Finding Approximate Patterns in Undirected Acyclic Graphs

Jason T. L. Wang* Kaizhong Zhang† George Chang‡

Dennis Shasha§

Abstract

We consider an approximate pattern matching problem for undirected acyclic graphs. Specifically, let P be a pattern graph, D a data graph and t an integer. We present an algorithm to locate a subgraph in D whose distance from P is at most t. The distance measure used here is the degree-2 metric published previously. The time complexity of our algorithm is $O(N_P N_D d\sqrt{d}\log d)$ where N_P and N_D are the number of nodes in P and D, respectively; $d = \min\{d_P, d_D\}$; d_P and d_D are the maximum degree of P and D, respectively. Central to our algorithm is a procedure for finding approximate patterns in rooted unordered trees and freely allowing cuts. We discuss two applications of the algorithms in chemical information search and website management on the Internet.

Keywords: Distance metric; Graph matching; Pattern discovery; Website analysis; Chemical substructure search

1 Introduction

Pattern matching in graphs finds many applications in biochemistry, computer vision, and web data management. Pattern matching requires a distance metric. In the past a number of metrics have been proposed and various algorithms for graph matching have been studied [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]. Here we are interested in an approximate pattern matching

*Department of Computer and Information Science, New Jersey Institute of Technology, University Heights, Newark, NJ 07102, USA. Corresponding author, Tel.: 001-973-596-3396; Fax: 001-973-596-5777; Email: jason@cis.njit.edu.

†Department of Computer Science, The University of Western Ontario, London, Ontario, Canada N6A 5B7.

‡Department of Mathematics, Kean University, Union, NJ 07083, USA.

§Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, NY 10012, USA.

problem whose instance is specified by giving a pattern object P, a data object D and an integer t. The question is whether D contains a sub-object D' that resembles P in the sense that the distance from P to D' is at most t. Ukkonen studied this problem for strings [12]; Zhang and Shasha extended it to ordered trees [13]. This paper addresses the problem for undirected acyclic graphs (also known as unrooted, unordered trees or free trees).[1]

Our motivation comes from the study of information retrieval for biochemical and molecular databases [14, 15, 16]. In such systems, a common operation is to perform a substructure search in a database of graphs [17, 18], i.e., to find the graphs in the database that contain a given pattern. Since the subgraph isomorphism problem is NP-complete [19], these systems employ various heuristics (e.g. genetic algorithms) [20, 21, 22] to conduct the search. Here we extend the operation to allow a mismatch to exist when comparing a data graph with the pattern. This extension is useful as the scientific data are often obtained experimentally and error rates of up to 5% are not uncommon with some of the currently used techniques [23]. Our work also contributes to graph-based hypothesis testing where an expert draws a model of a pattern he or she would expect to find in a database, given his or her understanding of the domain. Searching for the pattern can help to confirm or deny the original hypothesis and inspire new ones. Since the patterns to be searched are frequently averaged from several similar models or are the result of an "educated guess" on the part of the expert, approximate matching becomes important in practice.

The distance metric we adopt in this paper is the degree-2 edit distance for graphs previously proposed in [24]. In the next section we review the distance metric and introduce notation and terminology used in the paper. Section 3 presents our algorithm for finding approximate patterns in graphs based on the degree-2 metric. Section 4 discusses applications. Section 5 concludes the paper.

[1] In the paper we will refer to the undirected acyclic graphs simply as "graphs" when the context is clear.

2 Preliminaries

2.1 Degree-2 Edit Operations

The graphs we consider here are undirected, acyclic and labeled (i.e., these graphs do not contain cycles and each node in the graphs has a label).[2] There are three types of edit operations: relabeling, deleting and inserting a node. Relabeling node n means changing the label on n. Deleting a node n means selecting a node n' that is a neighbor of n and making the other neighbors of n be the neighbors of n', and then removing n. This amounts to the contraction of the edge between n and n' [25] and making the resulting node have the label n'. Insert is the complement of delete. This means that inserting n as a neighbor of n' makes a subset of the current neighbors of n' become the neighbors of n (and cease to be the neighbors of n') where the subset is again determined by the algorithm. We represent an edit operation as $u \to v$. We call $u \to v$ a relabeling operation if $u \neq \lambda$ and $v \neq \lambda$; a delete operation if $v = \lambda$; and an insert operation if $u = \lambda$. As in [14, 18], we assign an integer cost (or weight) to each edit operation $u \to v$, denoted $\gamma(u \to v)$. The edit distance between two graphs G and G' is the minimum weighted number of edit operations needed to transform G to G'. In general, finding the edit distance is NP-complete [24].

In view of the hardness of the problem, Zhang *et al.* [24] proposed a degree-2 constraint on the edit operations by requiring that any node to be inserted (deleted) have no more than two neighbors.[3] Intuitively one can delete either a leaf or a node n with two neighbors; in the latter case, after deleting n, we simply connect its two neighbors together. When inserting n between two nodes n' and n'', we remove the edge between n' and n'' and make n the neighbor of both n' and n''. These constrained edits are referred to as the degree-2 edit operations. The degree-2 distance between graph G and graph G', denoted $\delta(G, G')$, is the minimum weighted degree-2 edit operations needed to transform G to G'. This metric is a natural extension of the edit distance for strings [26] and Selkow's [27] distance for trees. As shown in Section 4, this metric is suitable for measuring the difference of two website or chemical representations. [24] presented an algorithm for computing $\delta(G, G')$ with the time complexity being $O(N_G N_{G'} d\sqrt{d}\log d)$ where N_G and $N_{G'}$ are the number of nodes in

[2]In practice, the edges of a graph may also have labels. In this case, one can transform a labeled edge between two nodes u and v to a labeled node connecting u and v.

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

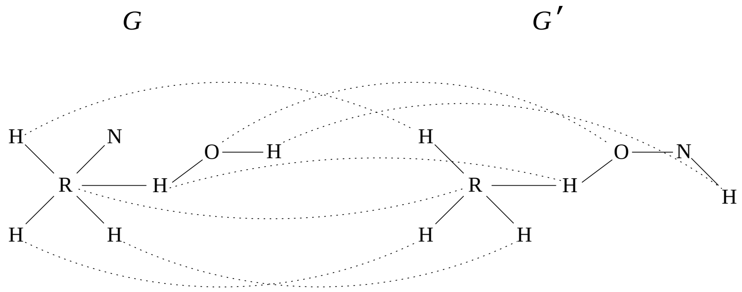


Fig. 1. A mapping from graph G to graph G'. Nodes in G not touched by a mapping line are to be deleted; nodes in G' not touched by a mapping line are to be inserted. The mapping shows a way to transform G to G'.

