Correspondence

Exact and Approximate Algorithms for Unordered Tree Matching

Dennis Shasha, Jason Tsong-Li Wang,
Kaizhong Zhang and Frank Y. Shih

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. In companion work, we have shown this problem to be NP-complete. This paper presents an efficient 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.**

I. INTRODUCTION

Comparing trees has applications in image processing [24], molecular biology [26], natural language processing [20], and other fields of pattern recognition [5], [6], [10], [12], [17], [18]. 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 [9].

For example, various researchers [3], [7] have observed that the secondary structures of RNA (trees) influence translation rates from RNA to proteins. Because different sequences can produce similar secondary structures [8], 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) [6], [28], [31], [36]–[38]. 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. A previous result has shown this problem to be NP-complete, even for binary trees having a label alphabet of size two [39]. This paper is concerned with enumerative and approximate algorithms for this problem.

Section II presents the definitions and basic lemmas concerning tree comparison. Section III shows why one intuitive approach to unordered tree comparison does not work, and presents an efficient enumerative algorithm. Section IV gives a state space model for the heuristic approach we choose, and several approximate algorithms. Section V discusses empirical results. Section VI concludes the paper.

Manuscript received April 24, 1992; revised July 12, 1993. A preliminary version of this paper was presented at the Fourth International Conference on Tools with Artificial Intelligence, Nov. 1992, Arlington, Virginia. The work was supported in part by the National Science Foundation under grants IRI-9109138, IRI-8901699, and CCR-9103953, in part by the Office of Naval Research under grants N00014-90-J-1110, N00014-91-J-1472 and N00014-92-J-1719, in part by the New Jersey Institute of Technology under grant SBR-421280, in part by the Natural Sciences and Engineering Research Council of Canada under grant OGP0046373, and in part by a grant from the AT&T Foundation.

D. Shasha is with the Courant Institute of Mathematical Sciences, New York University, New York, NY 10012 USA.

J. T. L. Wang is with the Department of Computer and Information Science, New Jersey Institute of Technology, University Heights, Newark, NJ 07102.

K. Zhang is with the Department of Computer Science, The University of Western Ontario, London, Ontario, Canada N6A 5B7.

F. Y. Shih is with the Institute for Integrated Systems Research and Center for Manufacturing Systems, New Jersey Institute of Technology, University Heights, Newark, NJ 07102 USA.

IEEE Log Number 9400457.

II. EDITING DISTANCE BETWEEN TREES

A. 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 [32] and ordered trees [36]. 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 Λ or a label of a node in tree T_1 and b is either Λ or a label of a node in tree T_2. We call $a \rightarrow b$ a relabel operation if $a \neq \Lambda$ and $b \neq \Lambda$; a delete operation if $a \neq \Lambda$ and $b = \Lambda$; an insert operation if $a = \Lambda$ and $b \neq \Lambda$. Let T_2 be the tree that results from the application of an edit operation $a \rightarrow b$ to tree T_1; this is written $T_1 \Rightarrow T_2$ via $a \rightarrow b$. Fig. 1 illustrates the edit operations.

Let S be a sequence $e_1, \ldots, e_k$ of edit operations. An S-derivation from tree T to tree $\hat{T}$ is a sequence of trees $T_0, ..., T_k$ such that $T = T_0$, $\hat{T} = T_k$, and $T_{i-1} \Rightarrow T_i$ via e_i for $1 \leq i \leq k$.

Let γ be a cost function which assigns to each edit operation $a \rightarrow b$ a nonnegative real number $\gamma(a \rightarrow b)$. We constrain γ 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 γ to the editing operations sequence S by letting $\gamma(S) = \sum_{i=1}^{|S|} \gamma(e_i)$. Formally the distance between T and $\hat{T}$ is defined as:

$\delta(T, \hat{T}) = \min_S \{\gamma(S) \mid S$ is an edit operation sequence taking T to $\hat{T}\}$.

δ is also a distance metric according to the definition of γ.

B. 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 Fig. 2 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)).

Suppose we have an ordering (e.g., postorder) for nodes in a tree. Let $T[i]$ be the ith node of tree T in the given ordering. Formally, a mapping from T to $\hat{T}$ is a triple $(M, T, \hat{T})$ (or simply M if there is no confusion), where M is any set of pairs of integers (i, j) satisfying the following constraints:

- $1 \leq i \leq |T|$, $1 \leq j \leq |\hat{T}|$, where $|.|$ represents the number of nodes in the indicated tree;

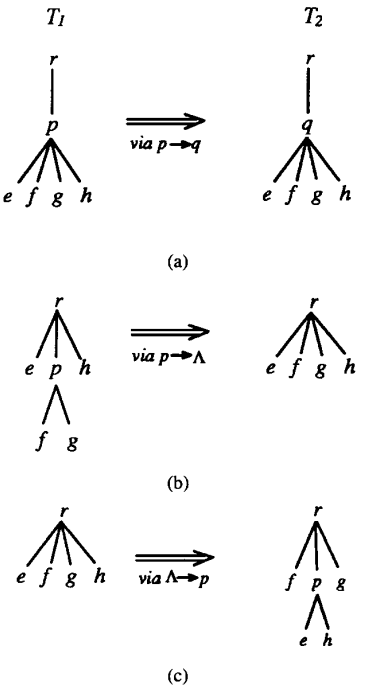

Fig. 1. Examples illustrating the edit operations. (a) Relabeling to change one node label (p) to another (q). (b) Deletion of a node; all children of the deleted node p become children of the parent r. (c) Insertion of a node; a subset of siblings among the children of r (here e, h) become the children of p.

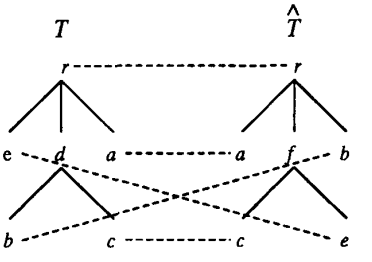

Fig. 2. A mapping from T to $\hat{T}$. A dotted line from a node u in T to a node v in $\hat{T}$ indicates that u should be changed to v if $u \neq v$, or that u remains unchanged if $u = v$. The nodes of T not touched by a dotted line are to be deleted and the nodes of $\hat{T}$ not touched are to be inserted. The mapping shows a way to transform T to $\hat{T}$. The only constraint is that ancestor-descendant relationships be preserved.

- 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).

Thus, the mapping in Fig. 2 is {(1, 3), (2, 5), (3, 2), (5, 1), (6, 6)}.

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:

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

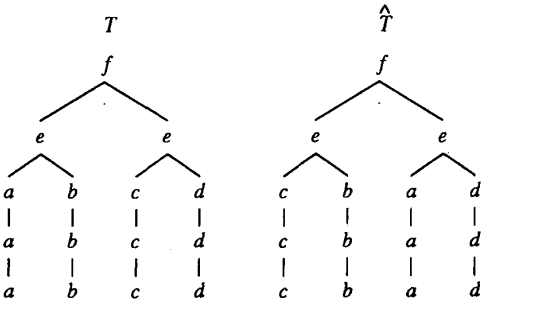

Fig. 3. Example showing that only considering the ordered trees derived from T and $\hat{T}$ can not get $\delta(T, \hat{T})$. Viewed as unordered trees, T and $\hat{T}$ are at distance 4. The best ordering would give a distance of 6.

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

The following lemmas are proved in [39].

