An Efficient Algorithm for Optimizing Whole Genome Alignment with Noise

Prudence WH Wong, TW Lam, N Lu, HF Ting and SM Yiu
Department of Computer Science, University of Hong Kong, Hong Kong
April 8, 2004

ABSTRACT

Motivation: This paper is concerned with algorithms for aligning two whole genomes so as to identify regions that possibly contain conserved genes. Motivated by existing heuristic-based software tools, we initiate the study of an optimization problem that attempts to uncover conserved genes with a global concern. Another interesting feature in our formulation is the tolerance of noise, which also complicates the optimization problem. A brute-force approach takes time exponential in the noise level.

Results: We show how an insight into the optimization structure can lead to a drastic improvement in the time and space requirement (precisely, to $O(k^2n^2)$ and $O(k^2n)$, respectively, where n is the size of the input and k is the noise level). The reduced space requirement allows us to implement the new algorithm, called MaxMinCluster, on a PC. It is exciting to see that when tested with different real data sets, MaxMinCluster consistently uncovers a high percentage of conserved genes that have been published by GenBank. Its performance is indeed favorably compared to MUMmer (perhaps the most popular software tool for uncovering conserved genes in a whole-genome scale).

Availability: The source code is available from the website http://www.csis.hku.hk/∼colly/maxmincluster/. Detailed proof of the propositions can also be found there.

Contact: whwong@cs.hku.hk

INTRODUCTION

This paper is concerned with algorithms for aligning the whole genomes of two related species so as to identify regions on the genomes that possibly contain their conserved genes. This problem has attracted a lot of attention in the past few years and a number of software tools have been developed (Baillie and Rose, 2000; Buhler, 2001; Delcher et al., 1999, 2002; Kurtz et al., 2004; Morgenstern, 1999; Morgenstern et al., 1998; Schwartz et al., 2000; Vincens et al., 1998). Whole genome alignment has also been studied for other applications (Altschul et al., 1997; Batzoglou et al., 2000; Frazer et al., 2003; Nieduszynski et al., 2002).

Related species (such as mouse and human) often have a lot of genes conserved, i.e., having the same functionality. Though a conserved gene rarely comprises the same entire sequence in two genomes (probably due to mutations (Gangloff et al., 1996; Mushegian and Koonin, 1996; Plasterk, 1993; Griffiths et al., 2000)), there are usually a lot of short common substrings and some of these substrings are indeed unique to this conserved gene. Thus, the first step to align two genomes would be to identify pairs of maximal substrings that appear uniquely in both genomes. This can be done in linear time using suffix trees (Gusfield, 1997). Of course, not every pair of matched substrings correspond to a conserved gene; in fact, most matched substrings in the input are "noise" and many of them actually originate from intergenic regions. Extracting the right pairs is not a trivial problem. See Figure 1 for an example.

Matched substrings that are noise are usually short and isolated. A conserved gene usually corresponds to a sequence of matched substrings that are consecutive and close in both genomes and have sufficient length. We call such a sequence a *cluster*. A collection of clusters gives us an *alignment* of the conserved genes between the two given genomes. At first glance, the problem of finding conserved genes is a simple clustering problem. Such a clustering approach has been used in practice (Delcher et al., 2002), yet the success is limited. There are two major concerns.

- Some conserved genes do not induce clusters of sufficient size; the primary reason is the presence of noise (which separates the actual matched substrings). To uncover such genes, we have to relax the definition of a cluster to allow the presence of noise.
- To report an alignment, one can simply use some ad hoc or greedy approach to cluster the pairs, but this does not have much control over the quality of the alignment. It is more desirable to find an alignment that satisfies some instinctive

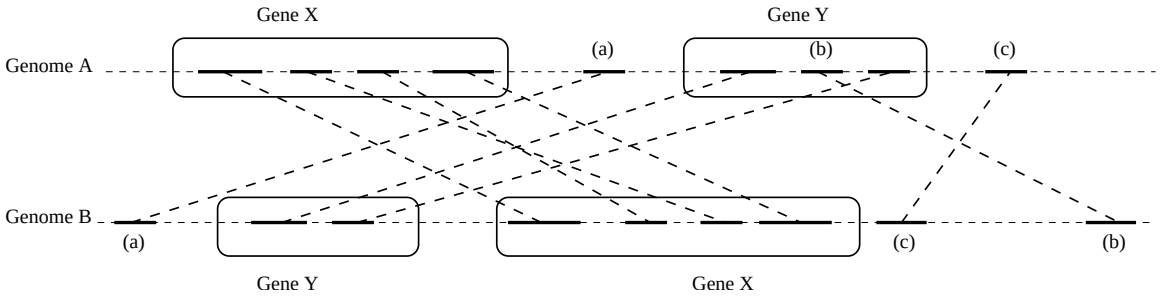

Fig. 1. Among all pairs of matched substrings, a number of them do not originate from conserved genes. See (a), (b), (c) for examples.

criterion, say, maximizing the size of the smallest cluster. Such a criterion could possibly improve the overall quality of the alignment as we avoid reporting relatively small clusters, which are less likely to be conserved genes. But imposing such a criterion increases drastically the time and space requirement for finding an alignment.

In this paper we introduce the notion of k-noisy clusters, which allows us to ignore up to k pairs of matched substrings in considering a cluster (see the formal definition below). We expect that k is a small integer; otherwise, some clusters reported may be of very poor quality. Furthermore, we investigate the optimization problem of finding an alignment that maximizes the size of the smallest cluster. Intuitively, this selection criterion does not favor small clusters, which are less likely to be conserved genes. We believe that the tolerance of noise would enable more conserved genes to be uncovered and the selection criterion would trim away those small clusters.

The bottleneck in solving this optimization problem is on checking whether two regions on the two genomes contain a k-noisy cluster. A brute-force approach is to examine all possible combinations of at most k substrings within the regions, then check whether throwing these substrings away would leave two sequences of matched substrings that are consecutive on each genome and satisfy the size requirement. This requires $O(n^{k+1})$ time, where n is the number of matched substrings in the input. In this paper we show how to exploit the structure of the problems to devise a more efficient algorithm; it requires only $O(k^2)$ time for a pair of regions on average, and $O(k^2n^2)$ time for checking all pairs. This improvement allows us to solve the optimization problem in $O(k^2n^2)$ time using dynamic programming.

