Efficient Algorithms for Optimizing Whole Genome Alignment with Noise

TW Lam*, N Lu, HF Ting, Prudence WH Wong, and SM Yiu

Department of Computer Science,
University of Hong Kong, Hong Kong
{twlam,nlu,hfting,whwong,smyiu}@cs.hku.hk

Abstract. Given the genomes (DNA) of two related species, the whole genome alignment problem is to locate regions on the genomes that possibly contain genes conserved over the two species. Motivated by existing heuristic-based software tools, we initiate the study of optimization problems that attempt to uncover conserved genes with a global concern. Another interesting feature in our formulation is the tolerance of noise. Yet this makes the optimization problems more complicated; a brute-force approach takes time exponential in the noise level. In this paper we show how an insight into the problem 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 algorithms on a PC. It is exciting to see that when compared with the most popular whole genome alignment software (MUMMER) on real data sets, the new algorithms consistently uncover more conserved genes (that have been published by GenBank), while preserving the preciseness of the output.

1 Introduction

Given the genomes (DNA) of two related species, the whole genome alignment problem[2, 6, 9] is to identify potential regions that contain genes conserved over the two species. This problem has attracted a lot of attention in the past few years and a number of software tools have been developed [1, 3–5, 7, 8, 10, 11].

Related species (such as mouse and human) often have a lot of genes conserved, i.e., having the same functionality. Though a pair of conserved genes rarely contain the same entire sequence (probably due to mutations), they share a lot of short common substrings and some of these substrings are indeed unique to this pair of genes. 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. Of course, not every pair of matched substrings correspond to a pair of conserved genes; 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. In other words, a pair of conserved genes are likely corresponding 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* and a maximal collection of clusters

* This research was supported in part by Hong Kong RGC Grant HKU-7042/02E.

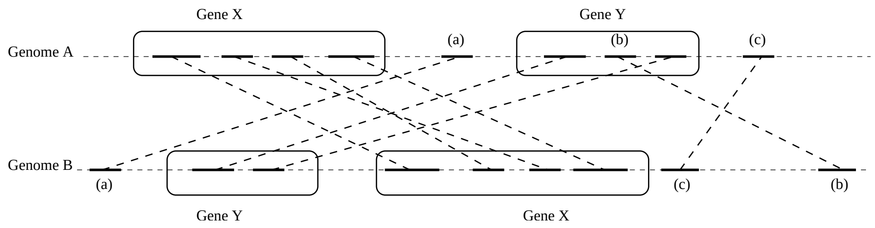

Fig. 1. Among all pairs of matched substrings, a number of them do not originate from conserved gene pairs. See (a), (b), (c) for examples.

an *alignment*. At first glance, the problem of finding conserved genes is a simple clustering problem. Such a clustering approach has been used in practice [5], 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 does not have much control over the quality of the alignment. It is more desirable to find an alignment that satisfies some instinctive 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. Based on k-noisy clusters, we first investigate the optimization problem of finding an alignment that maximizes the size of the smallest cluster. Another optimization problem we have studied is finding an alignment that minimizes the number of clusters. Intuitively, both criteria do 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 global selection criterion would trim away those small clusters, retaining the preciseness of the output.

The bottleneck in solving these optimization problems 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. 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 checking a pair of regions on average, and $O(k^2n^2)$ time for checking all pairs. This improvement allows us to solve either 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 these algorithms, 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.

We have implemented the new algorithms on a PC and compared them with a heuristic-based software called MUMMER (which is the most popular whole genome alignment software) on seven different sets of human and mouse chromosomes. It is encouraging to see that the new algorithms consistently achieve better coverage while preserving preciseness. They are able to uncover 10% to 25% more conserved genes (that are currently known to the biological community) in each case. See Table 2 in Section 4 for details. It is worth-mentioning that both selection criteria show improvement even if the noisy level is kept to zero.

1.1 Problem Definition

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 strands of the same orientation or 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 $\texttt{Gap}$ and $\texttt{MinSize}$ be two predefined positive constants.

Noisy clusters: A segment C 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 C of M is a *k-noisy cluster* if we can remove at most k elements from C, 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 have $|a_p - a_q| \leq \texttt{Gap}$ and $|b_p - b_q| \leq \texttt{Gap}$. This is called the distance requirement.
4. $\texttt{Size}(S)$, defined to be $\sum_{m_i \in S} \ell_i$, is at least $\texttt{MinSize}$. 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 replace some cluster(s) in A with a bigger cluster).