Lemma 1: (i) $M \circ M'$ *is a mapping*; (ii) $\gamma(M \circ M') \leq \gamma(M) + \gamma(M')$.

The second lemma says 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.

Lemma 2: Given S, a sequence $e_1, e_2, \ldots, e_k$ 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 $\gamma(S) = \gamma(M)$.

Hence, we obtain

Theorem 1: $\delta(T, \hat{T}) = \min\{\gamma(M) \mid M$ *is a mapping from* T *to* $\hat{T}\}$.

In [39], we have shown that finding $\delta(T, \hat{T})$ is NP-complete. The goal of this paper is to develop enumerative and approximate algorithms for solving the problem.

III. 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 Fig. 3. 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 $a \in \hat{T}$).[1] 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 ith node in the numbering. We use $leaves(A)$ to represent the number of leaves in A, $par(n)$ the

[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 not only satisfies the map conditions stated in Section II, 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 $B[j_2]$ [36].

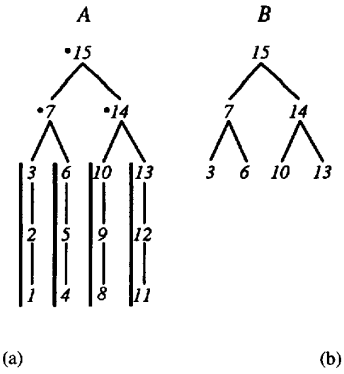

Fig. 4. (a) Each node in tree A is labeled with its postorder number. $Head(A)$ contains nodes with numbers 3, 6, 7, 10, 13, 14, 15. For each node $n \in Head(A)$, its corresponding string $s(n)$ is highlighted either by a bullet or by a boldface line. (Note that the strings corresponding to nodes 7, 14, 15 contain a single node only.) In total, there are seven strings in tree A; (b) Replacing each string $s(n)$ in tree A by its corresponding node n yields tree B. Note that every node in tree B is branching.

parent of node n, and $deg(n)$ the number of children of n. Define

$$Head(A) = \{n \mid (n \text{ is the root of } A) \vee (deg(par(n)) > 1)\}.$$

For each $n \in Head(A)$, 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., $deg(v) > 1$). Fig. 4 illustrates such a case.

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 $K(T)$, $n \in Head(K(T))$, by the node n. (Note that $Head(K(T))$ may be different from $Head(T)$; a string in $K(T)$ may consist of several strings in T.) Let $ndist(m,n)$ be the distance between nodes $m \in RK(T)$ and $n \in RK(\hat{T})$, and let $sdist(s(m),s(n))$ be the editing distance between the strings $s(m) \in K(T)$ and $s(n) \in K(\hat{T})$. We define $ndist(m,n) = sdist(s(m),s(n))$. Fig. 5 (a)(b)(c) illustrate the definitions.

A marking K is *legal* if $RK(T)$is isomorphic to $RK(\hat{T})$ (denoted $RK(T) \cong RK(\hat{T})$).[2] 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 $cost(CM)$, is defined as

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

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.

[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 [2].

A. 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:$deg(RK(T)[i]) \neq deg(RK(\hat{T})[j])$. 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: $deg(RK(T)[i]) = deg(RK(\hat{T})[j])$. 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

$$cost((u,v)) = \begin{cases} RK\_tdist(u,v) & \text{if } deg(u) = deg(v) \\ \infty & \text{otherwise} \end{cases}$$

Fig. 5(d) shows the constructed bipartite graph BG for computing $RK\_tdist(4,4)$ in Fig. 5(c). Thus, the problem of determining $RK\_tdist(i,j)$ becomes that of finding the optimal complete matching in BG. We can solve this problem by utilizing Kuhn's cubic time algorithm [16], [21].

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

Lemma 3: (i) If $deg(RK(T)[i]) \neq deg(RK(\hat{T})[j])$, *then* $RK\_tdist(i,j) = \infty$.

(ii) If $deg(RK(T)[i]) = deg(RK(\hat{T})[j]) = 0$, *then* $RK\_tdist(i,j) = ndist(RK(T)[i], RK(\hat{T})[j])$.

(iii) If $deg(RK(T)[i]) = deg(RK(\hat{T})[j]) \neq 0$, *then*

$$RK\_tdist(i,j) = ndist(RK(T)[i], RK(\hat{T})[j]) + \sum_{(u,v)\in Ma} cost((u,v))$$

where Ma is the optimal complete matching in the constructed bipartite graph.

Fig. 6 summarizes the algorithm.

Theorem 2: Algorithm Reduced_Marked_Tree_Match correctly computes the cost of the best complete mapping from $RK(T)$ to $RK(\hat{T})$ (i.e., the distance between $RK(T)$ and $RK(\hat{T})$).

Proof: By Lemma 3, each $RK\_tdist(i,j)$ where $RK(T)[i]$ and $RK(\hat{T})[j]$ are at the same level is calculated correctly. By induction on the level k, the algorithm computes $RK\_tdist(|RK(T)|, |RK(\hat{T})|)$ correctly. ■

Theorem 3: The time complexity of the algorithm is $O(|T| \times |\hat{T}| + \min(leaves(T), leaves(\hat{T})) \times leaves(T) \times leaves(\hat{T}))$.

Proof: Given a marking K, using Wagner and Fischer's string matching algorithm,we can compute $sdist(s(m),s(n))$ for $s(m) \in K(T)$ and $s(n) \in K(\hat{T})$ (i.e., $ndist(m,n)$ for $m \in RK(T)$ and $n \in RK(\hat{T})$) in $O(|T| \times |\hat{T}|)$ time [32]. We compute $RK\_tdist(m,n)$ in $O(1)$ time if $deg(m) \neq deg(n)$, and in $O((deg(m) + deg(n))^3)$ time otherwise [16], [21]. Let T_e be the time taken by the algorithm. Then, (see top of page 672)