A straightforward implementation of the dynamic programming would require $O(kn^2)$ space. This is actually too demanding. For example, in a pair of human and mouse chromosomes, there can be up to a hundred thousand pairs of matched substrings. Assuming $k = 3$, we already need more than ten Gigabytes of memory to implement this algorithm, which far exceeds the capacity of ordinary workstations. Fortunately, we are able to reduce the space requirement of the algorithm to $O(k^2n)$, while maintaining the same time complexity.

The new algorithm, which will be referred to as MaxMinCluster, has been implemented on a PC and compared with a heuristic-based software tool called MUMmer-3 (Kurtz et al., 2004) (which is currently the most popular whole genome alignment software tool). The evaluation of DNA sequence alignment was based on ten pairs of human and mouse chromosomes. Furthermore, we have tested both software tools for aligning translated protein sequences; notice that detecting protein sequence homology is particularly needed for distantly related genomes. For this purpose, we evaluated the software tools using nine virus genomes.

It is encouraging to see that for aligning DNA sequences, MaxMinCluster consistently achieves better coverage while preserving the sensitivity (for coverage, we count the percentage of published conserved genes that are reported; for sensitivity, we count the percentage of the reported clusters that are known to reside in a conserved gene; note that sensitivity is only an estimate as the literature has not yet identified all conserved genes). In some cases, the coverage of MaxMinCluster is as high as 1.5 times of MUMmer-3. See the section on experimental results for details. For aligning translated protein sequences, both software tools are able to achieve very high coverage (an average of 92%) when aligning the human and mouse chromosomes,

MaxMinCluster has a slightly better sensitivity, though. For the case of virus, MaxMinCluster shows a higher coverage consistently.

Remarks: We have also studied another optimization problem which is to find an alignment that minimizes the number of clusters. This selection criterion does not favor small clusters either. We have derived a dynamic programming algorithm whose performance is very similar to MaxMinCluster in terms of coverage, and space and time complexity (see our website).

AN OPTIMIZATION PROBLEM

The input is a sequence $M = (m_1, m_2, \ldots, m_n)$, where each m_i denotes a pair of uniquely matched substrings on two given genomes A and B. More precisely, each m_i is represented as a 4-tuple $(a_i, b_i, \ell_i, \sigma_i)$ where a_i and b_i are respectively the starting positions on A and B, ℓ_i the length of the substrings, and σ_i the orientation of the substrings. A pair of matched substrings on two genomes can originate from two sense strands of the same orientation or from a sense strand and an antisense strand of opposite orientations. If m_i is of same orientation, we set $\sigma_i = 1$; if m_i is of opposite orientations, we set $\sigma_i = -1$. Intuitively, same orientation means that the a_i-th character of the sense strand of A matches the b_i-th character of the sense strand of B, the $(a_i + 1)$-th matches the $(b_i + 1)$-th and so on; while opposite orientations mean that the a_i-th character of the sense strand of A matches the $(b_i + \ell_i - 1)$-th character of the antisense strand of B, the $(a_i + 1)$-th matches the $(b_i + \ell_i - 2)$-th and so on. We assume that $a_1 < a_2 < \cdots < a_n$. Let `MinSize` and `Gap` be two predefined positive constants.

Noisy clusters: A segment of M is a subsequence in the form $(m_i, m_{i+1}, \ldots, m_{i+t})$. Let k be a positive integer. We say that a segment of M is a *k-noisy cluster* if we can remove at most k elements from the segment, denoted by X, such that the resulting subsequence S satisfies the following conditions:

1. The σ_i's of all m_i's in S are the same.
2. If $\sigma_i = 1$, both the a_i's and b_i's of S are increasing; otherwise, the a_i's are increasing while the b_i's are decreasing.
3. For any two consecutive elements m_p and m_q in S, we require that they satisfy some kind of *distance requirement*. Let $g_1 = |a_p - a_q|$ and $g_2 = |b_p - b_q|$. A simple distance requirement is $g_1 \leq$ `Gap` and $g_2 \leq$ `Gap`.[†]
4. `Size`(S), defined to be $\sum_{m_i \in S} \ell_i$, is at least `MinSize`. This is called the *size requirement*.

Intuitively, X corresponds to the noise.

Alignment: An alignment of M, denoted A below, is a maximal collection of disjoint k-noisy clusters, i.e.,

- clusters in A are mutually disjoint;
- there does not exist another k-noisy cluster which is disjoint with all clusters in A (i.e., we cannot add more clusters to A);
- there does not exist another k-noisy cluster which includes some cluster(s) in A and is disjoint with all other clusters in A (i.e., we cannot enlarge any cluster in A).

Max-min alignment problem: We define the weight of a k-noisy cluster C as follows. Note that there may be more than one subset X that makes C qualified as a k-noisy cluster. Among all such X's, let X_o be the one with the smallest size. Define $w(C)$, the weight of C, to be `Size`$(C - X_o)$.

The max-min alignment problem is defined as follows. Given a set M of pairs of matched substrings, we want to find an alignment A^* of M such that

$$\min_{C \in A^*} w(C) = \max_{A \in \Sigma} \min_{C \in A} w(C),$$

where Σ denotes the set of all possible alignments of M. We call A^* an optimal alignment of M and $\min_{C \in A^*} w(C)$ the weight of A^*.

ALGORITHM AND IMPLEMENTATION

The dynamic programming algorithm

In this section, we describe a dynamic programming algorithm, called MaxMinCluster, for the max-min alignment problem. Recall that the input is a sequence $M = (m_1, m_2, \ldots, m_n)$ and Σ denotes the set of all alignments of M. We want to find an optimal alignment A^* of M, i.e., $\min_{C \in A^*} w(C) = \max_{A \in \Sigma} \min_{C \in A} w(C)$. Intuitively, this selection criterion does not favor small clusters, which are less likely to be conserved genes.

