A Mutation-Sensitive Approach for Locating Conserved Gene Pairs between Related Species

HL Chan† TW Lam† * WK Sung‡ Prudence WH Wong† SM Yiu†

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

‡ Department of Computer Science, National University of Singapore, Singapore
ksung@comp.nus.edu.sg

Abstract

This paper proposes a new approach for solving the whole genome alignment problem. Our approach is based on a new structural optimization problem (called the MUM Selection Problem) related to mutations via reversals and transpositions. We have devised a practical algorithm for this optimization problem and have evaluated the algorithm using 15 pairs of human and mouse chromosomes. The results show that our algorithm is both effective and efficient. More specifically, our algorithm can reveal 91% of the conserved gene pairs that have been reported in the literature. When compared to existing software MUMmer [7, 16] and MaxMinCluster [12], our algorithm uncovers 15% and 7% more genes on average, respectively. The sensitivity of our algorithm is also slightly higher. The paper concludes with a remark on the computational hardness of the MUM Selection Problem.

Keywords: whole genome alignment, conserved gene pairs, mutation

1. Introduction

Given the genomes of two related species, one important task is to uncover and locate the conserved genes, i.e., genes sharing similar functions [3, 15, 18]. This task is non-trivial as most parts of a genome are non-coding areas and the information of genes of each genome is often not available. In this paper, we propose a more effective algorithm for identifying locations on the genomes that correspond to pairs of conserved genes.

In the process of uncovering conserved genes, Delcher et al. [6] had given a very useful observation: A pair of conserved genes rarely contain the same entire sequence, but they share a lot of short common substrings that are unique to this pair of genes. Thus, the first step to locate conserved gene pairs is to identify pairs of matched substrings that appear uniquely in both genomes. This can be done in linear time using a suffix tree [6]. Such pairs of matched substrings are called the MUM pairs. However, not every MUM pair corresponds to a pair of conserved genes; there are often a lot of noisy MUM pairs, originating from intergenic regions as well as from unrelated genes. The key step is how to select the right MUM pairs.

Different approaches have been proposed to select the right MUM pairs. MUMmer-1 [6] simply selects the largest subset of MUM pairs that have the same ordering in both genomes. This is based on the assumption that two related species should preserve the ordering of most conserved genes. MUMmer-2 [7] and MUMmer-3 [16] select MUM pairs that are close together, i.e., forming a cluster. Intuitively, a pair of conserved genes should introduce a group of MUM pairs that are close together, while noisy MUM pairs are random in nature and tend to be separated. In practice, MUMmer-2 and MUMmer-3 show significant improvement over their predecessor. MaxMinCluster [12] refines the clustering approach by allowing a small degree of noise. From the biological point of view, the conserved genes of two related genomes would not occur in a random ordering in each genome. The difference in the orderings is most likely caused by the mutations that have occurred between the two concerned species during the evolution. In other words, the difference in the orderings of the selected MUM pairs in the two genomes could be related to the mutations that have occurred. For related species, the number of such mutations would be small. One should select those MUM pairs whose difference in the orderings can be explained by a few mutations. Using this idea, we propose a new approach to select MUM pairs based on structural optimization. Our approach has been shown to be significantly more effective than the previous ones when tested with real data.

In fact, the study of mutations between two related species is not a new topic. A closely related problem is the genome rearrangement problem [1, 2, 4, 8, 10, 11]. We consider the signed version of this problem: The conserved gene pairs and their locations in two genomes are given in advance. Each conserved gene pair is represented by a

* This research was supported in part by Hong Kong RGC Grant HKU-7042/21E.

unique signed integer, and the ordering of the genes in the two genomes is represented by two permutations of signed integers π_1 and π_2. Given π_1 and π_2, the problem asks for the smallest number of mutations needed to transform π_1 into π_2. The genome rearrangement problem has been studied intensively. There are several results on mutations restricted to reversals only (e.g. $[1, 4, 11]$), while in some other studies, transpositions and reversed-transpositions are also included (e.g. $[2, 8, 10]$). The genome rearrangement problem is solvable in polynomial time when the only mutations are reversals $[4, 11]$. Yet when transpositions and reversed-transpositions are also allowed, the complexity of the problem is unknown. It is believed that the problem is NP-hard, and a 2-approximation algorithm has been proposed [10]. As for the unsigned version of the genome rearrangement problem, see [14] for a more complete discussion of the problem.