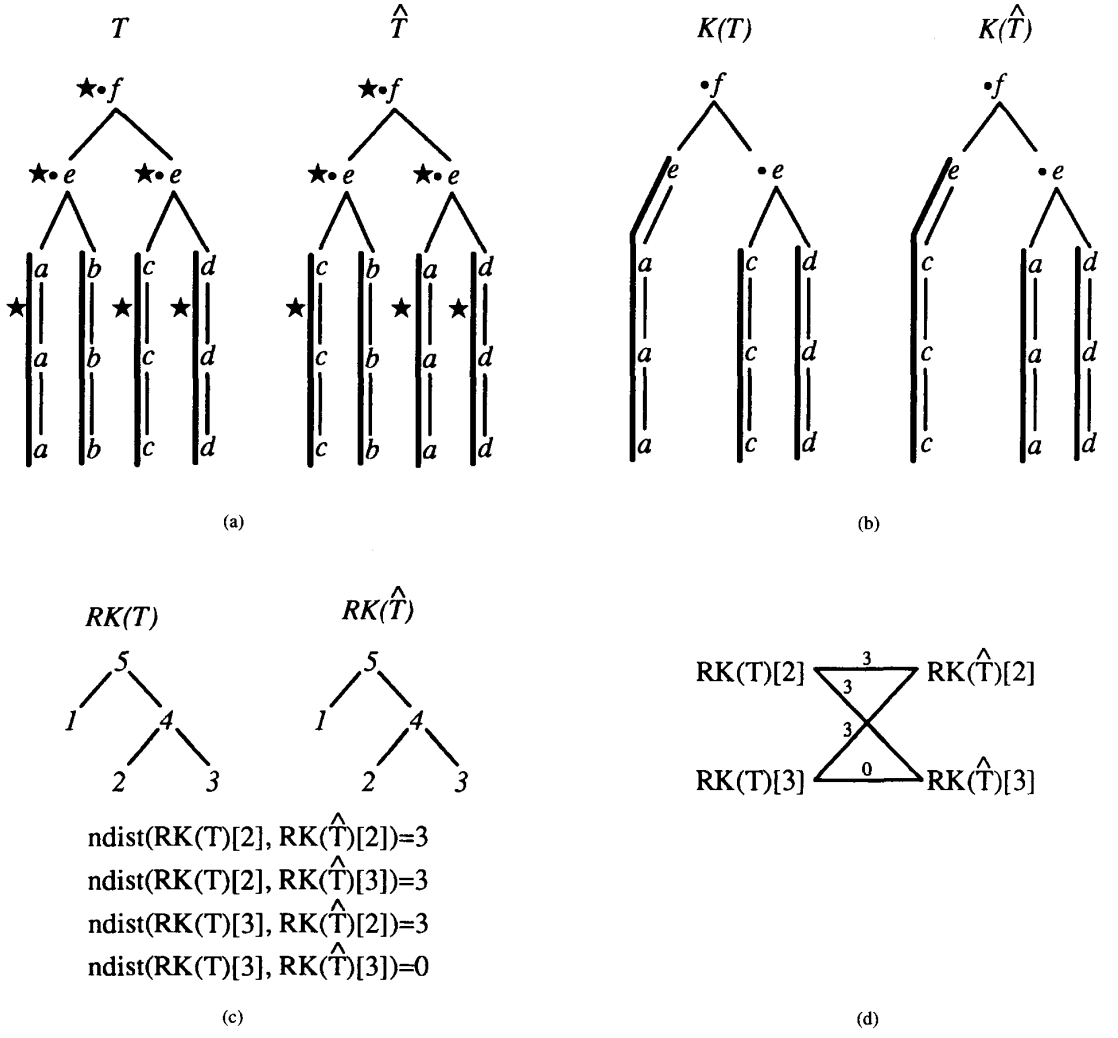

Fig. 5(a) A marking K on trees T and $\hat{T}$. The strings in T and $\hat{T}$ are highlighted either by a bullet or by a boldface line. K decorates the strings in T and $\hat{T}$ with marks $\star$. The unmarked strings in both T and $\hat{T}$ are composed of *bbb*. (b) The trees $K(T)$ and $K(\hat{T})$ obtained by deleting the unmarked string (i.e., the three b's) in T and $\hat{T}$, respectively. The strings in $K(T)$ and $K(\hat{T})$ are highlighted either by a bullet or by a boldface line. Note that a string in $K(T)$ may contain several strings in T. For example, the string composed of $eaaa$ in $K(T)$ consists of two strings e and aaa in T. (c) The reduced marked trees $RK(T)$ and $RK(\hat{T})$ obtained by replacing each string $s(n)$ in $K(T)$ and $K(\hat{T})$, respectively, by its corresponding node n. Nodes in $RK(T)$ and $RK(\hat{T})$ are labeled by their postorder numbers. The distance between nodes $m \in RK(T)$ and $n \in RK(\hat{T})$ is defined to be the distance between their corresponding strings $s(m) \in K(T)$ and $s(n) \in K(\hat{T})$. Assume all edit operations on strings have unit cost. The distances between the leaves of $RK(T)$ and those of $RK(\hat{T})$ are shown in the figure. (d) The weighted bipartite graph BG constructed for computing $RK\_tdist(4,4)$ in (c). The number on each edge represents the cost of that edge. The optimal complete matching in BG gives the best mapping between the children of $RK(T)[4]$ and $RK(\hat{T})[4]$.

Similarly, $T_e \leq O(|T| \times |\hat{T}|) + O(\max_{n \in RK(\hat{T})}\{deg(n)\} \times |RK(T)| \times |RK(\hat{T})|)$. Since (i) $\max_{m \in RK(T)}\{deg(m)\} \leq leaves(T)$ and $\max_{n \in RK(\hat{T})}\{deg(n)\} \leq leaves(\hat{T})$, and (ii) $|RK(T)| \leq 2 \times leaves(T) - 1$ and $|RK(\hat{T})| \leq 2 \times leaves(\hat{T}) - 1$, it follows that T_e is $O(|T| \times |\hat{T}| + \min(leaves(T), leaves(\hat{T})) \times leaves(T) \times leaves(\hat{T}))$. ■

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

$$cost(K) = \sum_{s \in S_T} sdist(s, \emptyset) + \sum_{\hat{s} \in S_{\hat{T}}} sdist(\emptyset, \hat{s}) + RK\_tdist(|RK(T)|, |RK(\hat{T})|).$$

B. The Algorithm

We are now ready to give the exhaustive search algorithm (see Fig. 7).

Lemma 4: Given a legal marking K of T and $\hat{T}$, there exists a mapping M from T to $\hat{T}$ such that $\gamma(M) = cost(K)$.

Proof: Since K is legal, $RK(T) \cong RK(\hat{T})$. Let CP be the best complete mapping between $RK(T)$ and $RK(\hat{T})$. We can expand CP to get a mapping M from $K(T)$ to $K(\hat{T})$. Clearly, M is also a mapping between T and $\hat{T}$, and $\gamma(M) = cost(K)$. ■

Lemma 5: For any mapping M from T to $\hat{T}$, there exists a legal marking K of T and $\hat{T}$ such that $cost(K) \leq \gamma(M)$.

Proof: We decorate strings s in T and $\hat{T}$ with marks if there exists at least one node on s which is touched by lines in M, and

$$T_e \le O(|T| \times |\hat{T}|) + \sum_{m \in RK(T)} \sum_{n \in RK(\hat{T})} \begin{cases} 1 & \text{if } deg(m) \ne deg(n) \\ 2^3 \times deg(m) \times deg(m) \times deg(n) & \text{otherwise} \end{cases}$$

$$\le O(|T| \times |\hat{T}|) + \max_{m \in RK(T)} \{deg(m)\} \times \sum_{m \in RK(T)} \sum_{n \in RK(\hat{T})} \begin{cases} 1 & \text{if } deg(m) \ne deg(n) \\ 2^3 \times deg(m) \times deg(n) & \text{otherwise} \end{cases}$$

$$\le O(|T| \times |\hat{T}|) + \max_{m \in RK(T)} \{deg(m)\} \times 2^3 \times \sum_{m \in RK(T)} \sum_{n \in RK(\hat{T})} (deg(m)+1) \times (deg(n)+1)$$

$$\le O(|T| \times |\hat{T}|) + O(\max_{m \in RK(T)} \{deg(m)\} \times |RK(T)| \times |RK(\hat{T})|)$$

```
/* Note since RK(T) ≅ RK(T̂), height(RK(T)) = height(RK(T̂)). */
for k := 0 to height(RK(T)) do
    for each pair (i, j) where RK(T)[i] and RK(T̂)[j] are at level k do
        if deg(RK(T)[i]) ≠ deg(RK(T̂)[j]) then
            RK_tdist(i, j) := ∞;
        else
            calculate RK_tdist(i, j) as in (ii) or (iii) of Lemma 3;
    end for
end for
```

Fig. 6. Algorithm Reduced_Marked_Tree_Match.

delete those strings not touched by any line in M. We will show that this produces the desired marking.