[†] Another useful form of distance requirement is $|g_1 - g_2| \leq$ `Diff`, and $|g_1 - g_2| \leq$ `Ratio` $\cdot \max\{g_1, g_2\}$, where `Diff` and `Ratio` are predefined constants. As to be shown, our algorithm is applicable for any distance requirement.

The dynamic programming algorithm computes A^* incrementally, by considering the sequences (m_1), (m_1, m_2), $\ldots$, $(m_1, m_2, \ldots, m_j)$ and so on. Roughly speaking, for any $1 \leq j \leq n$, the algorithm finds the optimal alignment involving $m_1, m_2, \ldots, m_j$ by considering every k-noisy cluster $(m_i, \ldots, m_j)$ for some $1 \leq i \leq j$ and the alignment involving $m_1, m_2, \ldots, m_i$; the best alignment is chosen by trying all such i values. Details are as follows.

For $1 \leq j \leq n$, let Φ_j be the set of all possible k-noisy clusters of M whose elements are in $(m_1, m_2, \ldots, m_j)$. Let Σ_j be the set of all maximal collections of disjoint k-noisy clusters in Φ_j. Define

$$\mathtt{W}(j) = \max_{A \in \Sigma_j} \min_{C \in A} w(C).$$

Note that Φ_n equals the set of all possible k-noisy clusters of M and $\Sigma_n = \Sigma$. Thus, $\mathtt{W}(n)$ is the weight of the optimal alignment of M. To find $\mathtt{W}(j)$, we consider two cases according to whether m_j is included in some cluster in the alignment. Let $\Gamma_j \subseteq \Sigma_j$ be the set of those maximal collections each of which has a k-noisy cluster containing m_j. We define

$$\mathtt{WI}(j) = \max_{A \in \Gamma_j} \left(\min_{C \in A} w(C) \right), \text{ and}$$

$$\mathtt{WE}(j) = \max_{A \in \Sigma_j - \Gamma_j} \left(\min_{C \in A} w(C) \right).$$

Obviously, $\mathtt{W}(j) = \max\{\mathtt{WI}(j), \mathtt{WE}(j)\}$.

The computation of $\mathtt{WI}(j)$ and $\mathtt{WE}(j)$ requires us to determine whether a segment of M in the form $(m_i, m_{i+1}, \ldots, m_j)$ is a k-noisy cluster, for all $1 \leq i \leq j$. To ease our discussion, we denote the segment $(m_i, m_{i+1}, \ldots, m_j)$ as M_{ij}. Proposition 1 states that the computation can be done recursively. We first give the intuitive ideas.

Let S_j be the set of the starting positions of all segments which end at position j and which form a k-noisy cluster. For every $i \in S_j$, the best alignment in Σ_j that contains M_{ij} is the union of $\{M_{ij}\}$ and the best alignment in Σ_{i-1}. On the other hand, let i^* be the largest position in S_j. Let h be the largest index of the matched substring pair in some alignment $A \in \Sigma_j - \Gamma_j$. Note that $i^* \leq h \leq j-1$, otherwise, $A \cup \{M_{i^*j}\}$ is an alignment which contradicts that A is maximal. Therefore, we can compute $\mathtt{WI}(j)$ and $\mathtt{WE}(j)$ recursively as follows.

Proposition 1. *Assume that* $\mathtt{W}(0) = \mathtt{WI}(0) = \mathtt{WE}(0) = 0$. *Then, for any* $j \geq 1$,

- $\mathtt{WI}(j) =$
$$\max \left\{ \begin{array}{l} \max\limits_{i \in S_j, \mathtt{W}(i-1) \neq 0} \min(\mathtt{W}(i-1), w(M_{ij})), \\ \max\limits_{i \in S_j, \mathtt{W}(i-1) = 0} w(M_{ij}) \end{array} \right\};$$
- $\mathtt{WE}(j) = \begin{cases} \max_{h \in [i^*, j-1]} \mathtt{WI}(h) & \text{if } S_j \neq \varnothing, \\ \mathtt{W}(j-1) & \text{otherwise.} \end{cases}$

Based on Proposition 1, we can solve the max-min alignment problem using dynamic programming (see Algorithm MaxMinCluster).

Algorithm MaxMinCluster

1. For each subsequence M_{ij} of M with $1 \leq i \leq j \leq n$, determine whether it is a k-noisy cluster and compute its weight if so. For any $1 \leq j \leq n$, let S_j be the set of all i values such that M_{ij} is a k-noisy cluster, and i^* be the largest such value.
2. Set $\mathtt{W}(0) = \mathtt{WI}(0) = \mathtt{WE}(0) = 0$.
3. For $j = 1$ to n
   (a) For each $M_{ij} \in S_j$, compute the value according to Proposition 1 part 1 in terms of $\mathtt{W}(i-1)$ and $w(M_{ij})$.
   (b) Compute $\mathtt{WE}(j)$ according to Proposition 1 part 2 in terms of $\mathtt{W}(j-1)$ and $\mathtt{WI}(h)$ for $i^* \leq h \leq j-1$.
   (c) Set $\mathtt{W}(j) = \max\{\mathtt{WI}(j), \mathtt{WE}(j)\}$.

Suppose that we have a preprocessing to find all k-noisy clusters and their weights (i.e., Step 1 of Algorithm MaxMinCluster) in $f(n)$ time so that we can answer in $O(1)$ time whether a particular segment is a k-noisy cluster. Consider each iteration of Step 3 of Algorithm MaxMinCluster. Computing $\mathtt{WI}(j)$ and $\mathtt{WE}(j)$ takes $O(j)$ time. Then $\mathtt{W}(j)$ can be computed in $O(1)$ time. Therefore, Step 3 of Algorithm MaxMinCluster takes $O(n^2)$ time and the whole algorithm takes $O(n^2 + f(n))$ time.

The preprocessing, i.e., finding all k-noisy clusters, is non-trivial and indeed the bottleneck. Below, we give an algorithm to find all k-noisy clusters with time $f(n) = O(k^2 n^2)$. In other words, the whole algorithm runs in $O(k^2 n^2)$ time.