Let us switch the context back to our problem of selecting the right MUM pairs that correspond conserved gene pairs. Our problem can be regarded as a generalization of the genome rearrangement problem. Note that a subset of MUM pairs induce two signed permutations σ_1 and σ_2, according to the ordering of these MUM pairs in the two given genomes. To select the right MUM pairs, we try to find a subset of MUM pairs with maximum total length, such that the induced permutations can be transformed to each other by a few mutations. In this paper, a mutation is considered to be a reversal, transposition or reversed transposition. We limit the number of mutations to a small constant k because for related species, there should only be a few mutations undertaken, leading to their positional difference. Obviously, if we make no restriction on the number of the mutations, all the MUM pairs will be selected. By restricting the number of mutations, we can effectively filter the MUM pairs that are noise, while preserving those that capture conserved gene pairs. We call this problem the MUM Selection Problem. Note that the genome rearrangement problem involving reversals, transpositions and reversed transpositions can be reduced to the MUM Selection Problem. As the former is believed to be NP-hard, the MUM Selection Problem is even more likely to be NP-hard.

Main result: Technically speaking, this paper gives an efficient algorithm called MSS (mutation-sensitive selection) which, given a set of MUM pairs and an integer $k \geq 0$, selects a subset of MUM pairs such that the induced permutation σ_1 can be transformed to σ_2 by a sequence of at most k mutations. The subset of MUM pairs reported by this algorithms often has a total length very close to the maximum possible length. In fact, from a theoretical viewpoint, we are able to prove that even in the worst case, the subset selected by MSS has a total length at least $1/(3k+1)$ times the maximum weighted subset.

Based on MSS and a simple filtering, we have implemented a new software for locating conserved gene pairs. The software performs very well in practice. We have tested the software with fifteen sets of real data (which are different mouse and human chromosomes), and its performance is compared with that of MUMmer-3 and MaxMinCluster. The average performance is shown in the following table (see Section 4 for details). Our software is found to be effective in terms of both coverage and sensitivity (for coverage, we count the percentage of published gene pairs that are reported by the software; for sensitivity, we count the percentage of the reported MUM pairs that are known to reside in a conserved gene pair; note that sensitivity is an estimate as not all conserved gene pairs have been uncovered by the biological community).

	Coverage	sensitivity
MUMmer-3	76.6%	26.5%
MaxMinCluster	84.3%	27.4%
MSS ($k = 4$)	91.3%	29.3%

Remarks: Though the algorithm MSS performs very well in practice, yet from a computational viewpoint, MSS may not report the optimal subset of MUM pairs, satisfying the maximum total length requirement of the MUM Selection Problem. The complexity of the MUM Selection Problem is still open, but we believe that it is unlikely the problem can be solved in polynomial time, and in this paper, we are able to show that if one puts a simple restriction to the mutation operations, then the MUM Selection Problem can be proven to be NP-hard. Details are as follows. In reality the mutations between two related species do not occur arbitrarily. This motivates us to consider restrictions on the relationship of the mutations. In particular, we say that a sequence of mutations is "atomic" if any MUM pair involved in a mutation will not be involved in any other mutation (formal definition will be given in Section 5). We show that if the MUM Selection Problem is NP-hard when the mutation operations must be atomic.

If we confine our attention to mutation by reversals only, the complexity of the MUM Selection Problem is still open. Yet if the mutations are atomic, we can derive an algorithm to solve the problem optimally in $O(n^2 \log n)$ time. However, when used to locate conserved gene pairs, the performance of this algorithm on the above-mentioned data sets is only comparable to MUMmer-3, i.e., not as good as MaxMinCluster or MSS. We believe that it makes more sense to consider mutations including transpositions and inverted transpositions.

Organization of the paper: In Section 2, we give the formal problem definition of the MUM Selection Problem. In Section 3, we present the algorithm MSS. In Section 4, we give the details of the new software for identify conserved gene pairs, as well as the results of experiments with the real data. In Section 5, we show that the MUM Selection Problem is NP-hard if the mutations are atomic.