Fig. 2. Illustrations of the center of three nodes u, v, w, where the center is represented by the bullet $\bullet$.

G and G', respectively; $d = \min\{d_G, d_{G'}\}$; d_G and $d_{G'}$ are the maximum degree of G and G', respectively. In [10], Tanaka proposed a related but different metric for measuring the distance of two graphs and gave an algorithm to compute the metric. The algorithm runs in time $O(N_G N_{G'} q^3)$ where $q = \max\{d_G, d_{G'}\}$.

2.2 Mappings

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

To formalize the notion of mappings, we need some definitions. Let u, v, w be three nodes in a graph G; let $[u, v]$ denote the path between node u and node v inclusively. Define the *center* of the three nodes u, v, w, denoted $center(u, v, w)$, to be the intersection node of the three paths $[u, v]$, $[v, w]$ and $[w, u]$. Figure 2 illustrates the definition.

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

[**Mapping Conditions for Graphs**]

(1) $1 \leq i \leq |G|$, $1 \leq j \leq |G'|$.

(2) For any two pairs (i_1, j_1) and (i_2, j_2) in M, $i_1 = i_2$ iff $j_1 = j_2$ (one-to-one).

(3) For any three pairs (i_1, j_1), (i_2, j_2) and (i_3, j_3) in M, (i^*, j^*) is also in M where $g[i^*]$ $= center(g[i_1], g[i_2], g[i_3])$ and $g'[j^*] = center(g'[j_1], g'[j_2], g'[j_3])$ (center relationship preservation).

The cost of M, denoted $\gamma(M)$, is the cost of deleting nodes of G not touched by a mapping line plus the cost of inserting nodes of G' not touched by a mapping line plus the cost of relabeling nodes in those pairs related by mapping lines with different labels. It can be shown [24] that $\delta(G, G') = \min\{\gamma(M) | M$ is a mapping from G to $G'\}$. Thus, for example, consider Fig. 1 again. The mapping in the figure is a minimum cost mapping and $\delta(G, G') = 2$.

2.3 Cut Operations

Given a pattern graph P and a data graph D, we allow P to match only a part of D, i.e. subgraphs can be freely removed from D. Formally, cutting at an edge e from D means removing e, thus dividing D into two subgraphs D_1 and D_2 (Fig. 3). Let K be a set of edges of D. Let u be a node in D. We use $Cut(D_u, K)$ to represent the subgraph D_u containing u and resulted from cutting at all edges in K from D. Define $\delta(P, Cut(D, K))$ $= \min_{u \in D}\{\delta(P, Cut(D_u, K))\}$. Let $Edges(D)$ be the set of all possible edge sets in D. Define $\delta_{cut}(P, D) = \min_{K \in Edges(D)}\{\delta(P, Cut(D, K))\}$.

The problem of approximate pattern matching in graphs can now be stated as follows: Given a pattern graph P and a data graph D, find $\delta_{cut}(P, D)$. Intuitively we calculate the minimum distance between P and D, allowing any number of cuts at the edges in D at no cost. Call the remaining data graph D'. Then in the best mapping from P to D', there must exist a node u in P that is mapped to a node v in D'. Supposing we assign u, v as the roots of P and D' respectively, we obtain two rooted unordered trees.

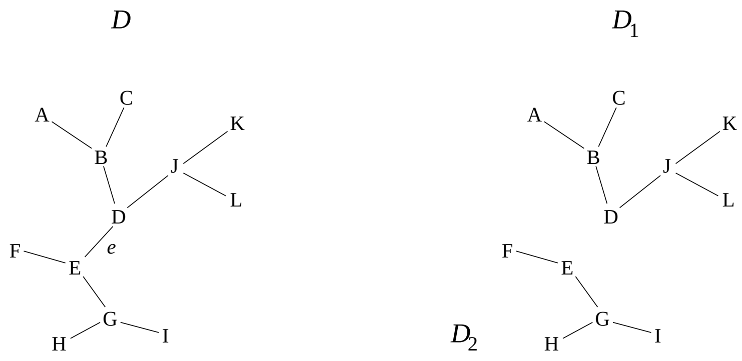


Fig. 3. Cutting at an edge e in data graph D to get D_1 and D_2.

(A rooted unordered tree is a tree having a root, and in the tree the order among siblings is insignificant.) Central to our algorithm for graphs is an algorithm that finds the best mapping and best cuttings in two rooted unordered trees.

3 Approximate Pattern Matching Algorithms

3.1 Algorithm for Rooted Unordered Trees

In this subsection, P (D, respectively) represents a rooted unordered pattern tree (data tree, respectively). Let $p[i]$ ($d[i]$, respectively) denote the ith node of pattern tree P (data tree D, respectively) according to the depth-first search order. $P[i]$ ($D[j]$, respectively) represents the subtree rooted at the node $p[i]$ ($d[j]$, respectively). $PF[i]$ ($DF[j]$, respectively) represents the forest obtained by deleting $p[i]$ ($d[j]$, respectively) from $P[i]$ ($D[j]$, respectively). We use $p[i_1], p[i_2], \ldots, p[i_{m_i}]$ to represent the children of $p[i]$ in $P[i]$ and use $d[j_1], d[j_2], \ldots, d[j_{n_j}]$ to represent the children of $d[j]$ in $D[j]$.

When applied to the rooted unordered trees P and D, the degree-2 constraint on the edit operations becomes the following: a node can be deleted (inserted) only when it has fewer than 2 children. The mapping between two rooted unordered trees P and D is the same as defined in § 2.2 except that condition (3) in that definition is replaced by the following:

[Mapping Conditions for Trees]

for any two pairs (i_1, j_1) and (i_2, j_2) in M, (i^*, j^*) is also in M where $p[i^*] = lca(p[i_1],$

$p[i_2]$), $d[j^*] = lca(d[j_1], d[j_2])$ and $lca(.)$ represents the least common ancestor of the indicated nodes.

Proposition 1. *Let r_P be the root of the unordered pattern tree P and let r_D be the root of the unordered data tree D. Let M be a mapping as defined in § 2.2. When applied to the rooted unordered trees P and D, M satisfies the mapping conditions for trees described above.*

Proof. The result follows by observing that for any two pairs (i_1, j_1) and (i_2, j_2) in M, $lca(p[i_1], p[i_2]) = center(p[i_1], p[i_2], r_P)$ and $lca(d[j_1], d[j_2]) = center(d[j_1], d[j_2], r_D)$. □

As a consequence, one can develop a dynamic programming algorithm for rooted unordered trees. We present several properties, which will be the basis of the algorithm.

