# Combinatorial Pattern Discovery for Scientific Data: Some Preliminary Results*

Jason Tsong-Li Wang† Gung-Wei Chirn‡ Thomas G. Marr§ Bruce Shapiro¶

Dennis Shasha‖ Kaizhong Zhang**

## Abstract

Suppose you are given a set of natural entities (e.g., proteins, organisms, weather patterns, etc.) that possess some important common externally observable properties. You also have a structural description of the entities (e.g., sequence, topological, or geometrical data) and a distance metric. Combinatorial pattern discovery is the activity of finding patterns in the structural data that might explain these common properties based on the metric.

This paper presents an example of combinatorial pattern discovery: the discovery of patterns in protein databases. The structural representation we consider are strings and the distance metric is string edit distance permitting variable length don't cares. Our techniques incorporate string matching algorithms and novel heuristics for discovery and optimization, most of which generalize to other combinatorial structures. Experimental results of applying the techniques to both generated data and functionally related protein families obtained from the Cold Spring Harbor Laboratory show the effectiveness of the proposed techniques. When we apply the discovered patterns to perform protein classification, they give information that is complementary to the best protein classifier available today.

*This work was supported, in part, by the National Science Foundation under Grants IRI-8901699, CCR-9103953, IRI-9224601 and IRI-9224602, by the Office of Naval Research under Grants N00014-90-J-1110, N00014-91-J-1472 and N00014-92-J-1719, by the Natural Sciences and Engineering Research Council of Canada under Grant OGP0046373, by NJIT under Grant SBR-421280 and by a grant from the AT&T Foundation. T. G. Marr was supported by the Department of Energy under Grant DE-FG02-91ER61190 and by the National Institutes of Health under Grant 1RO1-HG0020301A1.

†Computer and Information Science, New Jersey Institute of Technology, Newark, NJ 07102 (jason@vienna.njit.edu).

‡Computer and Information Science, New Jersey Institute of Technology, Newark, NJ 07102 (chin@cis.njit.edu).

§Cold Spring Harbor Laboratory, 100 Bungtown Road, Cold Spring Harbor, NY 11724 (marr@cshl.org).

¶Image Processing Section, Laboratory of Mathematical Biology, Division of Cancer Biology and Diagnosis, National Cancer Institute, National Institutes of Health, Frederick, MD 21701 (bshapiro@ncifcrf.gov).

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

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



SIGMOD 94- 5/94 Minneapolis, Minnesota, USA
© 1994 ACM 0-89791-639-5/94/0005..$3.50

## 1 Introduction

*Combinatorial pattern discovery* or *combinatorial data mining* is the activity of finding structural or topological patterns in data that can lead to important conclusions or prediction of new phenomena. With the significant growth of database sizes in several applications, discovering potentially useful patterns in those databases has become the subject of significant research [6, 10, 31, 33].

Much of the work in database mining (e.g., [2, 9, 14]) has concentrated on record-oriented applications. The goal there is to find correlations among attribute-value pairs that give rise to useful properties. A typical example of a useful property is a purchase pattern (or rule), e.g., which other products do people buy when they buy Khaki pants?

In this paper, we focus on the discovery of structural patterns in scientific data. Thus, our work falls philosophically in the mainstream of data mining work because we search for useful patterns in large databases. Our work differs pragmatically from the mainstream, however, because our pattern metrics are topological and therefore require topological comparison algorithms and novel heuristics for discovery and optimization. The paper focuses on the widely used string edit distance metric in molecular genetics, though the approach has applications to other topological metrics.

### 1.1 The Application

Biologists represent proteins as sequences made from 20 amino acids, each represented as a letter. If

two sequences are almost the same or share very similar subsequences, it may be that the common part may perform similar functions via related biochemical mechanisms [11, 18, 19, 21]. These similarities may involve as few as two or three amino acids in some cases. Whereas computerized tools exist for comparing entire sequences, discovering similar subsequences is still usually done by visual inspection.

We present a new approach to discovering similar patterns in a database of genetic sequences. The patterns (or similarities) we wish to discover are regular expressions of the form $*X_1 * X_2 * \ldots$. The $X_1, X_2, \ldots$ are *segments* of a sequence, i.e., subsequences made up of consecutive letters, and $*$ represents a variable length don't care (VLDC). In matching the expression $*X_1 * X_2 * \ldots$ with a sequence $S$, the VLDCs may substitute for zero or more letters in $S$ at zero cost.

The dissimilarity measure used in comparing two sequences is the *edit distance*, i.e., the minimum weighted number of insertions, deletions and substitutions used to transform one sequence to the other [35] after an optimal substitution for the VLDCs. Theory holds that the edit distance is a useful measure of evolutionary distance [27]. For the purpose of this work, we assume that all the edit operations have unit cost, though the techniques we propose do not depend on that cost assumption or essentially on the edit distance metric.

Example – Finding Sequence Similarities
Consider the three sequences in Figure 1.

$S_1$: YDPMIEDKEYSRLVG
$S_2$: RMKQLGRTYDPAVWG
$S_3$: YDPMNWNFEKETLVG

Fig. 1. Three sequences.