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 $\mathtt{Size}(C - X_o)$.

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^* a max-min optimal alignment of M and $\min_{C\in A^*} w(C)$ the weight of A^*.

Min-cardinality alignment problem: Given a set M of pairs of matched substrings, we want to find an alignment A^* of M such that $|A^*| = \min_{A\in\Sigma} |A|$, where $|A|$ denotes the number of clusters in the alignment A and Σ denotes the set of all alignments of M. We call A^* a min-cardinality optimal alignment of M.

2 The max-min alignment problem

2.1 The dynamic programming algorithm

In this section, we describe a dynamic programming algorithm for the max-min alignment problem. Recall that the input is a sequence $M = (m_1, m_2, \dots, m_n)$ and Σ denotes the set of all alignments of M. We want to find a max-min optimal alignment A^* of M, i.e., $\min_{C\in A^*} w(C) = \max_{A\in\Sigma} \min_{C\in A} w(C)$. The dynamic programming algorithm computes A^* incrementally, by considering the sequences (m_1), (m_1, m_2), ..., $(m_1, m_2, \dots, m_j)$ and so on. For $1 \le j \le n$, let Φ_j be the set of all possible k-noisy clusters of M whose elements are in $(m_1, m_2, \dots, m_j)$. Let Σ_j be 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 $\Phi_n = \Phi$ and $\Sigma_n = \Sigma$. Thus, $\mathtt{W}(n)$ is the weight of the max-min 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 $\mathtt{WI}(j) = \max_{A\in\Gamma_j} (\min_{C\in A} w(C))$, and $\mathtt{WE}(j) = \max_{A\in\Sigma_j-\Gamma_j} (\min_{C\in A} w(C))$. 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 $(m_i, m_{i+1}, \dots, m_j)$ is a k-noisy cluster, for all $1 \le$

$i \leq j$. To ease our discussion, we denote the segment $(m_i, m_{i+1}, \ldots, m_j)$ as M_{ij}. 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. Let i^* be the largest position in S_j. The following lemma gives a recurrence formula for $\mathtt{WI}(j)$ and $\mathtt{WE}(j)$ in terms of $\mathtt{W}(j')$ and $\mathtt{WI}(j')$ with $j' < j$.

Lemma 1. *Assume* $\mathtt{W}(0) = \mathtt{WI}(0) = \mathtt{WE}(0) = 0$. *For any* $j \geq 1$,

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

Proof. (1) For every $i \in S_j$, we want to determine the best alignment $A^* \in \Sigma_j$ that contains M_{ij}, i.e., the minimum weight of the clusters in A^* is the maximum among all such alignments. This alignment is the union of $\{M_{ij}\}$ and the best alignment in Σ_{i-1}. If $\mathtt{W}(i-1) \neq 0$, $\min_{C \in A^*} w(C) = \min(\mathtt{W}(i-1), w(M_{ij}))$. If $\mathtt{W}(i-1) = 0$, $\min_{C \in A^*} w(C) = w(M_{ij})$. Therefore, Statement (1) follows.

(2) Suppose $S_j \neq \varnothing$. Let A be any alignment in $\Sigma_j - \Gamma_j$, and h be the largest index of the matched substring pair that is contained in the clusters of A. Notice that we must have $i^* \leq h \leq j-1$, otherwise, $A \cup \{M_{i^*j}\}$ is an alignment which contradicts that A is maximal. Therefore, we have $\mathtt{WE}(j) = \max_{i^* \leq h \leq j-1} \mathtt{WI}(h)$. Suppose $S_j = \varnothing$. Then $\Gamma_j = \varnothing$. Therefore, $\mathtt{WE}(j) = \mathtt{W}(j-1)$.

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

Time complexity: Suppose that we have a preprocessing to find all k-noisy clusters and their weights (i.e., Step 1 of Algorithm 1) 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 1. 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 1 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. In the next section, we give an algorithm to find all k-noisy clusters with time $f(n) = O(k^2n^2)$. In other words, the whole algorithm runs in $O(k^2n^2)$ time.

2.2 Finding the k-noisy clusters

In this section, we show how to find all k-noisy clusters and determine their weights in $O(k^2n^2)$ 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

Algorithm 1 The dynamic programming algorithm for the max-min alignment problem.