Proposition 2. *For all $1 \le i \le |P|$ and $1 \le j \le |D|$,*

(i) $\delta_{cut}(\emptyset, \emptyset) = 0$;

(ii) $\delta_{cut}(\emptyset, D[j]) = 0$;

(iii) $\delta_{cut}(\emptyset, DF[j]) = 0$;

(iv) $\delta_{cut}(P[i], \emptyset) = \delta_{cut}(PF[i], \emptyset) + \gamma(p[i] \to \lambda)$;

(v) $\delta_{cut}(PF[i], \emptyset) = \sum_{1 \le s \le m_i} \delta_{cut}(P[i_s], \emptyset)$.

Proof. Immediate from definitions. □

Proposition 3. *For all $1 \le i \le |P|$ and $1 \le j \le |D|$,*

$$\delta_{cut}(P[i], D[j]) = \mathbf{min} \begin{cases} \gamma(\lambda \to d[j]) + \min_{1 \le t \le n_j} \{\delta_{cut}(P[i], D[j_t])\} \\ \delta_{cut}(P[i], \emptyset) + \min_{1 \le s \le m_i} \{\delta_{cut}(P[i_s], D[j]) - \delta_{cut}(P[i_s], \emptyset)\} \\ \delta_{cut}(PF[i], DF[j]) + \gamma(p[i] \to d[j]) \end{cases}$$

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

Case 1. $i \in M$ and $j \notin M$. Thus, $d[j]$ must be inserted. Furthermore, $P[i]$ must be mapped to $D[j_t]$ for some t and all other $D[j_k]$, $1 \le k \le n_j$, $k \ne t$, must be cut. The value of t ranges from 1 to n_j, and therefore we take the minimum of the corresponding costs.

Case 2. $i \notin M$ and $j \in M$. Let (z, j) be in M. Thus $p[z]$ must be a node in $PF[i]$. Let $p[i_s]$ be the child of $p[i]$ on the path from $p[z]$ to $p[i]$ (see Fig. 4). Thus, $\delta_{cut}(P[i], D[j]) = \delta_{cut}(P[i_s], D[j]) + \delta_{cut}(P[i_1], \emptyset) + \ldots + \delta_{cut}(P[i_{s-1}], \emptyset) + \delta_{cut}(P[i_{s+1}], \emptyset) + \ldots + \delta_{cut}(P[i_{m_i}], \emptyset) + \gamma(p[i] \to \Lambda)$. Since $\delta_{cut}(P[i], \emptyset) = \gamma(p[i] \to \Lambda) + \sum_{k=1}^{m_i} \delta_{cut}(P[i_k], \emptyset)$, we can rewrite the right hand side of the formula as $\delta_{cut}(P[i], \emptyset) + \delta_{cut}(P[i_s], D[j]) - \delta_{cut}(P[i_s], \emptyset)$. The range of s is from 1 to m_i; therefore we take the minimum of the corresponding costs.

Case 3. $i \in M$ and $j \in M$. By the mapping conditions, (i, j) must be in M. Thus $\delta_{cut}(P[i], D[j]) = \delta_{cut}(PF[i], DF[j]) + \gamma(p[i] \to d[j])$.

Case 4. $i \notin M$ and $j \notin M$. This means the entire $D[j]$ is cut. Thus we have to delete every node in $P[i]$, whose cost is greater than the cost in Case 1. Therefore we do not need to include this case in our fourmula. □

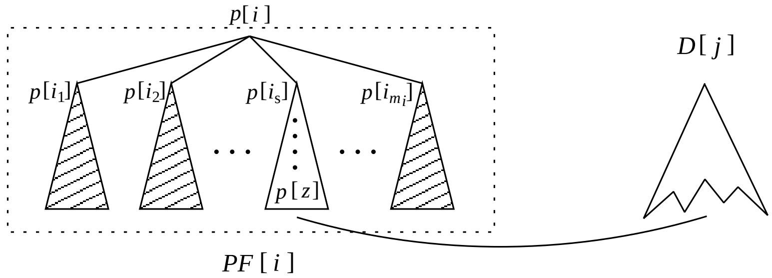


Fig. 4. Illustration of Case 2 of Proposition 3; the shaded areas are mapped to empty trees.

In calculating $\delta_{cut}(PF[i], DF[j])$, we first make the following observation: Supposing $(i, j) \in M$, if two nodes u_1 and u_2 of $PF[i_s]$ are in M for some $1 \leq s \leq m_i$, then there must exist an integer t, $1 \leq t \leq n_j$, such that the two nodes connected to u_1 and u_2, respectively, by the mapping lines of M are in $DF[j_t]$. Thus, a subtree rooted at some child of $p[i]$ must be mapped to a subtree rooted at some child of $d[j]$. We try to find a best mapping among the children of $p[i]$ and the children of $d[j]$ by constructing a weighted bipartite graph BG as follows. Let $U = \{p[i_1], \ldots, p[i_{m_i}]\}$ and $V = \{d[j_1], \ldots, d[j_{n_j}]\}$. Assign the weight to each edge $(p[i_s], d[j_t])$, denoted $\omega((p[i_s], d[j_t]))$, $1 \leq s \leq m_i$ and $1 \leq t \leq n_j$, based on the formula

$$\omega((p[i_s], d[j_t])) = \delta_{cut}(P[i_s], \emptyset) - \delta_{cut}(P[i_s], D[j_t]).$$

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

Proposition 4.

$$\delta_{cut}(PF[i], DF[j]) = \sum_{s=1}^{m_i} \delta_{cut}(P[i_s], \emptyset) - \sum_{(u,v) \in Ma} \omega((u, v)).$$

Thus the problem of calculating $\delta_{cut}(PF[i], DF[j])$ becomes that of finding the maximum weighted matching in BG. One can solve the problem by using Gabow and Tarjan's algorithm [28]. Figure 5 summarizes our algorithm.

Algorithm T

Input: Rooted unordered pattern tree P and data tree D.

Output: $\delta_{cut}(P[i], D[j])$ where $1 \le i \le N_P$ and $1 \le j \le N_D$;

$\delta_{cut}(P[N_P], D[N_D]) = \delta_{cut}(P, D)$.