Suppose only exactly coinciding segments of lengths greater than 3 are considered as 'similar.' Then $S_1$ and $S_3$ have one similarity (or common pattern):

$$*S_1[1,4]* = *\text{YDPM}* \iff *S_3[1,4]* = *\text{YDPM}*$$

where $V[x,y]$ is a segment of a sequence $V$ from the $x$th to the $y$th letter inclusively. If similarities within distance one are sought, i.e., one mutation (mismatch, insertion or deletion) is allowed in the similarities, then $S_1$, $S_2$ and $S_3$ share three similar patterns:

$$\begin{aligned} & *S_1[1,4]* = *\text{YDPM}* \\ \iff\ & *S_2[8,11]* = *\text{TYDP}* \\ \iff\ & *S_2[9,12]* = *\text{YDPA}* \\ \iff\ & *S_3[1,4]* = *\text{YDPM}* \end{aligned}$$

If similarities having the form $*X * Y*$ are sought with lengths greater than 7 and one mutation allowed, then $S_1$ and $S_3$ share the following four similar patterns:

$$\begin{aligned} & *S_1[1,4] * S_1[12,15]* = *\text{YDPM} * \text{RLVG} * \\ \iff\ & *S_1[1,5] * S_1[13,15]* = *\text{YDPMI} * \text{LVG} * \\ \iff\ & *S_3[1,4] * S_3[12,15]* = *\text{YDPM} * \text{TLVG} * \\ \iff\ & *S_3[1,5] * S_3[13,15]* = *\text{YDPMN} * \text{LVG} * \end{aligned}$$

End of Example

To discover such (approximately) common patterns in a database of sequences, our overall strategy is first to find candidate segments among a small sample, and then to combine the segments into candidate patterns and check which patterns satisfy the specified requirements.

Many techniques have been published in the literature to solve similar problems.[1] A commonly used one is based on multiple sequence alignment (see [38] for review). The technique is useful when entire sequences in the database are similar. However, when the sequences have only short regions of local similarities, this approach makes no sense. There are also techniques based on local similarity search. The techniques work effectively when similarities meet some constraints, such as they occur in a predetermined number of sequences in the database [26], they differ by mismatches, but not by insertions/deletions [3], or they are situated at almost the same distance from the start of the sequences [34]. In contrast to the above techniques, our approach can find similarities composed of nonconsecutive segments separated by variable length don't cares without prior knowledge of their structures, positions, or occurrence frequency.

The rest of the paper is organized as follows. Section 2 formalizes the discovery queries of interest and presents algorithms for basic query processing. Section 3 discusses optimization techniques that apply to combinatorial pattern discovery in general. Section 4 evaluates the effectiveness of the algorithms. Section 5 discusses an application of our techniques, showing how they can help in data classification. Section 6 concludes the paper.

## 2 The Basic Queries and Algorithms

### 2.1 Basic Query Type

Given a database $\mathcal{D}$ of sequences, there exist various requirements on the lengths and forms of similarities to be sought. The following parameters appear to be most

[1] These problems are mostly concerned with discovering patterns made up of single segments, or multiple segments separated by fixed length don't cares.

significant (all the parameter values are specified by the user):

- the form of patterns, in our case regular expressions of the form $*X_1 * X_2 * \ldots$.
- the minimum length of a pattern of interest *Length*, in our case the number of the non-VLDC letters.
- the distance metric, in our case edit distance with unit cost having free substitution for VLDCs (the asterisks).
- the allowed distance *Dist*.
- the minimum occurrence number *Occur* with respect to the distance and length of a chosen pattern. The occurrence number or activity of a pattern is the number of sequences in $\mathcal{D}$ matching the pattern within the allowed distance. We say the occurrence number of a pattern $P$ with respect to distance $i$ and set $\mathcal{S}$, denoted $occurrence\_no^i_{\mathcal{S}}(P)$, is $k$ if $*P*$ matches $k$ sequences in $\mathcal{S}$ within distance at most $i$, i.e., the $k$ sequences contain $P$ within distance $i$.

The basic query is to find the patterns $P$ where $P$ is within the allowed distance *Dist* of at least *Occur* sequences in $\mathcal{D}$ and $|P| \geq Length$.[2] We can use the results of this query in several ways. For example, natural scientists may attempt to evaluate whether the (approximately) common patterns are in fact the active sites; computer scientists may use the patterns to classify new proteins into one family or another as we will show later.

The basic subroutine is to match a given pattern against a given sequence in the database. For example, in matching *TQI* with a sequence MYALTIHKR, the first asterisk would substitute for MYAL and the second asterisk would substitute for HKR. The distance is 1 (representing the cost of deleting Q). The length of the pattern *TQI* is three.[3]

## 2.2 Query Processing Algorithms

Our approach is a two phase process:

1. Find candidate segments among a small sample $\mathcal{A}$ of the sequences.
2. Combine the segments to form candidate patterns and evaluate the activity of the patterns in all of $\mathcal{D}$ to determine which patterns are solutions of the query.

[2] Many related queries are also possible, e.g., a query to identify all patterns having at most a certain length with at least a certain activity.

[3] Given a regular expression pattern $P$ and sequence $S$, one can find if $P$ is within distance *Dist* of $S$ in $O(Dist \times |S|)$ time when $O(|P|) = O(\log |S|)$ [39].

Phase 1 consists of two subphases. In subphase A, we construct an index structure for the sequences in the sample. In subphase B, we traverse the structure to locate the candidate segments.

### 2.2.1 Subphase A of Phase 1

We construct a *generalized suffix tree* [17] (GST) for the sample of sequences. A suffix tree is a trie-like data structure that compactly represents a string by collapsing a series of nodes having one child to a single node whose parent edge has a string. Suffix trees are used extensively in string matching [20, 23]. A GST is an extension of the suffix tree, designed for representing a set of strings. Each suffix of a string is represented by a leaf in the GST. Each leaf is associated with an index $i$. The edges are labeled with character strings such that the concatenation of the edge labels on the path from the root to the leaf with index $i$ is a suffix of the $i$th string in the set. See Figure 2 for an example (the node labeled with a 1 above the leaf MTRM is an example of the result of a collapsing).[4]

For each non-leaf node $v$ in the GST, let $subtree(v)$ be the subtree rooted at $v$. Let $string(v)$ be the string on the edge labels from the root to $v$. Let $count(v)$ represent the number of different indexes associated with the leaves in $subtree(v)$.