2. The MUM Selection Problem

The input: Given two genomes G_1 and G_2 with n MUM pairs, we represent the MUM pairs as two sequences of n distinct characters, denoted $A = a_1 a_2 \cdots a_n$ and $B = b_1 b_2 \cdots b_n$, respectively, where each character represents a unique matched substring of a MUM pair and the orderings of these n characters follow the way the corresponding substrings appear in the genomes. For any a_i in A, we denote the index of the character in B that matches a_i as $\delta(i)$, i.e., $(a_i, b_{\delta(i)})$ represents an MUM pair. Both a_i and $b_{\delta(i)}$ are associated with the same weight, which is the length of the corresponding substring.

Note that when we extract the MUM pairs from the two genomes, we consider MUM pairs from two strands of the same orientation as well as those of opposite orientations. For each character a_i in A, a_i has a positive sign if the MUM pairs represented by $(a_i, b_{\delta(i)})$ are from two strands of the same orientation, and a_i has a negative sign otherwise. A character in B always has a positive sign. Intuitively, if a certain part of G_1 is found to be reversed in G_2, we expect that the MUM pairs extracted from this part have opposite orderings in A and B, and all the characters a_i of these MUM pairs carry a negative sign.

Common subsequences: A sequence $C = c_1 c_2 \cdots c_m$ is a subsequence of A if there exists indices $i_1, i_2, \cdots i_m$ such that $i_1 < i_2 < \cdots < i_m$ and $c_j = a_{i_j}$ for $1 \leq j \leq m$. C is said to be a maximum weight common subsequence (MWCS) of A and B if among all subsequences common to A and B, C is the one with the maximum total weight. Note that for C to be a common subsequence of A and B, we require that all the involved characters carry the same sign in both A and B.

Mutations: Given a sequence $X = x_1 x_2 \cdots x_\ell$, we consider the following three types of mutations.

- A *reversal* $r(i, j)$, where $1 \leq i \leq j \leq \ell$, reverses the ordering of $x_i x_{i+1} \cdots x_j$ and toggles their signs.
- A *transposition* $t(i, j, d)$, where $1 \leq i \leq j \leq \ell$ and $0 \leq d \leq \ell$ with $d \notin [i-1, j]$, moves the substring $x_i x_{i+1} \cdots x_j$ to the location between x_d and x_{d+1}. The signs of the characters are unchanged.
- A *reversed-transposition* $rt(i, j, d)$, where $1 \leq i \leq j \leq \ell$ and $0 \leq d \leq \ell$ with $d \notin [i-i, j]$, moves the substring $x_i x_{i+1} \cdots x_j$ to the location between x_d and x_{d+1} and reverses the ordering of $x_i x_{i+1} \cdots x_j$. The signs of the characters are toggled.

We illustrate the three mutation operations in Figure 1.

The MUM Selection Problem: Given two sequences A and B and an integer k, we call a pair of subsequence X of A and subsequence Y of B a pair of *k-mutated subsequences* if X can be transformed to Y by at most k mutations. The MUM Selection Problem is to find a pair of k-mutated subsequences such that the weight is maximized. When $k = 0$, the problem is equivalent to finding the maximum weight common subsequence.

3. A practical algorithm for selecting MUM pairs (MSS)

In this section, we present an efficient algorithm (MSS) for finding a pair of k-mutated subsequences with weight very close to (if not equal to) the largest possible weight. The time complexity is $O(n^2(\log n + k))$. This algorithm has been implemented and used in our new software for locating conserved gene pairs; as to be shown in the next section, it performs very well in all of the test cases.

To find a pair of k-mutated subsequences of A and B that have a high score, we first find the maximum weight common subsequence (MWCS) of A and B, which we call the *super-backbone*. Then we attempt to identify which parts of the super-backbone should be replaced with common subsequences corresponding to mutations so as to increase the overall weights. Roughly speaking, a good candidate for such a common subsequence should be a heavy-weight one and at the same time should replace only a small portion of the super-backbone. Details are as follows.

Definition 1. An interval $A[i, j]$ of A is said to be sign-consistent at its endpoints, or simply *sign-consistent*, if either both $A[i]$ and $A[j]$ have positive signs and $\delta(i) \leq \delta(j)$, or both $A[i]$ and $A[j]$ have negative signs and $\delta(i) \geq \delta(j)$.

Step 1. Super-Backbone: Find the maximum weight common subsequence (MWCS) of A and B. We call this subsequence as the *super-backbone*, based on which we want to add k subsequences corresponding to some mutations that are likely to maximize the overall weight.