$\delta_{cut}(\emptyset, \emptyset) := 0$;
for $i := 1$ **to** N_P **do**
  compute $\delta_{cut}(P[i], \emptyset)$ and $\delta_{cut}(PF[i], \emptyset)$ as in Proposition 2 (iv) (v);
for $j := 1$ **to** N_D **do**
  compute $\delta_{cut}(\emptyset, D[j])$ and $\delta_{cut}(\emptyset, DF[j])$ as in Proposition 2 (ii) (iii);
for $i := 1$ **to** N_P **do**
  for $j := 1$ **to** N_D **do begin**
    compute $\delta_{cut}(PF[i], DF[j])$ as in Proposition 4;
    compute $\delta_{cut}(P[i], D[j])$ as in Proposition 3;
  end;

Fig. 5. Algorithm for computing $\delta_{cut}(P, D)$ for two rooted unordered trees P and D.

Proposition 5. *The time complexity of Algorithm T is* $O(N_P N_D \sqrt{d} \log d)$.

Proof. By Proposition 3, the complexity of computing $\delta_{cut}(P[i], D[j])$ is bounded by $O(m_i + n_j)$. In constructing BG for calculating $\delta_{cut}(PF[i], DF[j])$, for each node $u \in U$, it takes $O(n_j)$ time to calculate the weights of the edges touching u and to pick the m_i edges with the maximum weights. Thus, it takes a total of $O(m_i n_j)$ time to construct BG. Let C be the number of nodes in BG and let E be the number of edges in BG. The complexity of finding the maximum weighted matching in BG is $O(\sqrt{C}E\log(CW))$ where W is the maximum weight [28]. Without loss of generality, assume $m_i \leq n_j$. Then C is at most $m_i + m_i m_i$ and E is at most $m_i m_i$. Suppose the node label alphabet for rooted unordered trees is finite. Then the maximum edit cost is finite. As a consequence, the maximum weight W in BG is bounded by cC for some constant c. Thus, the complexity is bounded by $O(m_i n_j + \sqrt{m_i + n_j} m_i m_i \log(m_i + m_i^2)) = O(m_i n_j \sqrt{\min\{m_i, n_j\}} \log(\min\{m_i, n_j\}))$. Therefore, the complexity of Algorithm T is $\sum_{i=1}^{N_P} \sum_{j=1}^{N_D} O(m_i n_j \sqrt{\min\{m_i, n_j\}} \log(\min\{m_i, n_j\})) \leq O(N_P N_D \sqrt{\min\{d_P, d_D\}} \log(\min\{d_P, d_D\})) = O(N_P N_D \sqrt{d} \log d)$. $\square$

3.2 Algorithm for Undirected Acyclic Graphs

By definition, if (i, j) is in a minimum cost mapping M from pattern graph P to data graph D, then we can assign $p[i]$ to be the root of P and assign $d[j]$ to be the root of D, resulting in two rooted unordered trees. From Proposition 1, M is a mapping (as defined in § 3.1) between the two rooted unordered trees. On the other hand, given any mapping M between the two rooted unordered trees, M is also a mapping (as defined in § 2.2) between pattern graph P and data graph D. Therefore by applying Algorithm T to the two rooted unordered trees, we can find $\delta_{cut}(P, D)$. Naively trying every possible pair of trees results in an algorithm that runs in time $O(N_P^2 N_D^2 \sqrt{\min\{d_P, d_D\}} \log(\min\{d_P, d_D\}))$.

A more careful analysis leads to a faster algorithm, referred to as Algorithm G. Let us choose an arbitrary node, say r, in pattern graph P and assign r to be the root of P. Thus the pattern graph P can be viewed as a rooted unordered tree, denoted P_r. For any node u in P, we use P_r^u to represent the unordered subtree rooted at u in P_r. Let M be a minimum cost mapping from pattern graph P to data graph D. There are two cases to be considered: in the rooted P, (i) there exists a node x such that x is touched by a line of M and all nodes touched by lines of M are in the tree P_r^x; (ii) there exist two nodes x_1 and x_2, where neither is an ancestor of the other, such that both x_1 and x_2 are touched by lines of M and all nodes touched by lines of M are either in the subtree $P_r^{x_1}$ or in the subtree $P_r^{x_2}$. Why is two the maximum number? Assume that there exist more than two nodes with these properties. Consider any three such nodes, say x_1, x_2, x_3, where neither node is an ancestor of the other. By the mapping conditions for graphs defined in § 2.2, the center c of x_1, x_2, x_3 must be touched by a line of M. By the definition of centers, c must be an ancestor of x_1, x_2, x_3. This contradicts the fact that all nodes touched by lines of M are in the subtrees rooted at the nodes x_i. Therefore we only consider the two cases.

Case 1. In this case, we choose an arbitrary node v in data graph D and assign v to be the root of D. Call the resulting tree D_v. Compute $\delta_{cut}(P_r^r, D_v^v)$ using Algorithm T and keep all the intermediate results. Let the neighbors of v be $y_1, y_2, \ldots, y_n$. Then compute $\delta_{cut}(P_r^u, D_{y_j}^v)$ for all $u \in P_r$ and for $1 \leq j \leq n$. Next we assign y_1 to be the new root of data graph D and repeat the above computation. The order in which we assign the nodes in data graph D to be its root is based on the preorder traversal of D_v. This order is chosen to minimize the needed computation.

Case 2. Let y_1, y_2 be in data graph D such that $(x_1, y_1) \in M$ and $(x_2, y_2) \in M$. Note that y_1 is a neighbor of y_2, because otherwise there would be a node v on the path between y_1 and y_2 and we would then construct a lower cost mapping by mapping r to v. Also note that no matter which node is assigned to be the root of D, we have either that y_1 is the parent of y_2 or y_2 is the parent of y_1. Neither of the two situations is covered in Case 1. We split D at the edge (y_1, y_2) into two rooted unordered trees $D_{y_2}^{y_1}$ and $D_{y_1}^{y_2}$. Each of $P_r^{x_1}$, $P_r^{x_2}$, $D_{y_2}^{y_1}$, $D_{y_1}^{y_2}$ is a rooted unordered tree (see Fig. 6). The restriction of M on $P_r^{x_1}$ and $D_{y_2}^{y_1}$ and on $P_r^{x_2}$ and $D_{y_1}^{y_2}$ gives mappings between rooted unordered trees and therefore can be obtained during the computation of Case 1. Since we don't know at which edge to split, we have to try all the edges, one by one, in D.