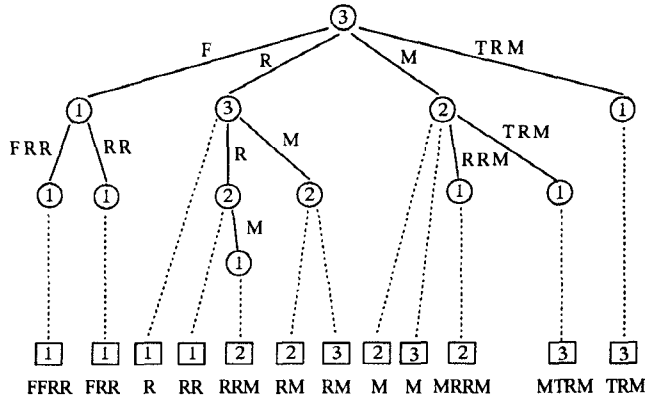


Fig. 2. The GST for a sample $\mathcal{A}$ = {FFRR, MRRM, MTRM}. Leaves are represented as rectangles, labeled with the indexes. Non-leaf nodes are represented as circles, labeled with the *count* values. The suffix corresponding to a leaf is shown below the leaf. Note that the suffixes RM and M appear in two strings and hence appear twice in the leaves.

[4] Informally, the algorithm for constructing the GST works as follows. We append a unique symbol to each sequence in the sample and concatenate the sequences into a single one. We insert the suffixes of the sequences as into a trie except that if a node has only one child, we collapse the child with the parent and label the edge going down from the parent with a substring instead of a single character.

We observe the following facts.

**Fact 1.** $\forall u \in subtree(v)$, $count(u) \leq count(v)$ *and* $|string(u)| > |string(v)|$.

**Fact 2.** *If* $count(v) = b$, *then* $occurrence\_no^0_{\mathcal{A}}(string(v)) = b$.
The reason is that if $count(v) = b$, $string(v)$ is a prefix of the suffixes from $b$ sequences in $\mathcal{A}$.

**Fact 3.** [17, 23] *The time and space needed to construct the GST is* $O(n)$ *where* $n$ *is the total length of all sequences in the sample.*

### 2.2.2 Subphase B of Phase 1

In this phase, we traverse the GST constructed in subphase A to find all segments (i.e., all prefixes of strings labeled on root-to-leaf paths) that satisfy the length minimum. If the pattern specified by the user has the form $*X*$, then the length minimum is simply the specified minimum length of the pattern. If the pattern specified by the user has the form $*X_1 * X_2*$, we find all the segments $V_1, V_2$ where at least one of the $V_i$, $1 \leq i \leq 2$, is (larger than or equal to) half of the specified length, and the sum of their lengths satisfies the length requirement. If the user-specified pattern has the form $*X_1 * X_2 * \ldots * X_k*$, we find the segments $V_1, V_2, \ldots, V_k$ where at least one of the $V_i$, $1 \leq i \leq k$, is (larger than or equal to) $1/k$th of the specified length, and the sum of the lengths of all these segments satisfies the length requirement.

### 2.2.3 Phase 2

This phase also has two subphases. In subphase A, we evaluate the activity of the candidate patterns and rank them from highest to lowest according to their occurrence numbers on the sample with respect to distance *Dist*. If interesting patterns are of the form $*X_1 * X_2 * \ldots$, we consider all possible combinations $V_1, V_2, \ldots$ of the segments obtained in phase 1 that meet the length requirement and match $*V_1 * V_2 * \ldots$ with the sequences in the sample. Subphase B evaluates the most likely candidate patterns found in subphase A with respect to the entire database.

The motivation for having two subphases is that comparing a regular expression pattern $P$ with a sequence $S$ requires a dynamic programming approach that can take, in the worst case, $O(|P| \times |S|)$ time. Screening out those unlikely candidate patterns in the first subphase may save significant time in the overall computation.

## 3 Generalizable Optimization Techniques

Whereas the discussion so far is specialized to the problem of finding patterns in sequences, certain heuristics can improve the efficiency of combinatorial pattern discovery in general.

### 3.1 Pruning Unlikely Candidates

We would like to compare only the most likely candidate patterns with the entire database. The main question from an optimization point of view is which candidates to compare. Our strategy is as follows.

We use *simple random sampling without replacement* [13, 16, 22, 25] to select sample sequences from the database. Consider a candidate pattern $P$. Let $D$ ($a$, respectively) denote the number of sequences in $\mathcal{D}$ ($\mathcal{A}$, respectively) that contain $P$ within the allowed distance. Let $N$ be the database size and $n$ the sample size; $F = D/N$ and $f = a/n$.

**Fact 4.** [8] *With probability = 99%,* $F$ *is in the interval* $(\hat{F}_L, \hat{F}_U)$ *where*

$$\hat{F}_L = f - (t\sqrt{\frac{N-n}{N-1}}\sqrt{\frac{f(1-f)}{n}} + \frac{1}{2n}),$$

$$\hat{F}_U = f + (t\sqrt{\frac{N-n}{N-1}}\sqrt{\frac{f(1-f)}{n}} + \frac{1}{2n}).$$

The symbol $t$ is the value of the normal deviate corresponding to the desired confidence probability. When the probability = 99%, $t = 2.58$ [8]. The values of $N, n$ are given; $f, a$ can be obtained by checking with the sample (cf. subphase A of phase 2). Thus, if the estimator $(\hat{F}_U \times N) < Occur$ for the candidate pattern $P$, then with probability $\geq$ 99%, $P$ won't be a solution. We therefore discard it. This pruning will be referred to as *candidate pattern optimization*. Since the optimization has to do only with sampling, it can be applied to not only sequences, but objects having other topological structures.

### 3.2 Eliminating Redundant Calculation of Occurrence Numbers

Observe that the most expensive operation in our algorithms is to find the occurrence number of a pattern with respect to the database, since that entails matching the pattern against all sequences. We develop two heuristics to avoid such computation when possible.