1. For each subsequence M_{ij} of M with $1 \le i \le j \le n$, determine whether it is a k-noisy clusters and compute its weight if so. For any $1 \le j \le n$, 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 Lemma 1 part (1) in terms of $\mathtt{W}(i-1)$ and $w(M_{ij})$, and set $\mathtt{WI}(j)$ to be the maximum of these values.
   (b) Compute $\mathtt{WE}(j)$ according to Lemma 1 part (2) in terms of $\mathtt{W}(j-1)$ and $\mathtt{WI}(h)$ for $i^* \le h \le j-1$.
   (c) Set $\mathtt{W}(j) = \max\{\mathtt{WI}(j), \mathtt{WE}(j)\}$.

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 $1 \le i \le j \le n$. 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 three requirements of a k-noisy cluster. We observe that the first two requirements of a k-noisy cluster are local concerns while the third requirement (i.e., the size requirement) is a global concern in the following sense. If the segment $M_{ij'}$ does not satisfy the first two 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 two 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.).

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

$$\max \left\{ \begin{array}{l} \max_{H \in N^+_{ij}, |H| \le k} \{\mathtt{Size}(M_{ij} - H)\}, \\ \max_{H \in N^-_{ij}, |H| \le k} \{\mathtt{Size}(M_{ij} - H)\} \end{array} \right\}$$

is at least `MinSize`*.*

Proof. The lemma follows from definition.

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

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

for all $1 \le i \le j \le n$. The counterpart can be computed similarly. Define, for all $1 \le i \le j \le n$ and $0 \le x \le k$,
$$\texttt{V}(i,j,x) = \max_{H \in N_{ij}^+, |H| \le x} \{\texttt{Size}(M_{ij} - H)\}.$$

Thus, Expression (1) equals $\texttt{V}(i,j,k)$. Let
$$\overline{\texttt{V}}(i,j,x) = \max_{H \in N_{ij}^+, |H| \le x, m_j \notin H} \{\texttt{Size}(M_{ij} - H)\}.$$

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

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 the 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: (i) m_p is of same orientation, (ii) $\max(i, j-x-1) \le p \le j-1$, (iii) $b_p < b_j$, (iv) m_p and m_j satisfy the distance requirement, and (v) m_p and m_j satisfy the consecutive requirement. Then we have the following lemma.

Lemma 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 $x' \le x$.*
$$\overline{\texttt{V}}(i,j,x) = \begin{cases} \max_{m_p \in P} \overline{\texttt{V}}(i,p,x-j+p+1) + \texttt{Size}((m_j)) & \text{if } P \ne \varnothing, \\ 0 & \text{otherwise.} \end{cases}$$

Proof. For any $m_p \in P$, the number of matched substring pairs in between m_p and m_j is $j-p-1$. If there is a set X such that $|X| \le x-(j-p-1)$ and $M_{ip} - X$ satisfies the first two 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 two requirements of noisy cluster. Therefore, $\overline{\texttt{V}}(i,j,x) = \max_{m_p \in P} \overline{\texttt{V}}(i,p,x-j+p+1) + \texttt{Size}((m_j))$. On the other hand, if $P = \varnothing$, $\overline{\texttt{V}}(i,j,x) = 0$.

Time and space complexity: Both the $\texttt{V}$ and $\overline{\texttt{V}}$ tables have kn^2 entries. Each $\overline{\texttt{V}}$ entry is the maximum of at most k precomputed values. Therefore, computing the $\overline{\texttt{V}}$ table takes $O(k^2n^2)$ time. Each $\texttt{V}$ entry is the maximum of two precomputed values. Therefore, computing the $\texttt{V}$ table takes $O(kn^2)$ time. With the computed $\texttt{V}$ values, we can determine whether a given subsequence of M is a k-noisy cluster in constant time. Together with the discussion in Section 2.1, 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 $\texttt{WI}(j)$ for some $1 \le j \le n$. We need to examine the values $\texttt{V}(i,j,k)$ with $1 \le i \le j$. Computing $\texttt{V}(i,j,k)$ requires the computation of $\overline{\texttt{V}}(i,p,x)$ for some $0 \le x \le k$ and some p with $\max(i, j-x-1) \le p < j$. There are at most $(k+1)$ such x values and at most $(k+1)$ such p values. In other words, the computation of $\texttt{WI}(j)$ requires $O(k^2n)$ precomputed values, so only $O(k^2n)$ space is needed to store these values.