Finding the k-noisy clusters

In this section, we show how to find all k-noisy clusters and determine their weights in $O(k^2 n^2)$ time. A brute-force approach is to determine whether a

particular segment M_{ij} is a k-noisy cluster by examining all possible combinations of up to k elements in the segment, and checking whether throwing these elements away would leave a sequence satisfying the k-noisy cluster requirement. This requires $O(n^{k+1})$ time. Hence, computing all k-noisy clusters and their weights takes $O(n^{k+3})$ time. Below we show how to improve the time complexity to $O(k^2n^2)$. The main observation is that we are able to determine whether a segment M_{ij} is a k-noisy cluster by examining a small number of $M_{ij'}$ for some $j' < j$ that have already been considered. Details are given below.

Consider any $1 \leq i \leq j \leq n$. To determine whether a segment M_{ij} is a k-noisy cluster, we try every $M_{ij'}$ with $j' < j$ to check whether it is possible to obtain a k-noisy cluster by extending $M_{ij'}$ to include m_j while satisfying the four requirements of a k-noisy cluster. We observe that the first three requirements of a k-noisy cluster are local concerns while the fourth requirement (i.e., the size requirement) is a global concern in the following sense. If the segment $M_{ij'}$ does not satisfy the first three requirements, it is impossible to extend it such that M_{ij} satisfies these requirements. On the other hand, even if $M_{ij'}$ does not satisfy the size requirement, it is still possible to extend $M_{ij'}$ such that M_{ij} is a k-noisy cluster because adding more matched substring pairs may make the extended segment to be of sufficient size. Based on this observation, we exclude the size requirement when we describe the sub-problem.

Now we describe how to find k-noisy clusters. Recall that `Gap` and `MinSize` are two predefined positive constants. A set $H \subset \{m_i, m_{i+1}, \ldots, m_j\}$ is said to be a set of noise in M_{ij} if $M_{ij} - H$ satisfies the first three requirements of a noisy cluster (i.e., the size requirement is excluded) and the elements in $M_{ij} - H$ are either all of the same orientation or all of the opposite orientations (i.e., the value of σ's are all the same). Notice that $\varnothing$ is also a candidate for H. Let N_{ij}^+ (N_{ij}^-, resp.) be the set of noise H in M_{ij} such that all elements in $M_{ij} - H$ have $\sigma = 1$ ($\sigma = -1$, resp.). Notice that M_{ij} is a k-noisy cluster if and only if there is some H^* in N_{ij}^+ or N_{ij}^- such that $|H^*| \leq k$ and $\texttt{Size}(M_{ij} - H^*) \geq \texttt{MinSize}$. Therefore, we have the following proposition.

Proposition 2. *M_{ij} is a k-noisy cluster if and only if the expression*

$$\max \left\{ \begin{array}{l} \max_{H \in N_{ij}^+, |H| \leq k} \{\texttt{Size}(M_{ij} - H)\}, \\ \max_{H \in N_{ij}^-, |H| \leq k} \{\texttt{Size}(M_{ij} - H)\} \end{array} \right\}$$

is at least `MinSize`.

Thus, to find all k-noisy clusters, it suffices to compute the expression in Proposition 2 for all $1 \leq i \leq j \leq n$. In the rest of this section, we show how to compute the expression

$$\max_{H \in N_{ij}^+, |H| \leq k} \{\texttt{Size}(M_{ij} - H)\} \tag{1}$$

for all $1 \leq i \leq j \leq n$. The counterpart can be computed similarly. Define, for all $1 \leq i \leq j \leq n$ and $0 \leq x \leq k$,

$$\texttt{V}(i, j, x) = \max_{H \in N_{ij}^+, |H| \leq x} \{\texttt{Size}(M_{ij} - H)\}.$$

Thus, Expression (1) equals $\texttt{V}(i, j, k)$. Let

$$\overline{\texttt{V}}(i, j, x) = \max_{H \in N_{ij}^+, |H| \leq x, m_j \notin H} \{\texttt{Size}(M_{ij} - H)\}.$$

To take care of the boundary conditions, for any $i, j < 1$ and $x < 0$, we set $\texttt{V}(i, j, x) = \overline{\texttt{V}}(i, j, x) = 0$. Then, we have

$$\texttt{V}(i, j, x) = \max\{\overline{\texttt{V}}(i, j, x), \texttt{V}(i, j-1, x-1)\}.$$

Now we show how to compute $\overline{\texttt{V}}(i, j, x)$. If m_j is of opposite orientations, $\overline{\texttt{V}}(i, j, x) = 0$. Suppose m_j is of same orientation. Let P be the set of matched substring pairs $m_p = (a_p, b_p, \ell_p)$ such that the following properties are satisfied:

- m_p is of same orientation,
- $\max(i, j - x - 1) \leq p \leq j - 1$,
- $b_p < b_j$, and
- m_p and m_j satisfy the distance requirement[‡].

For any $m_p \in P$, there are $j - p - 1$ matched substring pairs in between m_p and m_j. If there is a set X such that $|X| \leq x - (j - p - 1)$ and $M_{ip} - X$ satisfies the first three requirements of noisy cluster, then we can throw away a set X' with at most x matched substring pairs from M_{ij}, where $X' = (m_{p+1}, m_{p+2}, \ldots, m_{j-1}) \cup X$ so that $M_{ij} - X'$ also satisfies the first three requirements of noisy cluster. Therefore, we have the following proposition.

Proposition 3. *$\overline{\texttt{V}}(i, j, x)$ can be computed recursively as follows in terms of $\overline{\texttt{V}}(i, j', x')$ for some $j' < j$ and $x' \leq x$.* $\overline{\texttt{V}}(i, j, x) =$

$$\begin{cases} \max_{m_p \in P} \overline{\texttt{V}}(i, p, x - j + p + 1) + \texttt{Size}((m_j)) \\ \qquad \textit{if } P \neq \varnothing, \\ 0 \quad \textit{otherwise.} \end{cases}$$

[‡] The framework of the dynamic programming is applicable for any distance requirement.