**Definition 1.** (Subpatterns for sequences) Let $P = *U_1 * U_2 * \ldots * U_m*$ and $P' = *V_1 * V_2 * \ldots * V_n*$ where $m \leq n$. An *embedded mapping* $M$ from $P$ to $P'$ is a set of $m$ ordered pairs of integers $(i, j)$ satisfying:

1. $1 \leq i \leq m$, $1 \leq j \leq n$ and $i \leq j$;
2. for any two distinct pairs $(i_1, j_1)$ and $(i_2, j_2)$ in $M$, (a) $i_1 \neq i_2$ and $j_1 \neq j_2$, (b) $i_1 < i_2$ iff $j_1 < j_2$;
3. if $(i, j) \in M$, then $V_j = X \bullet U_i \bullet Y$ where $X$ and $Y$ are two (possibly empty) segments and $\bullet$ represents the concatenation of segments. (Thus, $U_i$ is a segment of $V_j$.)

$P$ is a *subpattern* of $P'$ if there exists an embedded mapping from $P$ to $P'$.

**Proposition 1.** *If $P$ is a subpattern of $P'$, then for any distance parameter $k$,*

$$occurrence\_no^k_{\mathcal{D}}(P) \geq occurrence\_no^k_{\mathcal{D}}(P').$$

**Proof.** Let $dist(P, S)$ represent the distance between a pattern $P$ and sequence $S$. The result follows by observing that for any sequence $S \in \mathcal{D}$, if $dist(P', S) = j$ for an integer $j$, we must have $dist(P, S) \leq j$. □

Thus, if $P'$ is in the final output set, then we need not bother evaluating $P$, since it will be too. If $P$ is not in the final output set, then $P'$ won't be either, since its occurrence number will be even lower.

Let $occurrence\_set^k_{\mathcal{D}}(P)$ denote the set of all sequences in $\mathcal{D}$ that contain $P$ within distance $k$, i.e., $|occurrence\_set^k_{\mathcal{D}}(P)| = occurrence\_no^k_{\mathcal{D}}(P)$.

**Proposition 2.** *If $P$ and $P'$ are subpatterns of $P''$, then for any distance parameter $k$,*

$$occurrence\_set^k_{\mathcal{D}}(P'') \subseteq$$

$$(occurrence\_set^k_{\mathcal{D}}(P) \cap occurrence\_set^k_{\mathcal{D}}(P')).$$

**Proof.** For any sequence $S \in \mathcal{D}$, if $S \in occurrence\_set^k_{\mathcal{D}}(P'')$, by definition, we must have $dist(P'', S) = j$ for some integer $j \leq k$. It follows that $dist(P, S) \leq j$ and $dist(P', S) \leq j$. Hence, $S \in occurrence\_set^k_{\mathcal{D}}(P)$ and $S \in occurrence\_set^k_{\mathcal{D}}(P')$. □

Thus if $|(occurrence\_set^k_{\mathcal{D}}(P) \cap occurrence\_set^k_{\mathcal{D}}(P'))| < Occur$, we can eliminate $P''$ from consideration, since its occurrence number will be even lower. We refer to the pruning strategies derived from the above propositions as *evaluation minimization.*

Example – Illustration of the Two-Phase Approach
Consider the database $\mathcal{D}$ = {TFUR, MRRM, FFRR, MTRM, DPKY, VRWM, AVLG, KMRR}. Consider the query "Find the patterns $P$ of the form $*X*$ where $P$ is within distance 1 of at least 5 sequences in $\mathcal{D}$ and $|P| \geq 3$."

Suppose the chosen sample $\mathcal{A}$ = {MRRM, FFRR, MTRM, DPKY, AVLG}. At the end of phase 1, we obtain the following candidate patterns:

| | | |
|---|---|---|
| *MRR* | *RRM* | *MRRM* |
| *FFR* | *FRR* | *FFRR* |
| *MTR* | *TRM* | *MTRM* |
| *DPK* | *PKY* | *DPKY* |
| *AVL* | *VLG* | *AVLG* |

By the statistical estimator, the most likely candidate patterns must occur (within distance 1) in at least 2 sequences in the sample. If a candidate is unlikely to be an answer, then any pattern containing it as a subpattern is unlikely either and should be discarded. Thus, at the end of subphase A of phase 2, we are left with

| | | |
|---|---|---|
| *MRR* | *RRM* | *MRRM* |
| | *FRR* | |
| *MTR* | *TRM* | *MTRM* |

In subphase B of phase 2, since $occurrence\_no^1_{\mathcal{D}}(*MRR*) = 4$, we discard *MRRM*. Similarly, since $occurrence\_no^1_{\mathcal{D}}(*MTR*) = 3$, we discard *MTRM*. We compute $occurrence\_no^1_{\mathcal{D}}(*TRM*) = 2$ and $occurrence\_no^1_{\mathcal{D}}(*FRR*) = 4$. Neither of the two patterns is an answer. The only answer to the query is *RRM* where $occurrence\_no^1_{\mathcal{D}}(*RRM*) = 5$.

End of Example

### 3.3 Generalizing to Other Combinatorial Structures

The evaluation minimization techniques presented in the previous subsection have to do with relationships among patterns of the following form: if $dist(P, O) = d$ for some data object $O$ and integer $d$, then $dist(P', O) \geq d$. This allows two conclusions. First, whenever $P$ matches too few objects in the database within distance $d$, then $P'$ will surely match no more. Second, if $P'$ matches enough objects in the database within distance $d$, then $P$ will surely match enough as well. The goal here is to characterize the relationships between $P$ and $P'$ of that form in a more general setting.