Step 2. Score of an interval: For every interval $A[i, j]$ that is sign-consistent, we calculate a score reflecting the gain if $(A[i, j], B[\delta(i), \delta(j)])$ is considered to include a common subsequence corresponding to a mutation that involves the endpoints. More precisely, if $A[i]$ and $A[j]$ both carry a positive sign, the gain is defined as the weight of the MWCS of $A[i, j]$ and $B[\delta(i), \delta(j)]$ minus the total weight of characters in the backbone that fall into $A[i, j]$ or $B[\delta(i), \delta(j)]$. If $A[i]$ and $A[j]$ both carry a negative sign, we consider the reversal of $B[\delta(i), \delta(j)]$ instead.

Step 3. Maximum score of k intervals: Among all intervals $A[i, j]$ that are sign-consistent, find k intervals that are mutually disjoint in A and maximize the total score. This

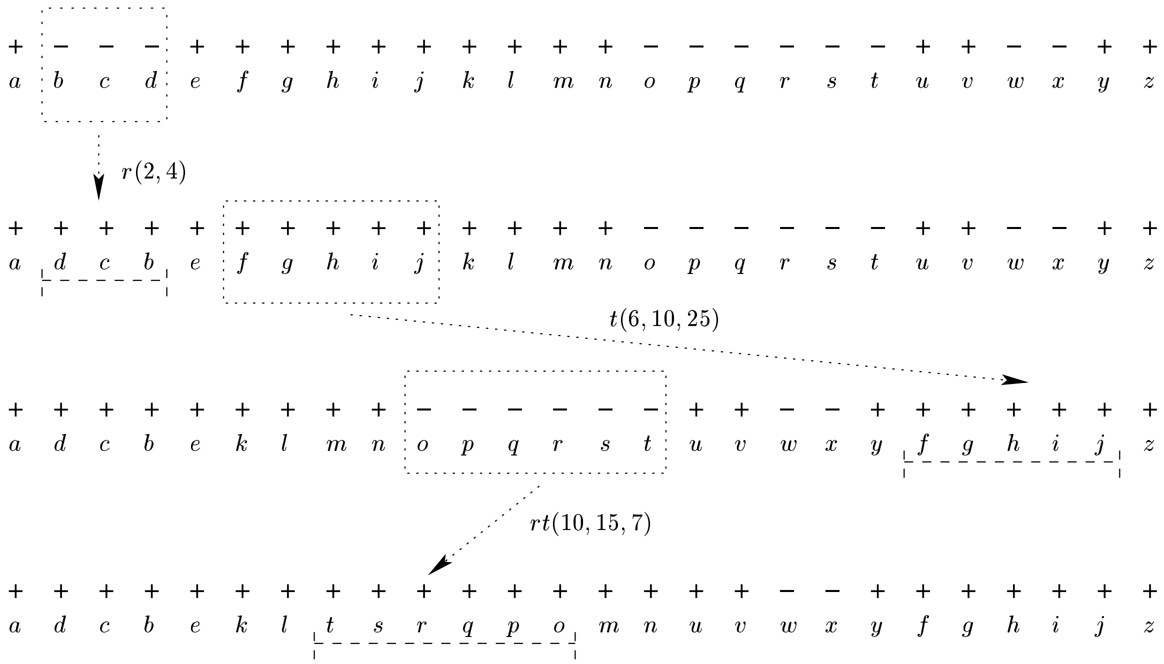


Figure 1: Three types of mutation operations: reversal, transposition, and reversed transposition.

step can be very time consuming if one simply examines every k intervals; fortunately, we can take advantage of the structural relationship and use dynamic programming to report the best k pairs in only $O(kn^2)$ time.

Step 4. Refinement: Consider any two of the k intervals selected in Step 3, say, $A[i, j]$ and $A[i', j']$. Note that $A[i, j]$ and $A[i', j']$ are disjoint, but $B[\delta(i), \delta(j)]$ and $[B_{\delta(i')}, B_{\delta(j')}]$ may not be disjoint. If this is the case, we examine all possible ways to shrink the intervals $A[i, j]$ and $A[i', j']$ so that the resultant intervals on B no longer overlap, and we select the two shrunk intervals that maximize the total score to replace $A[i, j]$ and $A[i', j']$. We repeat such refinement until no more problematic pairs of intervals are left.