Time and space complexity: Both the $\mathtt{V}$ and $\overline{\mathtt{V}}$ tables have kn^2 entries. Each $\overline{\mathtt{V}}$ entry is the maximum of at most k precomputed values. Therefore, computing the $\overline{\mathtt{V}}$ table takes $O(k^2n^2)$ time. Each $\mathtt{V}$ entry is the maximum of two precomputed values. Therefore, computing the $\mathtt{V}$ table takes $O(kn^2)$ time. With the computed $\mathtt{V}$ values, we can determine whether a given subsequence of M is a k-noisy cluster in $O(1)$ time. Together with the discussion on computing $\mathtt{W}(j)$, we have a dynamic programming algorithm for the max-min alignment problem which takes $O(k^2n^2)$ time.

For the space requirement, a straightforward implementation of the dynamic programming requires $O(kn^2)$ space. We can reduce the requirement to $O(k^2n)$ space. Consider the computation of $\mathtt{WI}(j)$ for some $1 \leq j \leq n$. We need to examine the values $\mathtt{V}(i,j,k)$ with $1 \leq i \leq j$. Computing $\mathtt{V}(i,j,k)$ requires the computation of $\overline{\mathtt{V}}(i,p,x)$ for some $0 \leq x \leq k$ and some p with $\max(i, j-x-1) \leq p < j$. There are at most $(k+1)$ such x values and at most $(k+1)$ such p values. In other words, the computation of $\mathtt{WI}(j)$ requires $O(k^2n)$ precomputed values, so only $O(k^2n)$ space is needed to store these values.

The Software Tool

By exploiting the MaxMinCluster algorithm, we have developed a software tool for locating conserved genes. The details are as follows.

Preprocessing: Finding matched substring pairs. We have constructed a program based on suffix tree to find those maximal uniquely matched substring pairs; the running time ranges from several to fifty minutes. We require that each pair of substrings reported has length at least α, for some $\alpha > 0$. The default value of α is 20 and 7 for aligning DNA sequences and protein sequences, respectively. As the suffix tree approach sometimes reports a large number of trivially isolated matched substring pairs which are almost certain to be noise, we remove such pairs before the clustering step. The default distance for a trivially isolated pair is set to 8000 and 2000 for DNA sequences and protein sequences, respectively. Note that we purposely make this distance bigger than necessary so as to avoid removing correct matched substring pairs.

Clustering: We apply the MaxMinCluster algorithm to find an alignment of the matched substring pairs reported before.

EXPERIMENTAL RESULTS AND DISCUSSION

In this section we report the quality of output of MaxMinCluster, and MUMmer-3, on aligning DNA sequences as well as translated protein sequences. We have implemented MaxMinCluster in a PC with 512M main memory and a 2.4GHz CPU. We have tested the program on data sets with n ranges from 30,000 to 100,000 and $k = 3$.

Notice that MUMmer-3 has two packages: NUCmer is for aligning DNA sequences and PROmer is for translated protein sequences. Both of them share a common clustering algorithm which has similar parameters (`Gap` and `MinSize`) as MaxMinCluster. In our experiments, we compare MaxMinCluster with this clustering algorithm on both DNA sequences and translated protein sequences using the same set of parameters.

The data sets we use include ten pairs of mouse and human chromosomes. For each pair of mouse and human chromosomes, a number of conserved genes have already identified by the biological community; details are published in GenBank (see the website http://www.ncbi.nlm.nih.gov/Homology). These published conserved gene will be the reference for our evaluation.

Measurement: We evaluate the output of a clustering algorithm from several perspectives: the coverage, the degree of coverage, and the sensitivity. For *coverage*, we count the number of published conserved genes that overlap with the clusters reported. In addition to coverage, we also measure, for each conserved gene covered, its portion that overlaps with the clusters reported, and we refer the average over all such genes as the *degree of coverage*.

Notice that high coverage alone may not imply high-quality output as an algorithm can simply output all matched substring pairs as a single cluster, thus achieving the highest possible coverage and degree of coverage. Therefore, we also measure the percentage of the clusters reported that actually cover the published conserved genes. This percentage is referred to as the *sensitivity*. It gives us an indicator of the accuracy, yet it may underestimate the actual accuracy as not all conserved genes have been identified. In other words, we expect a good algorithm to produce a set of clusters with high coverage and reasonable sensitivity.

Aligning DNA Sequences

We present the results on aligning DNA sequences of mouse and human chromosomes. The chromosomes used in our experiments are of length from 14

Exp. No.	Coverage		Degree of coverage		Sensitivity	
	MUMmer	MaxMinCluster	MUMmer	MaxMinCluster	MUMmer	MaxMinCluster
i	33 (65%)	39 (76%)	50%	60%	22%	24%
ii	99 (52%)	146 (76%)	40%	58%	25%	32%
iii	55 (54%)	71 (70%)	48%	66%	16%	16%
iv	26 (68%)	30 (79%)	57%	65%	28%	28%
v	46 (64%)	51 (71%)	49%	63%	34%	32%
vi	12 (39%)	18 (58%)	52%	52%	13%	16%
vii	21 (70%)	24 (80%)	43%	61%	55%	51%
viii	25 (54%)	35 (76%)	49%	60%	38%	47%
ix	13 (43%)	15 (50%)	28%	37%	36%	43%
x	61 (66%)	73 (78%)	62%	72%	35%	35%
Average	**58%**	**71%**	**48%**	**59%**	**30%**	**32%**

Table 1. The coverage, degree of coverage and the sensitivity of aligning the DNA sequences of mouse and human genomes for the setting `Gap` = 2000 and `MinSize` = 100. The figures in parenthesis are the percentage of coverage over the number of published conserved genes.

million to 65 million nucleotides. In finding matched substring pairs, we require that each pair of substrings reported has length at least 20. Substring pairs with length less than 20 are likely to be noise (Delcher et al., 1999). These substring pairs serve as the input data to our algorithm as well as MUMmer. The details of the data sets are given in the Appendix.