**Definition 2.** (Subpatterns for general objects) Let $P$ and $P'$ be two patterns containing VLDCs.[5] $P$ is a *subpattern* of $P'$ if any object (which may or may not be in the database) $C$ that matches $P'$ within distance 0 will match $P$ within distance 0. This conforms to the

[5]The definition of VLDCs may vary for different types of objects. For trees, for example, a VLDC may substitute for a path of nodes in a tree [41].

intuition that $P$ is the less constraining of the two patterns.

Call the set of objects within distance 0 of $P$, $Obj(P)$. The above definition of subpattern implies that $Obj(P) \supseteq Obj(P')$.

**Definition 3.** A distance metric *dist* is said to be *VLDC-sensitive* if for any pattern $P$ containing VLDCs and object $Q$, $dist(P,Q) = \min_{C \in Obj(P)}\{dist(C,Q)\}$.

**Proposition 3.** *If $P$ is a subpattern of $P'$ and dist is VLDC-sensitive, then for any object $Q$ in the database, $dist(P,Q) \leq dist(P',Q)$.*

**Proof.** Since $P$ is a subpattern of $P'$, $Obj(P) \supseteq Obj(P')$. So, $dist(P,Q) = \min_{C \in Obj(P)}\{dist(C,Q)\} \leq \min_{C \in Obj(P')}\{dist(C,Q)\} = dist(P',Q)$, which gives the result. □

Under this generalization of the notion of subpattern and this characterization of distance, Propositions 1 and 2 remain true, and therefore evaluation minimization still applies.

# 4 Performance Analysis

## 4.1 Data and Parameters

We carried out a series of experiments to evaluate the effectiveness and speed (measured by elapsed CPU time) of our approach. The programs were written in C and run on a Sun SPARC workstation under the SUN operating system version 4.1.2. The data was a set of randomly generated 150 sequences, each having length 100. Every letter of the generated sequences was drawn randomly from the protein alphabet. To gain a better understanding of the performance of our algorithms, we also tested the algorithms on real protein sequences. 150 proteins were selected randomly from the functionally related kinase family obtained from the Cold Spring Harbor Laboratory. The lengths of the kinase sequences ranged from 10 to 2938.

Table 1 shows the parameters and base values used in the experiments. The sequences in the sample were chosen randomly from the database. The parameter *NumSample* indicates the number of samples chosen for each database. In all the experiments presented here, only one sample was used in running a database. The sample size was obtained by multiplying *DBSize* by *SizeRatio*. The patterns of interest had the form $*X*Y*$.

| Parameter | Value | Description |
|---|---|---|
| *DBSize* | 150 | # of sequences in a database |
| *NumSample* | 1 | # of samples tested for a database |
| *SizeRatio* | 20% | Ratio between sample size and database size |
| *Length* | 5 | Minimum length of an interesting pattern |
| *Dist* | 1 | Allowed distance between a pattern and a sequence |

Table 1. Experimental parameters and base values.

The metric used to evaluate the effectiveness of our algorithms is

$$HitRatio = \frac{NumDiscovered}{TotalNum} \times 100\%$$

where *NumDiscovered* is the number of interesting patterns discovered by our techniques. *HitRatio* stands for the percentage of the interesting patterns obtained from the exhaustive search method. The method works by considering all combinations of the segment pairs $V_1$, $V_2$ appearing in the database.[6] One would like this percentage to be as high as possible.

## 4.2 Results

Figure 3 shows the effectiveness of our approach for varying sample sizes. In this experiment, we had turned on both candidate pattern optimization and evaluation minimization when running our algorithms.[7] The minimum occurrence number required $Occur = 60$ for the artificial data and $Occur = 8$ for kinase. (The different parameter values were chosen to illustrate different results using different data.) Examining the graphs, we see that when $Dist = 0$ and $SizeRatio \geq 0.2$, our approach behaves almost like exhaustive search. When $Dist = 1$, the hit ratio reaches 80% provided the $SizeRatio \geq 0.4$. We were somewhat disappointed that smaller samples didn't give a better hit ratio, but research is like that sometimes.

[6] We have rejected approximately occurring patterns that never appear in the database yet satisfy the *Dist* and *Occur* constraints in favor of those that obey the constraints and do appear in the database. This is a theoretical limitation of our work that we have introduced to save computation time, though this also seems pragmatically to be a reasonable approach.

[7] This is implemented as follows. The patterns of interest must have length $\geq 5$. So, we begin by enumerating segments of length 3 in the generalized suffix tree (GST). If it is estimated (or actually computed) that the combination of a segment $string(u_1)$ of length 3 with another segment $string(u_2)$ does not render a solution, then we eliminate $string(v_1)$ and $string(v_2)$ from consideration, where $v_1$ and $v_2$ are descendants of $u_1$ and $u_2$, respectively. Similar pruning operations are applied when enumerating longer segments in the GST.

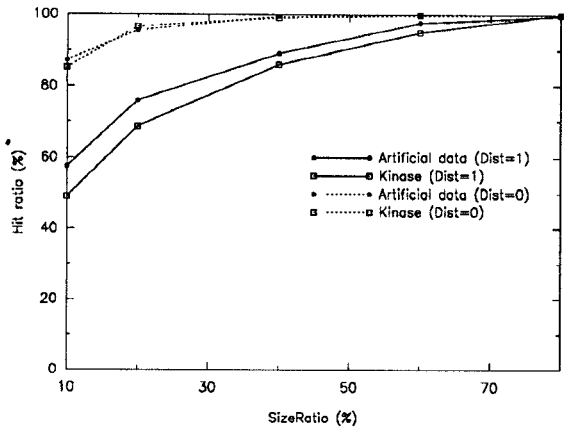


Fig. 3. Effect of sample size.