Let us call the marking K. Since (i) for any string s in $K(T)$ or $K(\hat{T})$, there is at least one node on s touched by lines in M, and (ii) mappings preserve the ancestor relationship, we know that for any pairs (u_1, v_1) and (u_2, v_2) in M, u_1 and u_2 are on the same string in $K(T)$ if and only if v_1 and v_2 are on the same string in $K(\hat{T})$. Thus, M establishes an isomorphism between $RK(T)$ and $RK(\hat{T})$. So, K is legal.

Since $cost(K)$ is obtained from the best mapping between strings in $K(T)$ and $K(\hat{T})$ [32], and the best isomorphism between $RK(T)$ and $RK(\hat{T})$, it follows that $cost(K) \le \gamma(M)$. ■

Theorem 4: Algorithm Exhaustive_Search correctly computes $\delta(T, \hat{T})$.

Proof: By Theorem 1 and Lemma 4, $\delta(T, \hat{T}) \le cost(minK)$. By Lemma 5, $cost(minK) \le \delta(T, \hat{T})$. Hence $cost(minK) = \delta(T, \hat{T})$, and the result follows. ■

Theorem 5: The time complexity of the algorithm is $O(4^{leaves(T)} \times 4^{leaves(\hat{T})} \times \min(leaves(T), leaves(\hat{T})) \times |T| \times |\hat{T}|)$.

Proof: Given a tree T, finding $Head(T)$ and the corresponding strings takes $O(|T|)$ time. For a marking K, constructing $K(T)$ and $RK(T)$ takes $O(|T|)$ time. Using the algorithm in [2], we can test if $RK(T) \cong RK(\hat{T})$ in $O(|RK(T)| + |RK(\hat{T})|)$ time. By Theorem 3 and the fact that there are at most $2^{2 \times leaves(T)} \times 2^{2 \times leaves(\hat{T})}$ markings, the total time required by the algorithm is $O(2^{2 \times leaves(T)} \times 2^{2 \times leaves(\hat{T})} \times (|T| \times |\hat{T}| + \min(leaves(T), leaves(\hat{T})) \times leaves(T) \times leaves(\hat{T}))) = O(4^{leaves(T)} \times 4^{leaves(\hat{T})} \times \min(leaves(T), leaves(\hat{T})) \times |T| \times |\hat{T}|)$. ■

Remark: If both $leaves(T)$ and $leaves(\hat{T})$ are bounded by pre-determined constants, algorithm Exhaustive_Search runs in polynomial time (with those constants as parameters, of course). Suppose only one of the trees, say T, has a bounded number of leaves. Then, the complexity becomes $O(4^{leaves(T)} \times leaves(\hat{T})^{2 \times leaves(T)} \times leaves(T) \times |T| \times |\hat{T}|)$ [27], which is still polynomial time. The algorithm runs in exponential time when both $leaves(T)$ and $leaves(\hat{T})$ are proportional to the size of their respective trees.

```
1.  find Head(T), Head(T̂), and strings s(u), s(v)
    for u ∈ Head(T) and v ∈ Head(T̂);
2.  mincost := ∞;
3.  repeat
4.      let K be a marking;
5.      find Head(K(T)), Head(K(T̂)), and strings s(m), s(n)
        for m ∈ Head(K(T)) and n ∈ Head(K(T̂));
6.      if RK(T) ≅ RK(T̂) (i.e., K is legal) then
7.          if cost(K) < mincost then
8.              minK := K;
9.              mincost := cost(minK);
10. until try out all markings;
11. return(minK, cost(minK));
```

Fig. 7. Algorithm Exhaustive_Search.

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

IV. HEURISTIC ALGORITHMS

A. Terminology

We consider each legal marking as a state and transform the problem of computing $\delta(T, \hat{T})$ to a state space searching problem. Our heuristics perform *random walks* in the state space via a series of *moves*. A state $K' = (S_T', S_{\hat{T}}')$ is reachable from $K = (S_T, S_{\hat{T}})$ in one move, denoted $K \Rightarrow K'$, if one of the following (or both) cases hold: (i) S_T' is obtained by removing a string from (or adding a string into) S_T; (ii) $S_{\hat{T}}'$ is obtained by removing a string from (or adding a string into) $S_{\hat{T}}$. (Recall that S_T ($S_{\hat{T}}$, respectively) is the set of unmarked strings (i.e., those to be deleted) in T ($\hat{T}$, respectively).) We shall call K' a neighbor of K.

Lemma 6: Given a state $K = (S_T, S_{\hat{T}})$, *the number of its neighbors is bounded by* $O(leaves(T) \times leaves(\hat{T}))$.

Proof: Any string in S_T (resp. $S_{\hat{T}}$) could be removed, and each string in T (resp. $\hat{T}$) could potentially be added to S_T (resp. $S_{\hat{T}}$). Thus, the total number of neighbors is bounded by $O(|Head(T)| + |Head(\hat{T})| + |Head(T)| \times |Head(\hat{T})|)$. Since $|Head(T)| \le 2 \times leaves(T)$ and $|Head(\hat{T})| \le 2 \times leaves(\hat{T})$, the result follows. ■

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

Lemma 7: Given two states $K = (S_T, S_{\hat{T}})$ *and* $K' = (S_T', S_{\hat{T}}')$, *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* $1 \le i \le k$.

Proof: Let CP be the best complete mapping between $RK(T)$ and $RK(\hat{T})$, and let $(m, n) \in CP$. Each node in $RK(T)$ ($RK(\hat{T})$, respectively) is a string in $K(T)$ ($K(\hat{T})$, respectively), which may contain several marked strings in T ($\hat{T}$, respectively). We add the

```
mincost := ∞;
while not (stopping_condition) do
    begin
        K := random state;
        while not (local_minimum) do
            begin
                K' := randomly pick a neighbor state of K;
                if cost(K') < cost(K) then K := K';
            end
        if cost(K) < mincost then
            minK := K;
            mincost := cost(minK);
    end;
return(minK, cost(minK));
```

Fig. 8. The iterative improvement algorithm for calculating $\delta(T, \hat{T})$.

```
K := random state;
Temp := initial temperature;
minK := K;
while not (frozen) do
    begin
        while not (equilibrium) do
            begin
                K' := randomly pick a neighbor state of K;
                Δ(C) := cost(K') − cost(K);
                if Δ(C) < 0 then K := K';
                if Δ(C) ≥ 0 then K := K' with probability e^{−Δ(C)/Temp};
                if cost(K) < cost(minK) then minK := K;
            end
        Temp := reduce(Temp);
    end
return(minK, cost(minK));
```

Fig. 9. The simulated annealing algorithm for calculating $\delta(T, \hat{T})$.

marked strings into S_T (i.e., remove their marks) one at a time until there is only one string s left that corresponds to $m \in RK(T)$. Similarly, add the marked strings into $S_{\hat{T}}$ until only one string $\hat{s}$ is left that corresponds to $n \in RK(\hat{T})$. Then, add s into S_T and $\hat{s}$ into $S_{\hat{T}}$ (i.e., delete the nodes m from $RK(T)$ and n from $RK(\hat{T})$) simultaneously. Starting with leaves of $RK(T)$ and $RK(\hat{T})$, we perform such string addings in bottom-up fashion until no node is left in $RK(T)$ and $RK(\hat{T})$.