The findings: We have analyzed the data and observed that the average gap between adjacent matched substrings within a conserved gene ranges from 1000 to 3000 nucleotides, and the total length of matched substrings ranges from tens to hundreds. Therefore, we have performed experiments using different `Gap` and `MinSize` values. The results reveal that (i) MaxMinCluster reports more conserved genes than MUMmer in all settings, and (ii) a sensible setting for MUMmer, as well as for MaxMinCluster, is to let `Gap` = 2000 and `MinSize` = 100.

Table 1 shows the coverage, degree of coverage and sensitivity of the output for the setting `Gap` = 2000 and `MinSize` = 100. In each data set, MaxMinCluster has a higher coverage than MUMmer, the average improves from 58% to 71%. It is evident that the global selection criterion is effective. The output from MaxMinCluster also shows a higher degree of coverage than MUMmer. Regarding sensitivity, both software tools are reasonable, though MaxMinCluster is slightly higher.

In conclusion, MaxMinCluster, based on a global selection criterion and the noise tolerance feature, is able to produce a set of clusters of higher quality.

Different `Gap` and `MinSize` values: We have also tested the algorithms with different values of `Gap` and `MinSize`. Firstly, we fix `MinSize` to 100 and vary `Gap` for 100, 300, 600, 1000, 2000, and 3000 (see Figure 2). The results show that coverage, degree of coverage and sensitivity increase as the value of `Gap` increases, yet increasing the value of `Gap` beyond 2000 gives no significant improvement in coverage and sensitivity.

Secondly, we fix `Gap` to 2000 and vary `MinSize` for 60, 80, 100, 120, 140, 200 and 300 (see Figure 3). The results show that sensitivity increases as the value of `MinSize` increases, while coverage and degree of coverage decrease as expected. Furthermore, increasing the value of `MinSize` beyond 100 gives no significant improvement in sensitivity. Therefore, `Gap` = 2000 and `MinSize` = 100 is a sensible choice to obtain a high coverage while maintaining a high sensitivity.

Running time and memory requirement: We have measured the actual running time and memory requirement of MaxMinCluster and MUMmer-3 for the ten sets of data. As expected, MUMmer-3, based on a linear time heuristic, is much faster and requires much less memory. The average running time is about one second and the average memory requirement is about two megabytes. On the other hand, MaxMinCluster is much slower and requires more memory as it uses a quadratic time algorithm to solve the

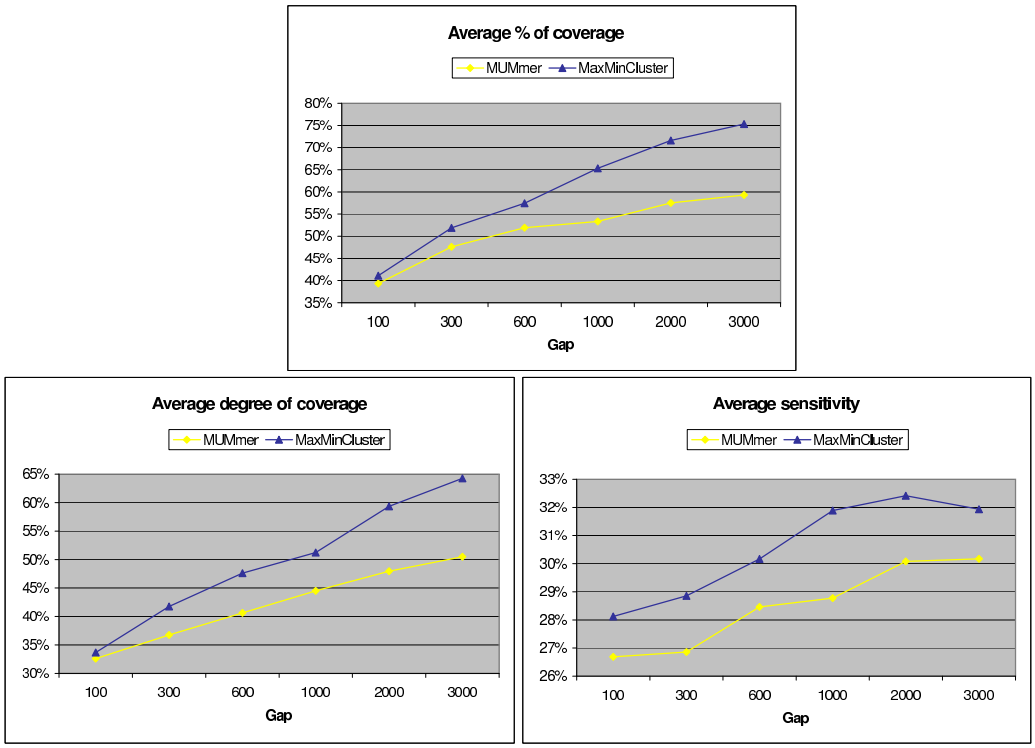

Fig. 2. The effect of varying the `Gap` value on coverage, degree of coverage and sensitivity (`MinSize` is fixed to 100) on aligning DNA sequences of mouse and human genomes.