We next compared the running times of the algorithms for the $Dist = 1$ case. Figure 4 shows the results. It can be seen that our algorithms run significantly faster than the brute force method. Even with $SizeRatio = 0.8$, in which case the algorithms achieve nearly 100% hit ratio, they are 10 times faster than exhaustive search. When the sample is this large, both segments $V_1$, $V_2$ in a solution pattern appear in the sample. Our algorithms work by enumerating all promising segment pairs in the sample, and therefore can find all the interesting patterns.

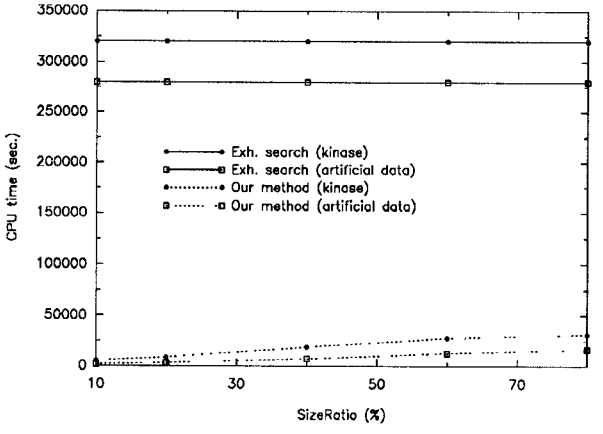


Fig. 4. Comparison of running time.

We also examined the effectiveness of the proposed optimization heuristics. To isolate the effect of the heuristics, we started by turning off the optimizations, and then turned on only one of them, and finally turned on both. To make the experiment manageable, we considered only patterns of the form $*X*$. The minimum occurrence number required $Occur = 55$. The other parameters had the values shown in Table 1. Figures 5 and 6 show the results obtained from the kinase sequences. (The results for the generated sequences are omitted since they lead to similar conclusions.)

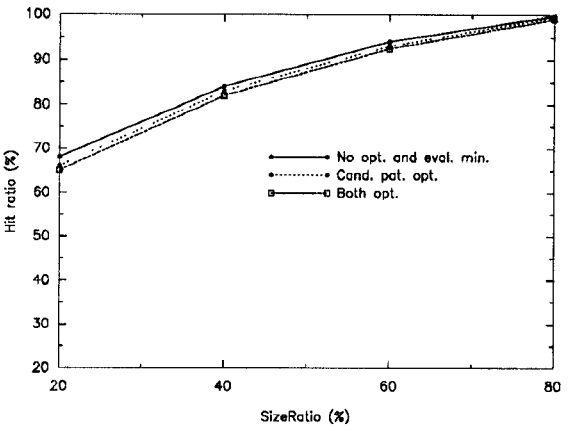


Fig. 5. Performance of the pruning techniques.

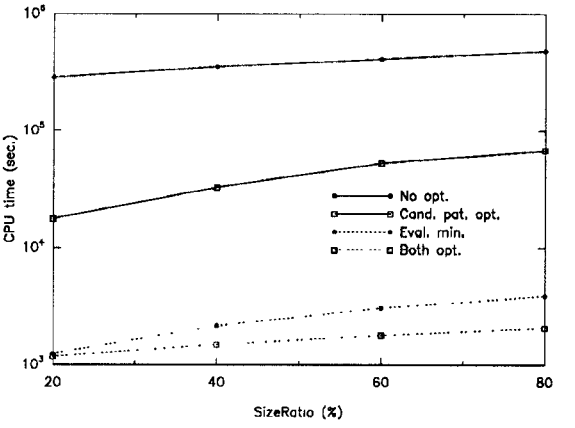


Fig. 6. Efficiency of the pruning techniques.

Examining the graphs, we see that very few solutions were missed by the candidate pattern optimization. Pruning based on subpattern information works more effectively than that based on statistical estimation. Both optimizations together sped up the algorithms by a factor of nearly 100.

We repeated the experiments by varying the compositions of samples and parameter values *Length*, *Dist*, *Occur*. The results obtained are mostly consistent with those given above.

## 5 Data Classification

One application of database mining involves the ability to do classification in database systems [1]. In the Quest data mining project, for instance, Agrawal et al. [1] enhanced the database capability with classification queries and proposed algorithms to generate classification rules for a database of records using information in a training set.

By combining our pattern discovery techniques and a previously published fingerprint technique, we have developed a classifier (referred to as the motif-hashing (or MH) classifier for reasons that will be clear later). We applied the classifier to all 698 groups of related proteins in the SWISS-PROT protein sequence databank version 27 [5]. These groups are categorized based on the documentation given in the PROSITE catalog version 11.0 [4]. The groups comprise more than 15,000 proteins.

At present, the best tool to classify these proteins is the blocks server developed in [15] (referred to as HH). HH generates a set of blocks for each group, where a block comprises ungapped aligned regions extracted from the sequences in the group. To classify a target sequence, HH matches the target against all the blocks and displays a collection of groups, ranked based on their relevance to the target.

In contrast to HH, we select 70% of the sequences in each group at random to serve as a training sample.[8] We then see whether the remaining 30% (referred to as the test sequences) are classified correctly according to our classifier and according to the HH method. This experiment favors HH since the HH blocks database is built from all sequences including the 30% that our method treats as unknowns. Nevertheless, we show that our test complements that of HH quite well.

The motif-hashing classifier preprocesses the training sequences in two ways:

- Find 50 representative patterns from each training sample. (These representative patterns constitute what we call "motifs".) The motifs are the length 4 segments having the highest occurrence numbers with respect to distance 0. When there are ties for occurrence numbers with respect to distance 0, we break the ties by considering occurrence numbers with respect to distance 1.[9] To reduce the effect made by 'chance motifs', we associate each motif with a weight based on Zipf's Law [42]. If a motif occurs in $m$ groups, its weight is assigned as $\log_2\lceil(698/m)\rceil$.
- Hash the training sequences using the gapped fingerprint technique [7].