Next, let CP' be the best complete mapping between $RK'(T)$ and $RK'(\hat{T})$, and let $(m', n') \in CP'$. We insert the nodes $m' \in RK'(T)$ and $n' \in RK'(\hat{T})$ by marking one of their corresponding strings in T and $\hat{T}$ (i.e., removing the strings from S'_T and $S'_{\hat{T}}$) simultaneously. Then, mark the rest of the corresponding strings in T, and the rest of the corresponding strings in $\hat{T}$. Starting with the roots of $RK'(T)$ and $RK'(\hat{T})$, we perform such string removals in top-down fashion until all nodes in $RK'(T)$ and $RK'(\hat{T})$ are inserted.

Since each string adding and string removal yield a neighbor of the previous state, the above produces the desired sequence of states. ■

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 4, $\delta(T, \hat{T})$ is the cost of a global minimum in the state space. We will present three approximate algorithms for finding the minimum: Iterative Improvement [19], Simulated Annealing [15] and Two Phase Heuristic [14].

B. Iterative Improvement

Iterative Improvement (II) is a greedy algorithm (Fig. 8) that has been widely used to solve combinatorial optimization problems [19]. 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.)

C. 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 [15]. It has been successfully applied to many areas, including VLSI design [22], [23], pattern recognition [1], combinatoric graph theory [4] and query optimization [14]. 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. Fig. 9 shows the algorithm.

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 $e^{-\Delta(C)/Temp}$, 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 [22].)

D. Two Phase Heuristic

The *Two Phase Heuristic* (2PH) was originally proposed by Ioannidis for selecting the best execution plans in optimizing complex queries [14]. The algorithm is a combination of both iterative improvement and simulated annealing. It consists of two phases. In phase one, II is run for a small period of time, i.e., a few local optimizations 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.

E. 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_s and $\hat{T}_s$ be the resulting (ordered) trees. Thus, for each pair of nodes m, n in T_s (or $\hat{T}_s$), m

is a left sibling of n iff 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 [36]. Let M_o be the mapping yielding the distance between T_s and $\hat{T}_s$. Let S_T and $S_{\hat{T}}$ be the set of strings in T_s and $\hat{T}_s$, respectively, not touched by any line in M_o. Then $(S_T, S_{\hat{T}})$ becomes the initial state for the algorithms described earlier.

F. 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 II, 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's experience [14], 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 [4], [14] and consists of two parts. First, the temperature must be below 1; second, $minK$ 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 [4], [14]). The *epoch_factor* is a constant, denoted c_e, and *epoch* = $(|T| + |\hat{T}|)$.[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 [14], namely each neighbor of the current state has equal probability to be chosen as the next state. I.e.,

$$P(\text{moving from } K \text{ to } K') = \begin{cases} \frac{1}{|N(K)|} & \text{if } K' \text{ is a neighbor of } K \\ 0 & \text{otherwise} \end{cases}$$

where $|N(K)|$ 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 $Temp_{new} = \alpha(Temp_{old}) \times Temp_{old}$. The function α takes values between 0 and 1. The literature suggests two quite different strategies for computing α. Romeo et al. [22] suggested that α range over time (i.e., it depends on $Temp_{old}$). 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. [4] suggested that α remain constant at a relatively high value (in the range of 0.5 to 0.95). We give results of experiments with both strategies (Section V).

Tables I, II, III summarize the choice of the parameters for II, SA, 2PH (some of which are self-explanatory).

[3] We have also experimented with different epoch values (e.g., $|T| \times |\hat{T}|$), 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 V).

TABLE I
APPLICATION SPECIFIC PARAMETERS FOR II.

parameter	value
initial state K_0	random
stopping_condition	equal time to SA
local_minimum	r-local minimum
next state	random neighbor

TABLE II
APPLICATION SPECIFIC PARAMETERS FOR SA.

parameter	value
initial state K_0	random
initial temperature $Temp_0$	$c_T \times cost(K_0)$
frozen	$Temp < 1$ and $minK$ unchanged for 4 stages
equilibrium	$c_e \times (\|T\| + \|\hat{T}\|)$
next state	random neighbor
temperature reduction	$Temp_{new} = \alpha \times Temp_{old}$

TABLE III
APPLICATION SPECIFIC PARAMETERS FOR 2PH.

parameter	value
initial state (II phase)	random
stopping_condition (II phase)	5 local optimizations
initial state K_0 (SA phase)	$minK$ of II phase
initial temperature $Temp_0$ (SA phase)	$3 \times cost(K_0)$

TABLE IV
SUMMARY OF HEURISTICS.

Abbreviation	Heuristic
II	Iterative Improvement
SHII	Sorting Heuristic + Iterative Improvement
SA	Simulated Annealing
SHSA	Sorting Heuristic + Simulated Annealing
2PH	Two Phase Heuristic
SH2PH	Sorting Heuristic + Two Phase Heuristic
SH	Sorting Heuristic

V. 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 performance. The evaluation was based on elapsed CPU time and the quality of the calculation (nearness to exhaustive search). Table IV summarizes the algorithms, and provides their abbreviations which we will use when referring to them. For many of the experiments, the parameters' values in simulated annealing and two phase heuristics were set as follows: c_T = 10; c_e = 10; α = 0.5 (cf. Tables II & III). An uphill move is accepted if the corresponding probability is greater than or equal to 0.5 (cf. Fig. 9).

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 [29], [30]. The model employs a grammar of shapes whereby each object is decomposed into primitive shapes. Each object is considered as a set of points. The shape operators used here include morphological dilation ($\oplus$) [25], [30], set union ($\cup$), intersection ($\cap$), and complement ($\sim$), all of which are commutative.