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

Proposition 6. *The time complexity of Algorithm G is $O(N_P N_D d\sqrt{d}\log d)$.*

Proof. The computation of $\delta_{cut}(P_r^u, D_{y_j}^v)$ for all $u \in P_r$ and for $1 \leq j \leq n$ in Case 1 is dominated by the computation of $\delta_{cut}(PF_r^u, DF_{y_j}^v)$, which in turn is bounded by the computation of the maximum weighted bipartite matching. Let the neighbors of u be $x_1, x_2, \ldots, x_m$. ($y_1, y_2, \ldots, y_n$ are neighbors of v.) If $deg(v) < deg(u)$ (i.e., $n < m$), then we can just apply Gabow and Tarjan's algorithm $deg(v)$ times. On the other hand, if $deg(v) \geq deg(u)$, then in the maximum weighted matching Ma between $x_1, x_2, \ldots, x_m$ and $y_1, y_2, \ldots, y_n$, there exist $deg(v) - deg(u)$ subtrees (among $D_v^{y_1}, D_v^{y_2}, \ldots, D_v^{y_n}$) that are not in the matching Ma. Suppose y_j is not in Ma for some $1 \leq j \leq n$. Then we do not need to compute the

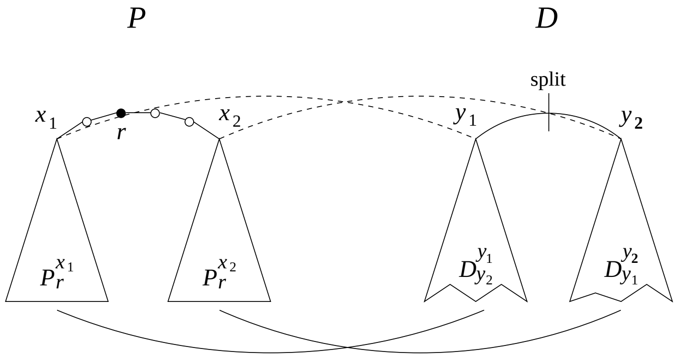


Fig. 6. Illustration of Case 2 in computing $\delta_{cut}(P, D)$ for two graphs P and D.

maximum weighted matching for $DF^v_{y_j}$. (If y_j is in Ma, we have to compute the maximum weighted matching.) The number of such computations is therefore $deg(u)$. Thus, we only need to apply Gabow and Tarjan's algorithm $\min(deg(u), deg(v))$ times. Therefore, the time complexity is $\sum_{i=1}^{N_P} \sum_{j=1}^{N_D} O(m_i n_j \sqrt{\min\{m_i, n_j\}} \log(\min\{m_i, n_j\})) \min\{m_i, n_j\} \leq O(N_P N_D (\min\{d_P, d_D\}) \sqrt{\min\{d_P, d_D\}} \log(\min\{d_P, d_D\})) = O(N_P N_D\, d \sqrt{d} \log d)$.

For Case 2, since the best mapping from $P_r^{x_1}$ to $D_{y_2}^{y_1}$ and the best mapping from $P_r^{x_2}$ to $D_{y_1}^{y_2}$ can be obtained during the computation of Case 1, this case requires at most $O(N_P N_D)$ time. Therefore the total time required by Algorithm G is as asserted in the proposition. □

This is the fastest algorithm, as far as we know, for finding approximate patterns in undirected acyclic graphs and freely allowing cuts. The most closely related work are the algorithm for computing the degree-2 distance between two such graphs without allowing cuts, which runs in time $O(N_P N_D d\sqrt{d} \log d)$ [24], and the algorithm for computing a related but different metric without allowing cuts, which runs in time $O(N_P N_D q^3)$ [10], where $q = \max\{d_P, d_D\}$, $d = \min\{d_P, d_D\}$, and d_P and d_D are the maximum degree of P and D, respectively.

4 Applications

In this section we discuss two applications of the proposed algorithms in chemical information search and website management. First, we show how Algorithm T can be applied to website design and analysis. Then we show the application of Algorithm G to chemical substructure search.

4.1 Approximate Website Matching

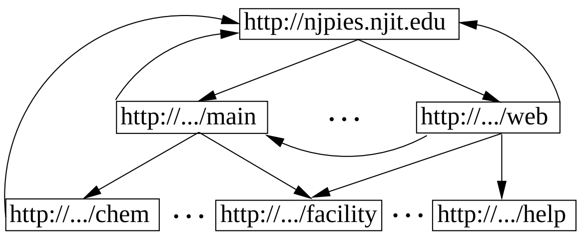
Topologically, a website is a cyclic graph. We delete those hyperlinks that do not point to the same domain and represent a website as a rooted, labeled, unordered tree (after removing certain edges in a user-directed manner as described below). Each node in the tree represents a page in the website; the node label is the URL of that page. The children of a node v contain the URLs of the pages referenced in v's page. That is, there is a link from v's page to each child page.

In translating the graph structure of a website to a rooted unordered tree, we designate a user-specified node as the root. (A common heuristic might be to use the main page as the root.) Then perform a breadth first search from the root and remove back edges, forward edges and cross edges, ending up with a breadth-first spanning tree. Figure 7 illustrates the result of the procedure, using the website of an Environmental Engineering Lab as an example. By employing Algorithm T, one may use the website in Fig. 7(b) as a template and find all the websites on the Internet that approximately contain the template. The "approximation" here is measured by the number of node inserts, node deletes and node relabelings, where a node relabeling refers to the fact that the URL of the corresponding page is being changed. This operation is useful in, for example, analyzing a competitor's site, designing new websites based on a predefined, preferred template, or as an enhancement to a query language.

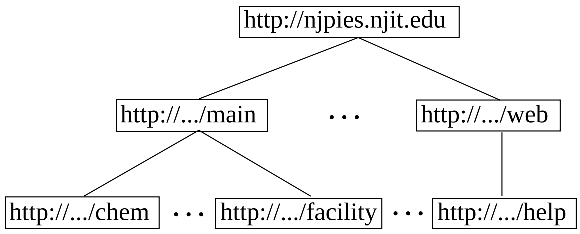
To evaluate the suitability of the degree-2 metric and the usefulness of Algorithm T in website management, we applied the algorithm to searching for websites approximately containing the pattern in Fig. 8(a), within various distances ranging from 1 to 5. It was observed that most qualified websites were returned except for several cases, which arise mainly due to the constraint imposed on the metric. Figure 8 illustrates one such case.