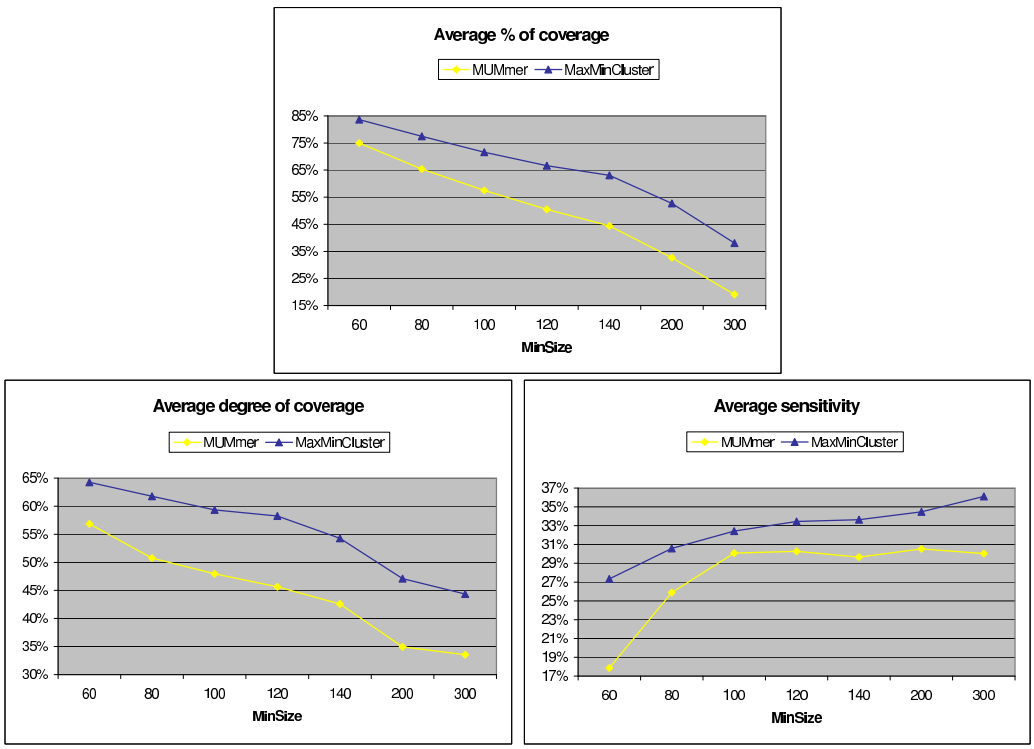

Fig. 3. The effect of varying the `MinSize` value on coverage, degree of coverage and sensitivity (`Gap` is fixed to 2000) on aligning DNA sequences of mouse and human genomes.

optimization problem optimally. Nevertheless, the actual running time of MaxMinCluster ranges from several seconds to ten minutes and the actual memory requirement ranges from tens to hundred megabytes; this is acceptable in most applications.

Aligning Translated Protein Sequences

Both MaxMinCluster and MUMmer-3 can be used to align the translated protein sequences of different genomes. Aligning on protein level is significant when the species are more distant because these species usually show much higher similarity on protein level than on nucleotide level.

We have run MaxMinCluster and MUMmer-3 on the translated protein sequences of the ten pairs of mouse and human chromosomes. We find that the translated protein sequences show much higher similarity and both MaxMinCluster and MUMmer-3 achieve very high coverage. In particular, based on the setting with matched substrings containing at least 7 amino acids ($\approx$ 20 nucleotides), `Gap` = 666 ($\approx$ 2000 nucleotides), and `MinSize` = 33 ($\approx$ 100 nucleotides), MaxMinCluster and MUMmer-3 both achieves an average coverage of 92%; the sensitivity is 32% and 30%, respectively.

The high-coverage result above triggers us to further analyze the conserved genes. We found that in protein level, the clusters residing in most conserved genes are quite obvious and this explains why both software tools can obtain a high coverage. To better understand the performance of MaxMinCluster and MUMmer-3 on aligning protein sequences, we perform experiments on aligning nine virus genomes, which do not show a high level of similarity as human and mouse. These virus genomes and the corresponding conserved genes have been published in the literature (Herniou et al., 2001). See the Appendix for more details. These genomes are of length from 100 thousand to 200 thousand nucleotides. We translate the sequences into amino acid sequences and use the suffix tree program to locate matched substring pairs with at least 3 amino acids.

We have run MaxMinCluster and MUMmer-3 using different parameters. A sensible setting is `MinSize` = 8, and the distance requirement has `Diff` = 5, and `Ratio` = 0.1. The coverage and sensitivity of MaxMinCluster and MUMmer-3 are shown in Figure 4. In general, the coverage of MaxMinCluster is higher than MUMmer, the average is 57% and 54% respectively; the average sensitivity of the two software tools is almost the same, which is about 66%.

Remarks: We have also aligned the DNA sequences of the nine virus genomes but find that most of them show very low similarity. Both MUMmer-3 and MaxMinCluster are able to uncover only very few conserved genes, indeed, for quite a number of virus pairs, no matched substring pairs can be found, not to mention conserved genes.

REFERENCES

Altschul, S. F., T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman (1997). Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. *Nucleic Acids Research 25*, 3389–3402.

Baillie, D. L. and A. M. Rose (2000). WABA success: A tool for sequence comparison between large genomes. *Genome Research 10*(8), 1071–1073.

Batzoglou, S., L. Pachter, J. Mesirov, B. Berger, and E. Lander (2000). Human and mouse gene structure: Comparative analysis and application to exon prediction. *Genome Research 10*, 950–958.

Buhler, J. (2001). Efficient large-scale sequence comparison by locality-sensitive hashing. *Bioinformatics 17*(5), 419–428.

Delcher, A. L., S. Kasif, R. D. Fleischmann, J. Peterson, O. White, and S. L. Salzberg (1999). Alignment of whole genomes. *Nucleic Acids Research 27*(11), 2369–2376.

Delcher, A. L., A. Phillippy, J. Carlton, and S. L. Salzberg (2002). Fast algorithms for large-scale genome alignment and comparison. *Nucleic Acids Research 30*(11), 2478–2483.

Frazer, K. A., L. Elnitski, D. M. Church, I. Dubchak, and R. C. Hardison (2003). Cross-species sequence comparisons: A review of methods and available resources. *Genome Research 13*, 1–12.

Gangloff, S., H. Zou, and R. Rothstein (1996). Gene conversion plays the major role in controlling the stability of large tandem repeats in yeast. *EMBO Journal 15*, 1715–1725.

Griffiths, A. J., J. H. Miller, and D. T. Suzuki (2000). *Introduction to Genetic Analysis*. W.H. Freeman & Company.

