

6 Conclusions

This paper has considered the comparison problem between unordered trees. We mentioned the NP-completeness results and presented an efficient enumerative algorithm. The algorithm runs in polynomial time when one of the trees has a bounded number of leaves. For the more general case, we have presented several approximate algorithms. All the algorithms perform random walks via a series of moves in the search space. To improve the performance of these algorithms, we have developed a sorting heuristic for guessing an initial match.

When trees are large, the sorting heuristic helps both running time and output quality, while Ioannidis' Two Phase Heuristic works well to find the best distance measurement from the initial match.

The proposed algorithms have been integrated into the tool for tree pattern matching we presented in TAI 91 [26]. Currently we are studying the extension of DAG matching, whose edges may or may not be labeled.

Acknowledgements

We would like to thank Jun Tian for his assistance in implementing the algorithms presented in this paper. We also thank the anonymous referees, whose excellent comments help improve the paper.

References


Appendix

Fig. 7. The decomposition of a three dimensional object into primitive shapes through a set of rewrite rules. Non-terminal nodes include complicated shapes and shape operators. Terminal nodes include primitive shapes such as lines and unit circles.

Fig. 8. Tree representation of the object in Fig. 7.