4.2 Chemical Substructure Search

The 2D representation of chemical molecules, as described in chemistry catalogs such as the Merck Index, comprises cycles (e.g. benzene rings) and atoms. In applying our algorithm to chemical substructure search, we remove the cycles by representing a molecule by a



(a)



(b)

Fig. 7. (a) The graph structure of a website. Each node in the graph represents a page in the website; the node label is the URL of that page. An edge from a node u to a node v indicates that v is referenced in u's page. That is, there is a link from u's page to v. (b) The rooted unordered tree representation of the website in (a).

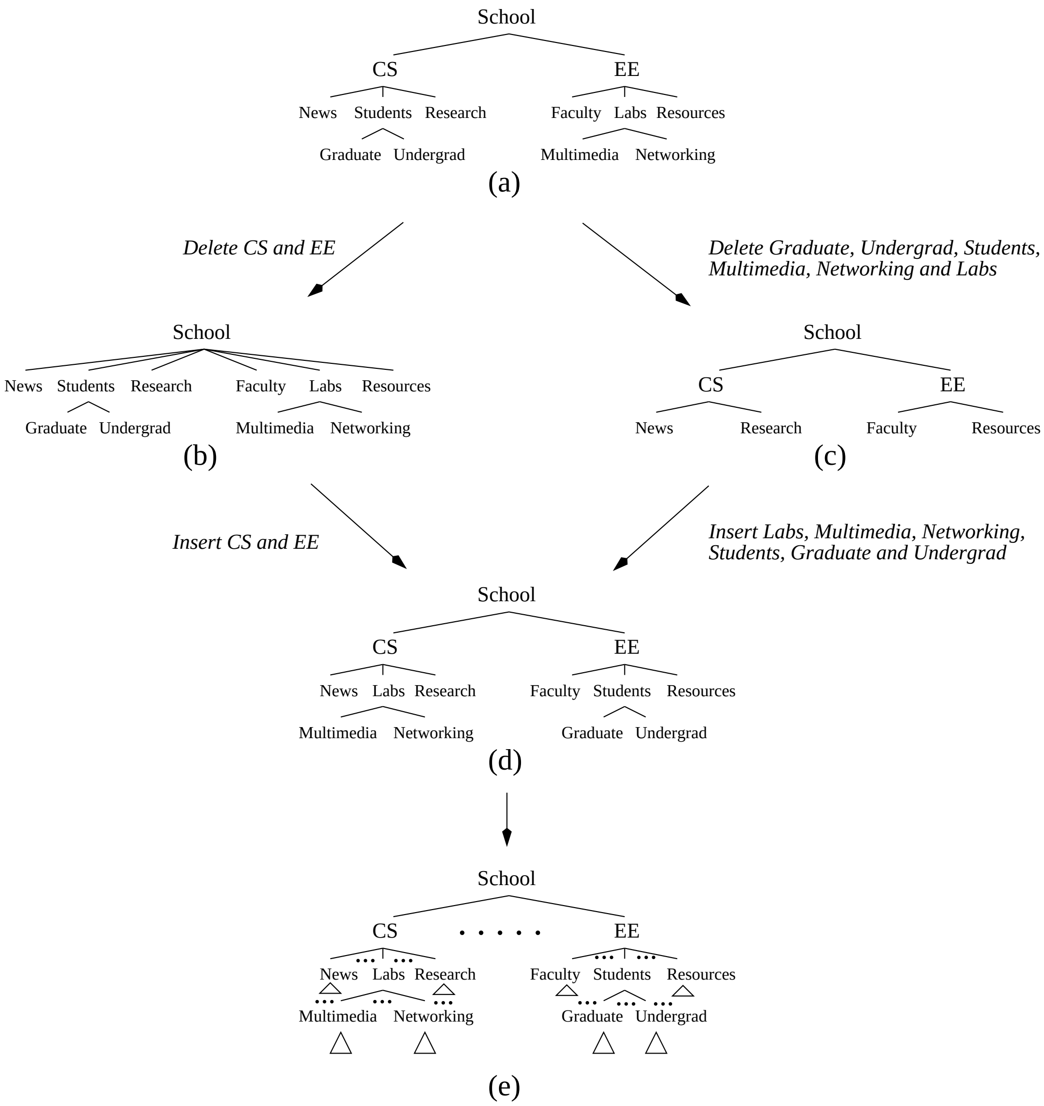


Fig. 8. (a) A website pattern tree. (b) Without the degree-2 constraint, one could delete the nodes labeled "CS" and "EE" and then insert the nodes labeled "CS" and "EE" back with some of their children altered. The resulting tree is as shown in (d). (c) With the degree-2 constraint, one would delete the nodes labeled "Graduate", "Undergrad", "Students", "Multimedia", "Networking" and "Labs". Then, insert the nodes labeled "Labs", "Multimedia" and "Networking" under the node labeled "CS" and insert the nodes labeled "Students", "Graduate" and "Undergrad" under the node labeled "EE", which also yields the tree in (d). Suppose that all the edit operations have unit cost and the allowed distance in the search is 4. Then, when matching the pattern tree in (a) with the data tree in (e) and allowing cuts, without the degree-2 constraint, the distance would be 4, and hence the data tree in (e) would be returned as a qualified search result. However, with the degree-2 constraint, the distance is 12, and therefore this data tree in (e) is missed. Note that nodes appearing in the tree of (e) but not in the tree of (d) are cut at no cost.