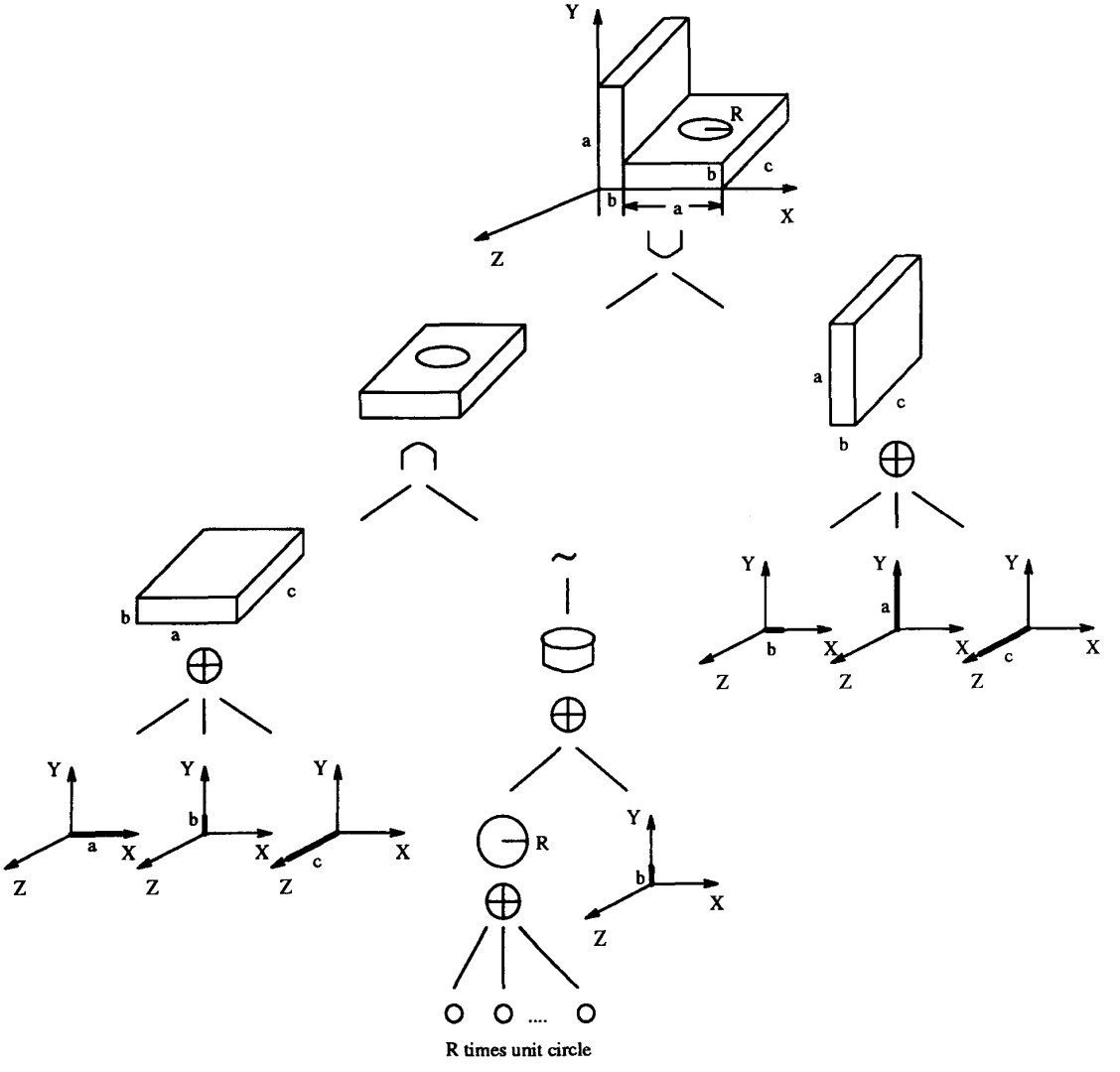

Fig. 10. The decomposition of a three dimensional object into primitive shapes through a set of rewrite rules [11], [13], [29]. Non-terminal nodes include complicated shapes and shape operators. Terminal nodes include primitive shapes such as lines and unit circles.

Fig. 10 shows how we represent a three dimensional object, and Fig. 11 gives the corresponding tree. Because the composition (shape) operators are commutative, comparing two shapes is an unordered tree comparison problem.

A. Experimental Results

In the first set of experiments, we examine the heuristic performance relative to optimal values obtained from exhaustive search. To keep the experiments manageable, the tree sizes are restricted to the range $[M, N]$, where M ranges from 1 to 10, and N ranges from 10 to 20, respectively. All edit operations are assumed to have unit cost.

Table V summarizes the results. Each mean in the table represents the average value of the relative distances obtained by applying an algorithm to 20 pairs of objects (trees). A relative distance is the quotient of the distance value obtained by the algorithm divided by the optimum. The time represents the average CPU time (in seconds) used by an algorithm.

TABLE V
HEURISTIC PERFORMANCE/OPTIMAL PERFORMANCE (THE AVERAGE CPU TIME USED BY EXHAUSTIVE SEARCH IS 7 WHEN $M = 1$, $N = 10$, AND 1.9×10^3 WHEN $M = 10$, $N = 20$).

	M = 1, N = 10			M = 10, N = 20		
Heuristic	Mean	Variance	Time	Mean	Variance	Time
II	1.021	0.00129	17	1.175	0.03294	202
SHII	1.020	0.00134	15	1.171	0.03182	162
SA	1.019	0.00114	18	1.101	0.01352	199
SHSA	1.015	0.00118	15	1.095	0.01243	160
2PH	1.012	0.00119	10	1.092	0.00853	193
SH2PH	1.012	0.00082	9	1.085	0.00755	155
SH	1.086	0.01135	1.1	1.38	0.05893	1.9

It can be seen that all the heuristics perform well for small trees (those having less than 10 nodes). In running these trees,

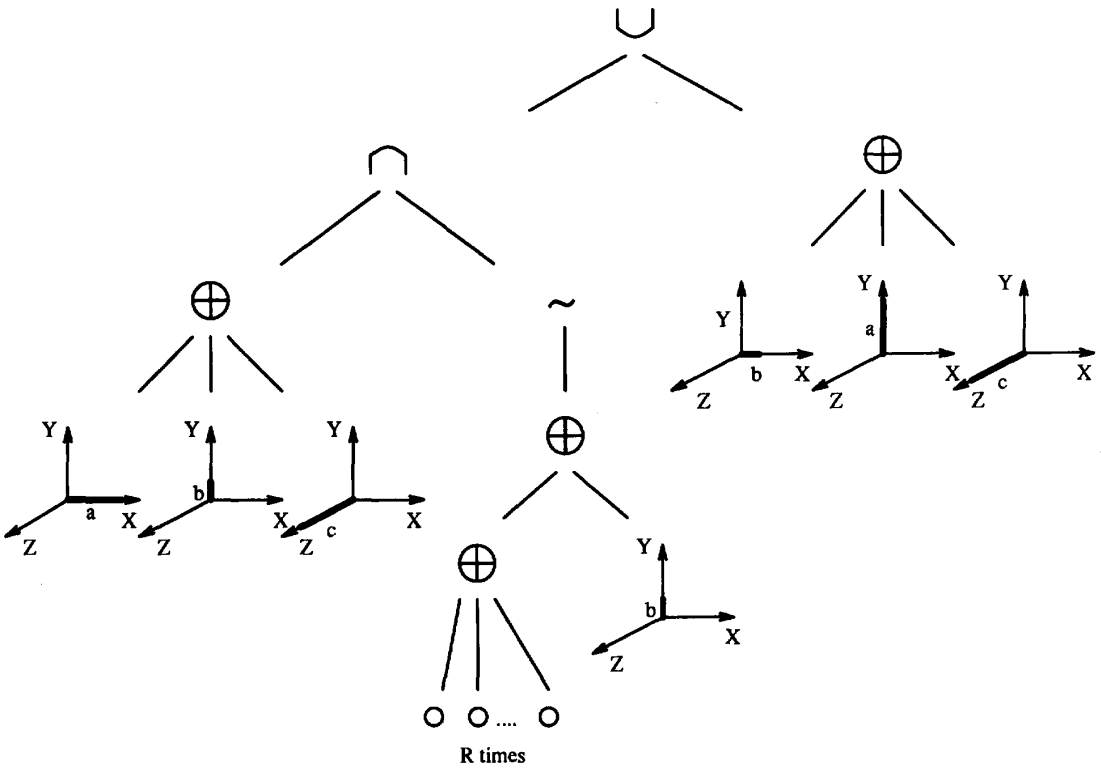

Fig. 11. Tree representation of the object in Fig. 10.

the enumerative algorithm is even faster than the hill climbing algorithms. When trees become large, two phase heuristics are better than the other heuristics, and exhibit consistent behavior (with small variances). Having a good initial guess improves both the quality of the output and the running time of the algorithms. Among the heuristics we presented here, SH runs fastest. Note also that the variances of the heuristic are very high; its performance is sensitive to the topology of the given trees. When two trees are nearly isomorphic, the heuristic performs well. On the other hand, when the trees' topologies differ substantially, it may have a poor performance.[4]

In the second set of experiments, we ran trees obtained from more complex objects and compared the relative behavior of the heuristics in terms of both output quality and running time. 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. We only present the results for algorithms using SH, and omit those not using it, as they lead to similar conclusions.