3 The min-cardinality alignment problem

In this section, we describe a dynamic programming algorithm for the min-cardinality alignment problem. This algorithm makes use of the algorithm described in Section 2.2 to determine whether a subsequence of M is a k-noisy cluster. Recall the definition Σ_j, Γ_j and $\Sigma_j - \Gamma_j$ in the previous section. For any alignment A, let $|A|$ denote the number of clusters in A. For $1 \leq j \leq n$, let $\mathtt{U}(j) = \min_{A \in \Sigma_j} |A|$. To take care of the boundary conditions, we set $\mathtt{U}(0) = \mathtt{UI}(0) = \mathtt{UE}(0) = \infty$. Note that $\mathtt{U}(n)$ is the number of clusters in the min-cardinality optimal alignment of M. To find $\mathtt{U}(j)$, we consider two cases according to whether m_j is included in some clusters in the alignment. We define $\mathtt{UI}(j) = \min_{A \in \Gamma_j} |A|$, and $\mathtt{UE}(j) = \min_{A \in \Sigma_j - \Gamma_j} |A|$. Then, we have $\mathtt{U}(j) = \min\{\mathtt{UI}(j), \mathtt{UE}(j)\}$. The following lemma gives a recursive formula for $\mathtt{UI}(j)$ and $\mathtt{UE}(j)$.

Lemma 4. *Let i' be the smallest value such that $M_{i'j}$ is a k-noisy cluster, and let i^* be the largest such value.*

1. $\mathtt{UI}(j) = \begin{cases} \infty & \textit{if there is no such } i', \\ \mathtt{U}(i'-1)+1 & \textit{if } \mathtt{U}(i'-1) \neq \infty, \\ 1 & \textit{if } \mathtt{U}(i'-1) = \infty. \end{cases}$
2. $\mathtt{UE}(j) = \begin{cases} \mathtt{UE}(j) = \min_{h \in [i^*, j-1]} \mathtt{UI}(h) & \textit{if } i^* \textit{ exists}, \\ \mathtt{U}(j-1) & \textit{otherwise}. \end{cases}$

Proof. The proof is similar to Lemma 1 and will be given in the full paper.

Similar to the Max-min algorithm, we can show that the Min-cardinality algorithm takes $O(k^2n^2)$ time and $O(k^2n)$ space.

4 Experiments

We have implemented the Max-min and the Min-cardinality algorithms in a PC with 512M memory and a 2.4GHz CPU. The actual running time of the programs depend on the noise level k and the number of input pairs n. We have tested the programs on some real data sets with n ranging from 30,000 to 70,000 and with $k = 0$ and $k = 3$. The running times are reasonable and range from one hour to several hours.

We compare the quality of the output of our algorithms with that of MUMMER[5], the most popular whole genome alignment software. We have chosen 7 pairs of mouse and human chromosomes as our testing data. For each pair of chromosomes, the biological community has already identified a number of conserved genes; details are published in GenBank**. The chromosomes we used in the experiments are of length about 30 million. For each pair of chromosomes, we identify the set of uniquely matched substring pairs with length at least 20 as input to the algorithms. Substring pairs with length less than 20 are likely to be noise [4]. The details of the data sets are given in Table 1.

** GenBank is the largest public database of DNA sequences and is maintained by the National Center for Biotechnology Information (NCBI), http://www.ncbi.nlm.nih.gov/Homology.

Experiment No.	Mouse Chr No.	Human Chr No.	# of Input Pairs	# of Published Conserved Gene Pairs
1	7	19	52,394	192
2	15	22	71,613	72
3	16	16	66,536	31
4	16	22	61,200	30
5	17	16	29,001	46
6	17	19	56,236	30
7	19	11	29,814	93

Table 1. Details of Data Sets

We measure the quality of output from two perspectives: the coverage and the preciseness. For coverage, we count the number of published gene pairs that are covered by the clusters reported by the algorithms. However, high coverage alone may not imply a high quality in the output as one can simply output every matched substring pair as a single cluster, thus achieving the highest possible coverage. Therefore, we also consider the percentage of output clusters that actually cover the conserved gene pairs. This percentage is referred to as the preciseness of the output. In other words, a good algorithm should produce a set of output clusters with high coverage and high preciseness.

Table 2 shows the coverage of the output from different algorithms in different test cases. In general, both the Max-min and Min-cardinality algorithms have a higher coverage than MUMMER even for $k = 0$. It implies that the global selection criteria are effective. If the noise level increases to 3, the coverage of the output from our algorithms has visible improvement. It shows that the noise tolerance feature does enable more conserved gene pairs to be uncovered. Combining the global selection criteria and the noise tolerance feature can increase the coverage of the output by 10% to 25% compared with that for MUMMER. Also, the set of gene pairs uncovered by the Max-min and the Min-cardinality algorithms (for both $k = 0$ and $k = 3$) is the superset of that uncovered by MUMMER. On the other hand, the output produced by the Max-min and the Min-cardinality algorithms have exactly the same coverage in all cases.