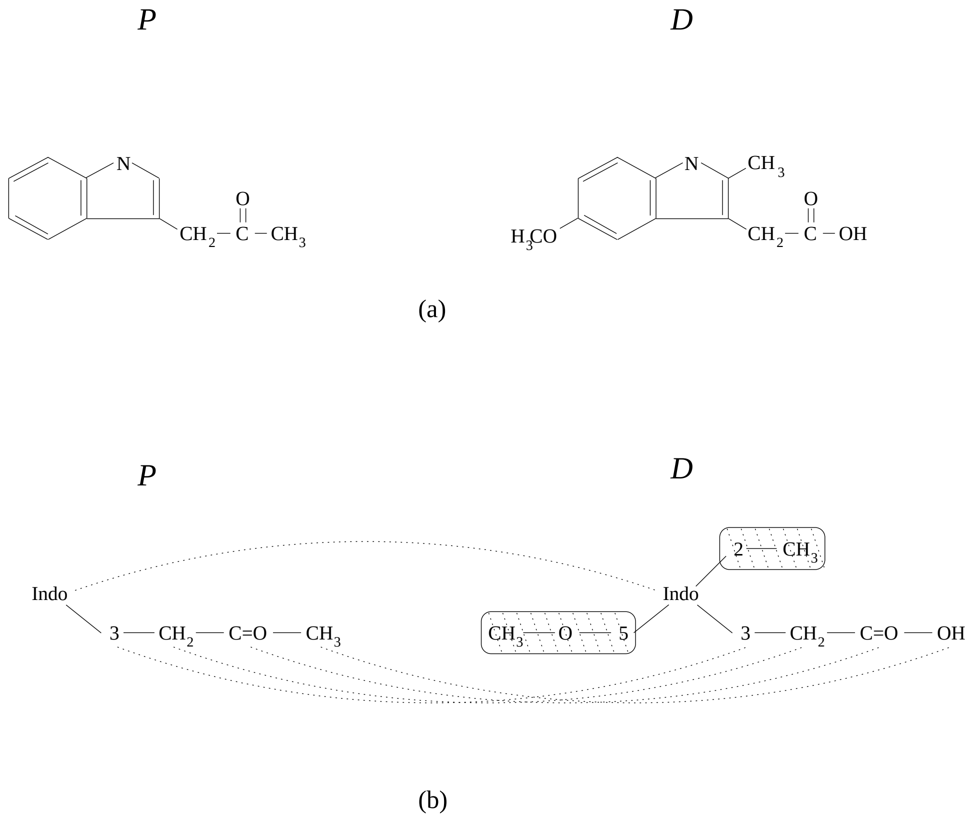


Fig. 9. (a) A pattern molecule P and a data molecule D; (b) the corresponding undirected acyclic graph representations and a minimum cost mapping from graph P to graph D.

collection of smallest, chemically meaningful groups. For example, we use the node label "Ph" to represent p-hydroxybenzoic for the benzene ring, "Indo" for the indo group, the label "C=O" for the ketone group, the label "OH" for the hydroxy in the acid group, and so forth. Figure 9 shows a pattern molecule P, a data molecule D, and their undirected acyclic graph representations. The label "3" is used to represent the position on which CH_2 substitutes, the label "5" is used to represent the position on which O substitutes, etc. A position on which substitution has taken place is labeled according to the traditional organic naming method [14].

In Fig. 9, $\delta_{cut}(P, D) = 1$, i.e., D approximately contains P with distance 1. The shaded areas are cut at no cost and the distance 1 comes from relabeling CH_3 by OH, which costs 1. In general, a pattern can be a base structure comprising one or two functional groups. One then applies Algorithm G to search a database of molecules to find those approximately containing the pattern. The result is a collection of molecules with the desired property, which can be used for further chemical analysis or drug design [29]. In our experiment, we applied Algorithm G to 86 chemical molecules using the molecule P in Fig. 10(a) as the pattern, and the distance allowed in the search was 1. All the 25 molecules having distance 1 to the pattern were found, except the molecule D shown in Fig. 10(a). This high recall indicates the usefulness of the algorithm in practical applications.

5 Conclusion

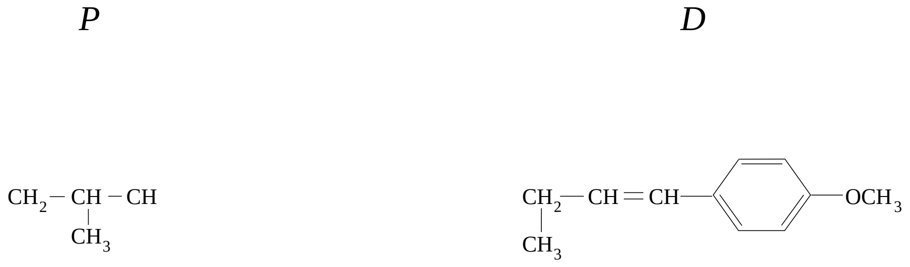
The approximate pattern matching problem has been studied for strings [12] and ordered trees [13]. The algorithms presented in this paper are the first for finding approximate patterns in undirected acyclic graphs as opposed to measuring the distance between two such graphs. The "approximation" here is measured by the degree-2 distance metric proposed in [24]. The time complexity of our algorithm for the graphs is $O(N_P N_D d\sqrt{d}\log d)$ where N_P and N_D are the number of nodes in pattern graph P and data graph D, respectively; $d = \min\{d_P, d_D\}$; d_P and d_D are the maximum degree of P and D, respectively. To demonstrate the utility of our algorithms, we have shown two applications of them in website management and chemical substructure search. Our experimental results indicated the suitability of the degree-2 metric and the usefulness of the algorithms in the applications. Currently we are integrating the algorithms into a previously developed graphics toolkit [30] for document, web and scientific data management.

Acknowledgments

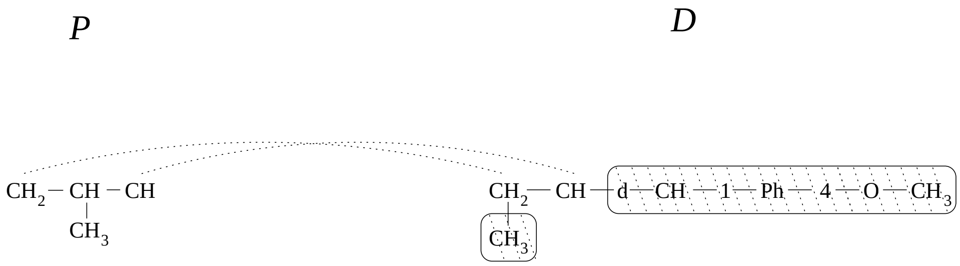
We thank Zasha Weinberg for his contributions in the early phase of this project. We also thank Professor Carol Venanzi for useful discussions on chemical retrieval systems. This work was supported in part by NSF grants IIS-9988345, IIS-9988636, and by the Natural Sciences and Engineering Research Council of Canada under Grant No. OGP0046373.

References

[1] F. A. Akinniyi and A. K. C. Wong. A new product graph based algorithm for subgraph isomorphism. In *Proceedings of the International Conference on Computer Vision and*



(a)



(b)

Fig. 10. (a) A pattern molecule P and a data molecule D; (b) the corresponding undirected acyclic graph representations and a minimum cost mapping from graph P to graph D. The shaded areas are cut at no cost. $\delta_{cut}(P, D) = 2$, representing the cost of deleting CH_3 and then deleting CH in P. Without the degree-2 constraint, one would only delete the CH and connect the CH_3 to CH_2, which would yield a distance 1.

Pattern Recognition, pages 457–467, 1983.