When classifying a test sequence $T$, we first compare $T$ with all the motifs. After comparison, each group obtains a raw score, which equals the sum of the weights of the group's motifs occurring in $T$. The raw score for a group is normalized by dividing it by the total weight of all the motifs in that group and multiplying by 100. We assign $T$ to the group with the highest score, provided that the score is greater than an experimentally determined threshold. (In the study presented here, the threshold was set to 20.) Otherwise we proceed to the second phase.[10]

In the second phase, we hash $T$, using the same hash function as the one used for the training sequences, and assign $T$ to the group containing sequences with the highest vote. If two sequences have the same highest vote, the shorter one is favored.

Table 2 summarizes the classification results. A sequence was classified correctly by HH (respectively, MH) if its group was ranked highest by HH (respectively, MH). The table also shows the results when the two methods agreed (i.e., the highest ranked group returned by both of them was the same) and disagreed on their rankings.

| Classification results | Percentage of the test sequences |
|---|---|
| HH was correct | 93.5% |
| MH was correct | 92.3% |
| HH and MH agreed & both were correct | 86.7% |
| HH and MH agreed & both were wrong | 0.5% |
| HH and MH disagreed & HH was correct | 7.3% |
| HH and MH disagreed & MH was correct | 4.8% |
| HH and MH disagreed & both were wrong | 0.7% |

Table 2. Comparison between HH and MH. (Note: The last five percentages in the table add up to 100%.)

Thus if HH and MH agree, the classification has a high likelihood of being correct. Specifically, the correct agreed-upon classifications divided by the total agreed-upon classification is 86.7%/(86.7% + 0.5%) = 99.4%. On the other hand, if HH and MH disagree, then the likelihood that one is right is (7.3% + 4.8%)/(7.3% + 4.8% + 0.7%) = 94.5%.

## 6 Conclusions

Combinatorial pattern discovery is useful for discovering internal structural properties that result in the common physical manifestations of a group of physical objects. The general strategy we propose here is first to find patterns satisfying structural constraints (of length and

[8] We tried this experiment 20 times, so the numbers we report in this section are averages using these techniques.

[9] We chose this length and this number of motifs because this seems to give good results. These decisions can be changed easily and are compile-time parameters.

[10] In the experiment, it was found that about 30% of the test sequences, on average, went to the second phase.

form) in a small sample, and then to evaluate these on the whole database. To improve the efficiency, we developed two optimization heuristics:

1. evaluate only those patterns that pass a statistical test in the sample;
2. eliminate patterns if certain combinations of simpler ones have already been evaluated and have been shown to be irrelevant.

We applied the proposed techniques to discover active patterns in generated data and functionally related protein sequences. Our experimental results indicated that the discovery algorithms are sensitive to the data, sample size and the distance allowed in matching a pattern with a sequence. When looking for exact matches (distance of 0), small samples work well. On the other hand for inexact matches (distance of 1), the sample needs to be large. The reason is that in some data, there are patterns which exactly appear in very few sequences, but which approximately occur, within distance 1, in many sequences. Unless the sample is so large that it contains at least one of those very few sequences, our algorithms cannot find the active patterns.

When databases are sizable, using a large sample may entail the construction of a "disk-based" suffix tree. To improve efficiency, one could use the coding scheme suggested in Patricia tries' implementation [12, 24] and pack as much as possible of the tree's upper part into memory. Doing so saves I/O since the tree's upper part is accessed more frequently than the lower part of the tree. Clustering the pages containing nodes that are close in depth-first order would also save disk accesses.

Alternatively, one could choose multiple small samples and combine the results obtained from each sample. Another possible approach would be to divide the whole database into $k$ sub-databases and to find the active patterns (i.e., those occurring in at least $Occur/k$ sequences) in each sub-database. If a pattern is active in all the sub-databases, then it must appear in at least *Occur* sequences in the entire database. However, if a pattern is active only in some of the sub-databases, one would need to compare it with all sequences to determine its overall activity.

The work reported here is part of a project for developing a comprehensive toolbox for pattern discovery and pattern retrieval [30, 36, 37] in scientific databases. The goal is to allow a natural scientist to present structural data and a pattern metric and to ask the toolbox to find the common structural motifs in the data according to the metric.

We have concentrated so far on sequences, but would like to apply our previous work on trees and graphs [28, 29, 40, 41] to scientific problems as well [32]. This will require new data structures.

We have little experience with scientific visualization, but believe that it could be a helpful part of our tool. In that area or in any other related to our goal, we welcome cooperation, suggestions, and friendly competition.

## Acknowledgments

We would like to thank Bharathi Subramanian, Ted Leung, and Stan Zdonik for their presentation of the ideas concerning bulk data types in the AQUA data model; Gadi Landau, Sun Wu and Udi Manber for some hints regarding approximate string and regular expression matching; Lucy Garnett and Abdullah Tansel for exchanging interesting ideas when the first author presented part of the work at CUNY; Alex Tuzhilin for pointing out many important references in the area of knowledge discovery; Jorja Henikoff for offering programs to generate PROSITE groups; Amos Bairoch for providing bio-software information available on the Internet; and to the SIGMOD referees whose comments helped to improve the paper.

## References

[1] R. Agrawal, S. Ghosh, T. Imielinski, B. Iyer, and A. Swami. An interval classifier for database mining applications. In *Proceedings of the 18th International Conference on Very Large Data Bases*, pages 560–573, Vancouver, Canada, Aug. 1992.

[2] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, Washington, D.C., May 1993.

[3] D. J. Bacon and W. J. Anderson. Multiple sequence alignment. *Journal of Molecular Biology*, 191:153–161, 1986.

[4] A. Bairoch. PROSITE: A dictionary of sites and patterns in proteins. *Nucleic Acids Research*, 20:2013–2018, 1992.