Gusfield, D. (1997). *Algorithms on Strings, Trees, and Sequences: Computer Science and Computa-*

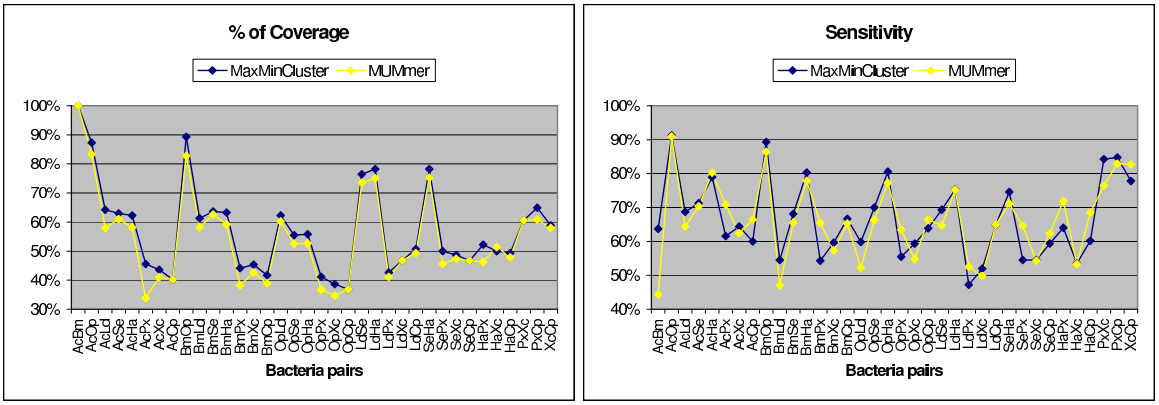

Fig. 4. The coverage and sensitivity of aligning protein sequences of virus genomes for the setting `MinSize` = 8, `Diff` = 5, and `Ratio` = 0.1.

tional Biology. Cambridge University Press.

Herniou, E. A., T. Luque, X. Chen, J. M. Vlak, D. Winstanley, J. S. Cory, and D. R. O'Reilly (2001). Use of whole genome sequence data to infer baculovirus phylogeny. *Journal of Virology 75*(17), 8117–8126. http://www.bio.ic.ac.uk/research/dor/research/eah.

Kurtz, S., A. Phillippy, A. Delcher, M. Smoot, M. Shumway, C. Antonescu, and S. Salzberg (2004). Versatile and open software for comparing large genomes. *Genome Biology 5*(2).

Morgenstern, B. (1999). Dialign 2: Improvement of the segment-to-segment approach to multiple sequence alignment. *Bioinformatics 15*, 211–218.

Morgenstern, B., K. Frech, D. Dress, and T. Werner (1998). Dialign: Finding local similarities by multiple sequence alignment. *Bioinformatics 14*, 290–294.

Mushegian, A. R. and E. V. Koonin (1996). Sequence analysis of eukaryotic developmental proteins: Ancient and novel domains. *Genetics 144*, 817–828.

Needleman, S. B. and C. D. Wunsch (1970). A general method applicable to the search for similarities in the amino acid sequences of two proteins. *Journal of Molecular Biology 48*, 443–453.

Nieduszynski, C. A., J. Murray, and M. Carrington (2002). Whole-genome analysis of animal a- and b-type cyclins. *Genome Biology 3*(12).

Plasterk, R. H. A. (1993). Molecular mechanisms of transposition and its control. *Cell 74*, 781–786.

Schwartz, S., Z. Zhang, K. A. Frazer, A. Smit, C. Riemer, J. Bouck, R. Gibbs, R. Hardison, and W. Miller (2000). Pipmaker - a web server for aligning two genomic dna sequences. *Genome Research 10*(4), 577–586.

Vincens, P., L. Buffat, C. Andre, J. Chevrolat, J. Boisvieux, and S. Hazout (1998). A strategy for finding regions of similarity in complete genome sequences. *Bioinformatics 14*, 715–725.

APPENDIX

Experiment No.	Mouse Chr No.	Human Chr No.	# of Input Pairs	# of Published Conserved Genes
i	2	15	96,473	51
ii	7	19	52,394	192
iii	9	11	93,855	101
iv	14	8	38,818	38
v	15	22	71,613	72
vi	16	16	66,536	31
vii	16	22	61,200	30
viii	17	16	29,001	46
ix	17	19	56,236	30
x	19	11	29,814	93

Table 2. Details of the data sets for mouse and human chromosomes.

Experiment No.	Virus	Virus	# of Input Pairs	# of Published Conserved Genes
i	AcMNPV	BmNPV	35,166	134
ii	AcMNPV	OpMNPV	59,949	126
iii	AcMNPV	LdMNPV	65,227	95
iv	AcMNPV	SeMNPV	66,898	100
v	AcMNPV	HaSNPV	64,291	98
vi	AcMNPV	PxGV	50,093	68
vii	AcMNPV	XcGV	85,443	78
viii	AcMNPV	CpGV	61,195	72
ix	BmNPV	OpMNPV	58,657	122
x	BmNPV	LdMNPV	63,086	93
xi	BmNPV	SeMNPV	66,448	99
xii	BmNPV	HaSNPV	63,939	98
xiii	BmNPV	PxGV	49,837	68
xiv	BmNPV	XcGV	84,110	75
xv	BmNPV	CpGV	60,708	72
xvi	OpMNPV	LdMNPV	75,906	98
xvii	OpMNPV	SeMNPV	63,261	101
xviii	OpMNPV	HaSNPV	59,125	95
xix	OpMNPV	PxGV	47,901	68
xx	OpMNPV	XcGV	77,986	75
xxi	OpMNPV	CpGV	59,715	76
xxii	LdMNPV	SeMNPV	62,545	102
xxiii	LdMNPV	HaSNPV	57,618	92
xxiv	LdMNPV	PxGV	46,668	68
xxv	LdMNPV	XcGV	75,350	77
xxvi	LdMNPV	CpGV	57,045	75
xxvii	SeMNPV	HaSNPV	64,980	101
xxviii	SeMNPV	PxGV	50,253	68
xxix	SeMNPV	XcGV	84,152	76
xxx	SeMNPV	CpGV	60,905	75
xxxi	HaSNPV	PxGV	49,146	67
xxxii	HaSNPV	XcGV	83,715	74
xxxiii	HaSNPV	CpGV	59,231	71
xxxiv	PxGV	XcGV	81,020	99
xxv	PxGV	CpGV	59,733	97
xxvi	XcGV	CpGV	63,258	107

Table 3. Details of the data sets for virus genomes.