Experiment No.	MUMMER	Max-min ($k = 0$)	Max-min ($k = 3$)	Min-cardinality ($k = 0$)	Min-cardinality ($k = 3$)
1	137	148	157	148	157
2	52	59	62	59	62
3	21	22	24	22	24
4	24	25	28	25	28
5	30	38	38	38	38
6	18	18	21	18	21
7	73	79	79	79	79

Table 2. Coverage of Output of Different Algorithms (In the experiments, we set `Gap` = 2000, `MinSize` = 65 for all programs including MUMMER. We have tried the default parameters of MUMMER, but the coverage of the output is much poorer.)

Table 3 shows the preciseness of the output from different algorithms in the test cases. In most cases, the output from the Max-min and Min-cardinality algorithms show a slightly higher preciseness than that of MUMMER. Again, the output from the Max-min and Min-cardinality algorithms show very similar preciseness.

Experiment No.	MUMMER	Max-min ($k = 0$)	Max-min ($k = 3$)	Min-cardinality ($k = 0$)	Min-cardinality ($k = 3$)
1	25.7%	24.3%	25.8%	24.3%	26.8%
2	23.3%	24.8%	24.4%	24.8%	24.5%
3	10.1%	12.2%	12.5%	12.2%	12.7%
4	22.9%	25.3%	25.4%	25.3%	22.8%
5	20.4%	24.1%	25.0%	24.1%	23.6%
6	12.5%	13.2%	14.9%	13.2%	14.3%
7	27.5%	27.9%	27.4%	27.9%	28.5%

Table 3. The Preciseness of Output of Different Algorithms

To conclude, based on the global selection criteria and the noise tolerance feature, both of our algorithms are able to produce a set of higher quality output clusters (ie, with higher coverage and similar preciseness) than MUMMER.

References

1. David L. Baillie and Ann M. Rose. Waba success: A tool for sequence comparison between large genomes. *Genome Research*, 10(8):1071–1073, 2000.
2. S. Batzoglou, L. Pachter, J.P. Mesirov, B. Berger, and E.S. Lander. Human and mouse gene structure: Comparative analysis and application to exon prediction. *Genome Research*, 10:950–958, 2000.
3. Jeremy Buhler. Efficient large-scale sequence comparison by locality-sensitive hashing. *Bioinformatics*, 17(5):419–428, 2001.
4. Arthur L. Delcher, Simon Kasif, Robert D. Fleischmann, Jeremy Peterson, Owen White, and Steven L. Salzberg. Alignment of whole genomes. *Nucleic Acids Research*, 27(11):2369–2376, 1999.
5. Arthur L. Delcher, Adam Phillippy, Jane Carlton, and Steven L. Salzberg. Fast algorithms for large-scale genome alignment and comparison. *Nucleic Acids Research*, 30(11):2478–2483, 2002.
6. Kelly A. Frazer, Laura Elnitski, Deanna M. Church, Inna Dubchak, and Ross C. Hardison. Cross-species sequence comparisons: A review of methods and available resources. *Genome Research*, 13:1–12, 2003.
7. B. Morgenstern. Dialign 2: Improvement of the segment-to-segment approach to multiple sequence alignment. *Bioinformatics*, 15:211–218, 1999.
8. B. Morgenstern, K. Frech, D. Dress, and T. Werner. Dialign: Finding local similarities by multiple sequence alignment. *Bioinformatics*, 14:290–294, 1998.
9. Conrad A Nieduszynski, James Murray, and Mark Carrington. Whole-genome analysis of animal a- and b-type cyclins. *Genome Biology*, 3(12), 2002.
10. Scott Schwartz, Zheng Zhang, Kelly A. Frazer, Arian Smit, Cathy Riemer, John Bouck, Richard Gibbs, Ross Hardison, and Webb Miller. Pipmaker - a web server for aligning two genomic dna sequences. *Genome Research*, 10(4):577–586, 2000.
11. P. Vincens, L. Buffat, C. Andre, J.P. Chevrolat, J.F. Boisvieux, and S. Hazout. A strategy for finding regions of similarity in complete genome sequences. *Bioinformatics*, 14:715–725, 1998.