Fig. 12 shows the results for the output quality, and Fig. 13 shows those for running times. It is apparent that as the trees become large, SH2PH outperforms the other heuristics. Note that 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 [35].[5] In such a situation, the local improvement operation may become useless.

[4] We also considered a heuristic based on bipartite matching techniques for guessing an initial match. The performance of the heuristic is comparable to SH [27], but is much more complicated, so we omit a discussion of it here.

[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.

We have also run simulated annealing and two phase heuristics, varying values for the parameters c_T, c_e and α (cf. Tables II & III). The performance of the heuristics did not appear to be sensitive to the values of these parameters once their values are sufficiently large. Large settings slightly improve the output quality, but suffer from long running time. For extremely small parameter values (e.g., $c_T = 1$, $c_e = 1$, $\alpha = 0.1$), the algorithms may behave poorly. No significant difference in quality was observed between the cooling schedules proposed by Romeo and Aragon (cf. Section IV), though the latter generally runs a bit slower.

In summary, when trees are small, one should use the enumerative algorithm. When trees are large, if the running time is crucial, one should use SH. If the output quality is important, one should use the two phase heuristic combined with SH.[6]

VI. CONCLUSIONS

This paper has considered the comparison problem between unordered trees.We first 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

[6] We also evaluated the effectiveness of applying this heuristic to classify the industrial parts. Twenty samples were used in the experiment. In classifying an industrial part, the sample best matching (i.e., with the minimum distance to) that part was considered to be in the same class of the part. It was observed that 19 out of 20 of these were correct. The classification error was encountered when very similarly-shaped objects were present.

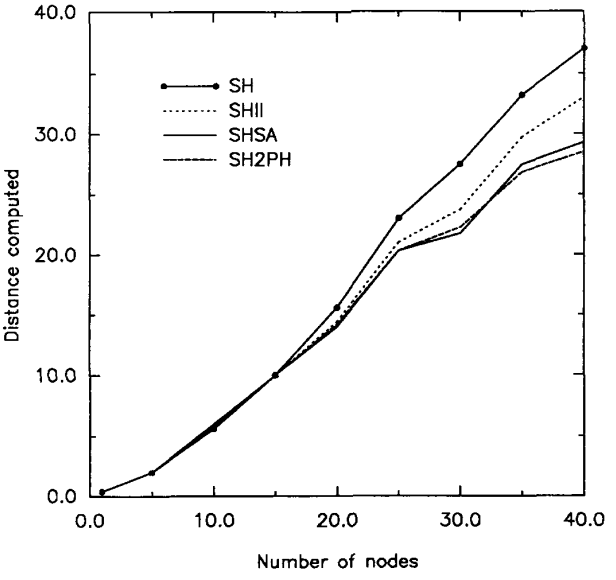

Fig. 12. Output quality of heuristics as a function of tree size (c_T = 10, c_e = 10, α = 0.5). Heuristic A is better than heuristic B if A finds a lower distance mapping than B.

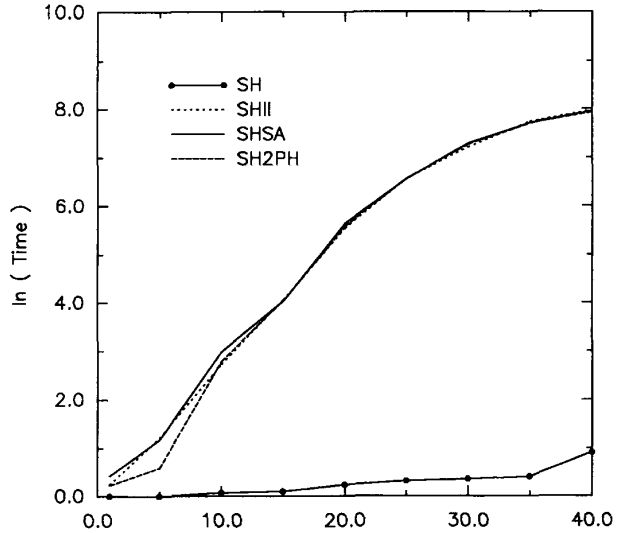

Fig. 13. CPU time of heuristics as a function of tree size (c_T = 10, c_e = 10, α = 0.5).

the performance of these algorithms, we have developed a soring heuristic for guessing an initial match.

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

The work presented here is part of a project to develop a comprehensive toolbox for tree pattern matching [33], [34]. The proposed algorithms and their implementation have been integrated into this toolbox.

ACKNOWLEDGMENT

We would like to thank Jun Tian for his assistance in implementing some of the algorithms. We also thank the anonymous referees, whose comments helped to improve the presentation of this paper.

REFERENCES

[1] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A learning algorithm for boltzmann machines," *Cognitive Sci.*, vol. 9, pp. 147–169, 1985.
[2] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Reading, MA: Addison Wesley, 1974.
[3] S. Alluvia, H. Locker-Giladi, S. Koby, O. Ben-Nun, and A. B. Oppenheim, "RNase III stimulates the translations of the cIII gene of bacteriophage lambda," in *Proc. Nat. Acad. Sci. U.S.A.*, 1987, pp. 1–5.
[4] C. R. Aragon, D. S. Johnson, L. A. Megeoch, and C. Schevon, "Optimization by simulated annealing: An experimental evaluation," Unpublished manuscript, Oct. 1984.
[5] J. M. Brayer and K. S. Fu, "A note on the k-tail method of tree grammar inference," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-5, pp. 409–422, 1975.
[6] Y. C. Cheng and S. Y. Lu, "Waveform correlation by tree matching," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-7, pp. 299–305, May 1985.
[7] I. C. Deckman and D. E. Draper, "S4-alpha mRNA translation regulation complex," *J. Mol. Biol.*, vol. 196, pp. 323–332, 1987.
[8] N. Delihas and J. Anderson, "Generalized structures of 5s ribonsomal RNA's," *Nucleic Acid Res.*, vol. 10, pp. 7323, 1982.
[9] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*. New York: Wiley, 1973.
[10] R. W. Ehrich and J. P. Foith, "Representation of random waveforms by relational trees," *IEEE Trans. Comput.*, vol. C-25, pp. 725–736, 1976.
[11] K. S. Fu, *Syntactic Pattern Recognition and Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1982.
[12] K. S. Fu and B. K. Bhargava, "Tree systems for syntactic pattern recognition," *IEEE Trans. Comput.*, vol. C-22, pp. 1087–1099, Dec. 1973.
[13] K. S. Fu and T. L. Booth, "Grammatical inference: Introduction and survey —Part I," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-5, pp. 95–111, 1975.
[14] Y. E. Ioannidis and Y. C. Kang, "Randomized algorithms for optimizing large join queries," in *Proc. ACM-SIGMOD Int. Conf. Management Data*, 1990, pp. 312–321.
[15] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, May 1983.
[16] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Res. Logistics Quarterly*, vol. 2, pp. 83–97, 1955.
[17] S. Y. Lu, "A tree-matching algorithm based on node splitting and merging," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-6, pp. 249–256, Mar. 1984.
[18] B. Moayer and K. S. Fu, "A tree system approach for fingerprint pattern recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-8, pp. 376–387, May 1986.
[19] S. Nahar, S. Sahni, and E. Shragowitz, "Simulated annealing and combinatorial optimization," in *Proc. 23rd Design Automation Conf.*, 1986, pp. 293–299.
[20] M. Neff and B. K. Boguraev, "Dictionaries, dictionary grammars and dictionary entry parsing," in *Proc. 27th Annual Meeting Asso. Computational Linguistics*, June 1989.
[21] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization Algorithms and Complexity*. Englewood Cliffs, NJ: Prentice-Hall, 1982.
[22] F. Romeo and A. Sangiovanni-Vincentelli, "Probabilistic hill climbing algorithms: Properties and applications," in *Proc. 1985 IEEE Conf. on VLSI*, 1985, pp. 393–417.
[23] F. Romeo, A. Sangiovanni-Vincentelli, and C. Sechen, "Research on simulated annealing at Berkeley," in *Proc. 1984 IEEE Int. Conf. on Computer Design*, Oct. 1984, pp. 652–657.
[24] H. Samet, "Distance transform for images represented by quadtrees," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-4, pp. 298–303, May 1982.
[25] J. Serra, *Image Analysis and Mathematical Morphology*. New York: Academic Press, 1982.
[26] B. A. Shapiro and K. Zhang, "Comparing multiple RNA secondary structures using tree comparisons," *Comput. Appl. Biosci.*, vol. 6, pp. 309–318, 1990.