[5] A. Bairoch and B. Boeckmann. The SWISS-PROT protein sequence data bank. *Nucleic Acids Research*, 20:2019–2022, 1992.

[6] W. Buntine and M. D. Alto, editors. *Collected Notes on the Workshop for Pattern Discovery in Large Databases.* Technical Report FIA-91-07, NASA Ames Research Center, Moffett Field, California, April 1991.

[7] A. Califano and I. Rigoutsos. FLASH: A fast look-up algorithm for string homology. In *Proceedings of the 1st International Conference on Intelligent Systems for Molecular Biology*, Bethesda, MD, July 1993.

[8] W. G. Cochran. *Sampling Techniques.* Wiley, 1977.

[9] V. Dhar and A. Tuzhilin. Abstract-driven pattern discovery in databases. *IEEE Transactions on Knowledge and Data Engineering*, 5(6):926–938, Dec. 1993.

[10] W. J. Frawley, G. Piatetsky-Shapiro, and C. J. Matheus. Knowledge discovery in databases: An overview. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 1–27. AAAI/MIT Press, 1991.

[11] K. A. Frenkel. The human genome project and informatics. *Communications of the ACM*, 34(11):41–51, Nov. 1991.

[12] G. H. Gonnet and R. Baeza-Yates. *Handbook of Algorithms and Data Structures.* Addison-Wesley, Reading, Massachusetts, 2 edition, 1991.

[13] P. J. Haas and A. N. Swami. Sequential sampling procedures for query size estimation. In *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data*, pages 341–350, San Diego, CA, June 1992.

[14] J. Han, Y. Cai, and N. Cercone. Knowledge discovery in databases: An attribute-oriented approach. In *Proceedings of the 18th International Conference on Very Large Data Bases*, pages 547–559, Vancouver, Canada, Aug. 1992.

[15] S. Henikoff and J. G. Henikoff. Automated assembly of protein blocks for database searching. *Nucleic Acids Research*, 19(23):6565–6572, 1991.

[16] W. C. Hou and G. Ozsoyoglu. Statistical estimators for aggregate relational algebra queries. *ACM Transactions on Database Systems*, 16(4):600–654, Dec. 1991.

[17] L. C. K. Hui. Color set size problem with applications to string matching. In A. Apostolico, M. Crochemore, Z. Galil, and U. Manber, editors, *Combinatorial Pattern Matching, Lecture Notes in Computer Science, 644*, pages 230–243. Springer-Verlag, 1992.

[18] L. Hunter, editor. *Artificial Intelligence and Molecular Biology.* AAAI Press/The MIT Press, Menlo Park, CA, 1993.

[19] N. Kamel, M. Delobel, T. G. Marr, R. Robbins, J. Thierry-Mieg, and A. Tsugita. Data and knowledge bases for genome mapping: What lies ahead? Panel Presentation in the 17th International Conference on Very Large Data Bases, Barcelona, Spain, Sep. 1991.

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

[21] R. J. Lipton, T. G. Marr, and J. D. Welsh. Computational approaches to discovering semantics in molecular biology. *Proceedings of the IEEE*, 77(7):1056–1060, July 1989.

[22] R. J. Lipton, J. F. Naughton, and D. A. Schneider. Practical selectivity estimation through adaptive sampling. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 1–12, Atlantic City, NJ, May 1990.

[23] E. M. McCreight. A space-economical suffix tree construction algorithm. *JACM*, 23:262–272, 1976.

[24] D. R. Morrison. PATRICIA – practical algorithm to retrieve information coded in alphanumeric. *JACM*, 15(4):514–534, 1968.

[25] F. Olken and D. Rotem. Random sampling from $B^+$ trees. In *Proceedings of the 15th International Conference on Very Large Data Bases*, pages 269–278, Amsterdam, The Netherlands, Aug. 1989.

[26] M. A. Roytberg. A search for common patterns in many sequences. *Computer Applications in the Biosciences*, 8(1):57–64, 1992.

[27] D. Sankoff and J. B. Kruskal, editors. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison.* Addison-Wesley, Reading, MA, 1983.

[28] B. A. Shapiro and K. Zhang. Comparing multiple RNA secondary structures using tree comparisons. *Computer Applications in the Biosciences*, 6(4):309–318, 1990.

[29] 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(3), March 1994.

[30] D. Shasha and T. L. Wang. New techniques for best-match retrieval. *ACM Transactions on Information Systems*, 8(2):140–158, Apr. 1990.

[31] A. Silberschatz, M. Stonebraker, and J. D. Ullman. Database systems: Achievements and opportunities. *Communications of the ACM*, 34(10):94–109, 1991.

[32] B. Subramanian, S. B. Zdonik, T. W. Leung, and S. L. Vandenberg. Ordered types in the AQUA data model. In C. Beeri, A. Ohori, and D. Shasha, editors, *Database Programming Languages (DBPL-4), Proceedings of the Fourth International Workshop on Database Programming Languages - Object Models and Languages*, pages 115–135. Springer-Verlag, Workshops in Computing Series, 1994.

[33] S. Tsur. Data dredging. *IEEE Data Engineering Bulletin*, 13(4):58–63, Dec. 1990.

[34] M. Vingron and P. Argos. A fast and sensitive multiple sequence alignment algorithm. *Computer Applications in the Biosciences*, 5:115–122, 1989.

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

[36] 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(2), April 1994.

[37] T. L. Wang and D. Shasha. Query processing for distance metrics. In *Proceedings of the 16th International Conference on Very Large Data Bases*, pages 602–613, Brisbane, Australia, Aug. 1990.

[38] M. S. Waterman, editor. *Mathematical Methods for DNA Sequence Analysis*. CRC Press, Boca Raton, FL, 1989.

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

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

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

[42] G. K. Zipf. *Human Behavior and the Principle of Least Effort*. Addison Wesley, Reading, MA, 1949.