[2] H. A. Almohamad and S. O. Duffuaa. A linear programming approach for the weighted graph matching problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15:522–525, 1993.

[3] J. K. Cheng and T. S. Huang. A subgraph isomorphism algorithm using resolution. *Pattern Recognition*, 13:371–379, 1981.

[4] M. A. Eshera and K. S. Fu. An image understanding system using attributed symbolic representation and inexact graph-matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8:604–619, 1986.

[5] S. Kasif, L. Kitchen, and A. Rosenfeld. A Hough transform technique for subgraph isomorphism. *Pattern Recognition Letters*, 2:83–88, 1983.

[6] B. T. Messmer and H. Bunke. A new algorithm for error tolerant subgraph isomorphism detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):493–504, 1998.

[7] L. G. Shapiro and R. M. Haralick. Structural descriptions and inexact matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3(5):504–519, 1981.

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

[9] A. Shoukry and M. Aboutabl. Neural-network approach for solving the maximal common subgraph problem. *IEEE Transactions on Systems, Man, and Cybernetics*, 26-B(5):785–790, 1996.

[10] E. Tanaka. A metric between unrooted and unordered trees and its bottom-up computing method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(12):1233–1238, 1994.

[11] W. H. Tsai and K. S. Fu. Error-correcting isomorphisms of attributed relational graphs for pattern analysis. *IEEE Transactions on Systems, Man and Cybernetics*, 12:757–768, 1979.

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

[13] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18(6):1245–1262, 1989.

[14] J. E. Ash, P. A. Chubb, S. E. Ward, S. M. Welford, and P. Willett. *Communication, Storage and Retrieval of Chemical Information.* Ellis Horwood, Chichester, England, 1985.

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

[16] J. T. L. Wang, B. A. Shapiro, and D. Shasha, editors. *Pattern Discovery in Biomolecular Data: Tools, Techniques and Applications.* Oxford University Press, New York, 1999.

[17] M. G. Hicks and C. Jochum. Substructure search system. 1. Performance comparison of the MACCS, DARC, HTSS, CAS registry MVSSS, and S4 substructure search systems. *Journal of Chemical Information and Computer Sciences*, 30:191–199, 1990.

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

[19] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York, 1979.

[20] R. D. Brown, G. Jones, P. Willett, and R. C. Glen. Matching two-dimensional chemical graphs using genetic algorithms. *Journal of Chemical Information and Computer Sciences*, 34:63–70, 1994.

[21] M. A. Johnson and G. M. Maggiora, editors. *Concepts and Applications of Molecular Similarity.* John Wiley & Sons, Inc., New York, 1990.

[22] X. Wang and J. T. L. Wang. Fast similarity search in three-dimensional structure databases. *Journal of Chemical Information and Computer Sciences*, 40(2):442–451, 2000.

[23] G. Mehldau. *A Pattern Matching System for Biosequences.* Ph.D. Dissertation, Department of Computer Science, University of Arizona, 1991.

[24] K. Zhang, J. T. L. Wang, and D. Shasha. On the editing distance between undirected acyclic graphs. *International Journal of Foundations of Computer Science*, 7(1):43–57, 1996.

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

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

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

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

[29] I. D. Kuntz. Structure-based strategies for drug design and discovery. *Science*, 257:1078–1082, 1992.

[30] J. T. L. Wang, K. Zhang, K. Jeong, and D. Shasha. A system for approximate tree matching. *IEEE Transactions on Knowledge and Data Engineering*, 6(4):559–571, 1994.

About the Author—JASON T.L. WANG received the B.S. degree in mathematics from National Taiwan University, and the Ph.D. degree in computer science from the Courant Institute of Mathematical Sciences, New York University, in 1991. He is a full professor in the Computer and Information Science Department at the New Jersey Institute of Technology and Director of the University's Data and Knowledge Engineering Laboratory.

Dr. Wang's research interests include data mining and databases, pattern recognition, bioinformatics, and Web information retrieval. He has published 100 papers, including 30 journal articles, in these areas. He is an editor and author of the book *Pattern Discovery in Biomolecular Data* (Oxford University Press), and co-author of the book *Mining the World Wide Web* (Kluwer). In addition, he is Program Co-Chair of the 2001 Atlantic Symposium on Computational Biology, Genome Information Systems & Technology held at Duke University.

About the Author—KAIZHONG ZHANG received the M.S. degree in mathematics from Peking University, Beijing, People's Republic of China, in 1981, and the M.S. and Ph.D. degrees in computer science from the Courant Institute of Mathematical Sciences, New York University, New York, NY, USA, in 1986 and 1989, respectively. Currently, he is an associate professor in the Department of Computer Science, University of Western Ontario, London, Ontario, Canada. His research interests include pattern recognition, computational biology, image processing, sequential and parallel algorithms.

About the Author—GEORGE CHANG is an Assistant Professor in the Mathematics and Computer Science Department at Kean University. He received his Ph.D. and M.S. in computer and information science from the New Jersey Institute of Technology in 2001 and 1995; and B.S. in computer science and applied mathematics and statistics from the State University of New York at Stony Brook in 1994. Dr. Chang was a Special Lecturer in the Computer and Information Science Department at the New Jersey Institute of Technology before joining Kean University in 1999. His research interests include databases, information retrieval, Web mining, and mobile/wireless application development.

About the Author—DENNIS SHASHA is a professor at NYU's Courant Institute where he does research on biological pattern discovery, combinatorial pattern matching on trees and graphs, software support for the exploration of unfamiliar databases, database tuning, and database design for time series.

After graduating from Yale in 1977, he worked for IBM designing circuits and microcode for the 3090. He completed his Ph.D. at Harvard in 1984 (thesis advisor: Nat Goodman). He has written a professional reference book *Database Tuning: a principled* *approach* (1992, Prentice-Hall), two books about a mathematical detective named Dr. Ecco entitled *The Puzzling Adventures of Dr. Ecco* (1988, Freeman, and republished in 1998 by Dover) and *Codes, Puzzles, and Conspiracy* (1992, Freeman) and a book of biographies about great computer scientists called *Out of Their Minds: the lives and discoveries of 15 great computer scientists* (1995, Copernicus/Springer-Verlag). Finally, he is a co-author of *Pattern Discovery in Biomolecular Data: Tools, Techniques, and Applications* published in 1999 by Oxford University Press. In addition, he has co-authored thirty journal papers and 40 conference papers and spends most of his time in building data mining and pattern recognition software these days. He writes a monthly puzzle column for Scientific American and Dr. Dobb's Journal.