[27] D. Shasha, J. T. L. Wang, K. Zhang, and F. Y. Shih, "Exact and approximate algorithms for unordered tree matching," Tech. Rep. CIS-92-07, Dept. Computer and Information Science, New Jersey Institute of Technology, 1992.
[28] D. Shasha and K. Zhang, "Fast algorithms for the unit cost editing distance between trees," *J. Algorithms*, vol. 11, pp. 581–621, 1990.
[29] F. Y. Shih, "Object representation and recognition using mathematical morphology model," *J. Systems Integration*, vol. 1, pp. 235–256, Aug. 1991.
[30] F. Y. Shih and O. R. Mitchell, "Threshold decomposition of grayscale morphology into binary morphology," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-11, pp. 31–42, Jan. 1989.
[31] K.-C. Tai, "The tree-to-tree correction problem," *J. ACM*, vol. 26, pp. 422–433, 1979.
[32] R. A. Wagner and M. J. Fischer, "The string-to-string correction problem," *J. ACM*, vol. 21, pp. 168–173, Jan. 1974.
[33] J. T. L. Wang, K. Zhang, K. Jeong, and D. Shasha, "A tool for tree pattern matching," in *Proc. 3rd IEEE Int. Conf. Tools for Artificial Intelligence*, San Jose, CA, Nov. 1991, pp. 436–444.
[34] J. T. L. Wang, K. Zhang, K. Jeong, and D. Shasha, "A system for approximate tree matching," to appear in *IEEE Trans. Knowledge Data Eng.*, 1994.
[35] P. H. Winston, *Artificial Intelligence*. Reading, MA: Addison-Wesley, 1984.
[36] K. Zhang and D. Shasha, "Simple fast algorithms for the editing distance between trees and related problems," *SIAM J. Comput.*, vol. 18, pp. 1245–1262, Dec. 1989.
[37] K. Zhang, D. Shasha, and J. T. L. Wang, "Fast serial and parallel algorithms for approximate tree matching with VLDC's," in *Proc. 3rd Annual Symp. Combinatorial Pattern Matching*, Tucson, Arizona, April 1992, pp. 148–158.
[38] K. Zhang, D. Shasha, and J. T. L. Wang, "Approximate tree matching in the presence of variable length don't cares," *J. Algorithms*, vol. 16, pp. 33–66, 1994.
[39] K. Zhang, R. Statman, and D. Shasha, "On the editing distance between unordered labeled trees," *Information Processing Letters*, vol. 42, pp. 133–139, 1992.

A Recurrent Neural Net Approach to One-Step Ahead Control Problems

Percy P. C. Yip and Yoh-Han Pao

***Abstract*—In this paper, we present a recurrent neural net technique to provide control actions for nonlinear dynamic systems. In most current neural net control approaches, two nets are usually required. One acts as a system emulator, and the other one is a controller network. Rather than using two nets, our system requires only one net which is the system emulator. In our proposed system, a neural net is used to learn the forward dynamics of the system, and the control signal is evolved from the output of the same net with use of an equation of motion. There is no need to learn the control law from another neural net, such as a system inverse net. The use of the proposed algorithm is illustrated with an example.**

I. Introduction

Control is concerned with the manipulation of the inputs so that the system evolves in a desired manner. To accomplish this task, a control system needs to have access to a body of input/output data for the plant being controlled so that it can specify the input/output characteristics of the controller. Fig. 1 shows a block diagram of a control system encompassing the controller and the plant being controlled. In its basic mode it functions as a one-step ahead controller or generally as a k-step ahead controller. During the past few decades many different adaptive schemes to accomplish this task have been proposed [1]–[4]. These adaptive control schemes can be classified as being of the *direct control or indirect control* methods. In a direct control system, the parameters of the controllers are adaptively updated according to the input and output data without explicitly knowing the model of the plant as shown in Fig. 2(a). The indirect control method, on the other hand, learns the model of the plant, and the control action is computed using the estimated parameters of the model as illustrated in Fig. 2(b).

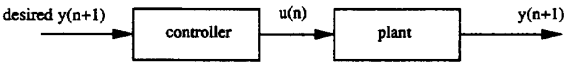

Fig. 1. Block diagram of one-step ahead controller.

Recently artificial neural networks have been considered for the intelligent control of nonlinear systems because of their learning and nonlinear modeling capabilities [5]–[13]. If a neural net is shown a representative collection of system input/output associated pairs in the training phase, the network can learn the nonlinear mapping between the inputs and outputs.

Consider a discrete nonlinear dynamic system described by (1), the control action from a one-step ahead controller at time step k should be generated so that the output of the system at time $k+1$ equals to the desired output at time $k+1$. That means that a controller should generate a $u(k)$ so that $y(k+1) = y_d(k+1)$.

$$y(k+1) = f[y(k), y(k-1), \cdots y(k-n), u(k), u(k-1), u(k-2), \cdots, u(k-m)] \quad (1)$$

where

$y(k)$ is the output of the system at time step k,

$u(k)$ is the input of the system at time step k.

Chen [6] proposed a neural network approach for nonlinear self-tuning adaptive control under the assumption that the system can be modeled as (2) and so that the control can be calculated using equation (3).

$$y(k+1) = f[y(k), y(k-1), \cdots y(k-n), u(k-1), u(k-2), \cdots, u(k-m)] + g[y(k), y(k-1), \cdots, y(k-n), u(k-1), u(k-2), \cdots, u(k-m)]u(k) \quad (2)$$

$$u(k) = -g^{-1}()f() + g^{-1}()y_d(k+1). \quad (3)$$

Two sub-nets learn the functions f and g which describe the behavior of the plant. The control action is then computed by (3). At time step k, $f()$ and $g()$ are known from the outputs of the neural nets, and so $g^{-1}()$ can be computed. That method does not seem to be extendible to multi-input/multi-output processes. One problem is the question of the existence of the inverse of g in (3). Another problem is the computational complexity of finding the matrix inverse for multi-input/multi-output processes.

In the event the *a priori* structure of a dynamic system is not known, some other researchers [7]–[9] have proposed the use of an inverse net to generate the control actions. Referring to Fig. 3, we see

Manuscript received June 8, 1992, revised December 16, 1992 and July 12, 1993.

The authors are with the Department of Electrical Engineering and Applied Physics, Case Western Reserve University, Cleveland, OH 44106, USA.

IEEE Log Number 9400451.