Step 5. Output: We report a pair of k-mutated subsequences (X, Y) for A and B as follows: X can be constructed from A by first including all characters in the super-backbone except those enclosed in the k intervals reported in Step 4, and then inserting, for each interval $A[i, j]$ reported in Step 4, the MWCS between $A[i, j]$ and $B[\delta(i), \delta(j)]$ (or its reversal if the sign is negative). Y can be obtained similarly.

3.1. Implementation details of MSS

Note that Steps 1 and 2 require us to find the MWCS of two sequences. Therefore, we first review the existing algorithms for finding common subsequences. If all characters of A and B have the same weight, their MWCS can be computed in $O(n \log n)$ time using the classical LCS algorithm for unique characters [13, 17]. For the case with arbitrary weights, the recent work of Cole et al. [5] showed that $O(n \log n)$ time is still sufficient. Cole et al.'s algorithm can be easily extended to perform the following preprocessing in $O(n^2 \log n)$ time, which will be used in our MSS algorithm.

> For all $1 \le i \le j \le n$, compute the MWCS of $A[i, j]$ and $B[\delta(i), \delta(j)]$, as well as of $A[i, j]$ and the reversal of $B[\delta(i), \delta(j)]$.

To compute the above values in $O(n^2 \log n)$ time, we divide the preprocessing into n phases; in Phase i, we apply Cole et al.'s algorithm to process $A[i, n]$ and $B[\delta(i), n]$; this gives us not only the weight of the MWCS of $A[i, n]$ and $B[\delta(i), n]$, but also a data structure (precisely, a persistent binary tree) allowing us to retrieve the weight of the MWCS of $A[i, h]$ and $B[\delta(i), \ell]$ for any combination of h and ℓ in $O(\log n)$ time. Thus we can retrieve the weight of the MWCS of $A[i, j]$ and $B[\delta(i), \delta(j)]$ for all $j \ge i$ in $O(n \log n)$ time. Details will be given in the full paper.

Using the algorithm of Cole et al. [5], Step 1 takes $O(n \log n)$ time. Assuming we have performed the $O(n^2 \log n)$-time preprocessing, the score of each interval $A[i, j]$ can be computed in $O(1)$ time, and Step 2 takes at most $O(n^2 \log n)$ time. Step 3 is the most non-trivial step; it makes use of dynamic programming so as to improve the time required. Details are as follows.

Define $\mathtt{OPT}[c, j]$ as the maximum total weight for at most c disjoint intervals of A, subject to the requirement that all intervals end at or before $A[j]$. Denote the score of the interval $A[i,j]$ calculated in Step 2 as $\texttt{Score}[i,j]$. The dynamic programming is based on the following recurrence.

Proposition 1. *If* $c = 0$ *or* $j = 0$, $\texttt{OPT}[c,j] = 0$. *Otherwise,* $\texttt{OPT}[c,j] =$

$$\max \begin{cases} \texttt{OPT}[c, j-1] \\ \max_{i=1,\ldots,n}\{\texttt{OPT}[c-1, i-1] + \texttt{Score}[i,j]\} \end{cases}$$

Notice that the $\texttt{OPT}[k,n]$ is the total weight of the k intervals that maximize the total score. We can use a two-level for-loop to compute $\texttt{OPT}[k,n]$ in $O(kn^2)$ time, and recover the positions of the the k intervals in the same time complexity. Steps 4 and 5 are straightforward, using at most $O(kn^2)$ and $O(n^2)$ time, respectively. Thus, the overall time of the algorithm is $O(n^2 \log n + kn^2)$.

The space complexity (memory requirement) of this algorithm is dominated by the preprocessing, which requires $O(n^2)$ space.

Performance guarantee: When tested with real data, the subset of MUM pairs reported by MSS often has a total length very close to the maximum possible length. From a theoretical viewpoint, we are also able to prove that even in the worst case, the MSS algorithm has a bounded performance.

Lemma 2. *Given two sequences A and B and an integer k, the weight of the k-mutated subsequences found by MSS is at least* $1/(3k+1)$ *times that of any k-mutated subsequences.*

Proof (Sketch). Let X^* and Y^* be the pair of maximum weight k-mutated subsequences between A and B. We claim that X^* can be divided into at most $3k+1$ substrings $S_1 S_2 \cdots S_{3k+1}$ such that Y^* is a permutation of these $3k+1$ substrings (we may reverse a substring during the permutation); precisely, Y^* equals to $S'_1 S'_2 \cdots S'_{3k+1}$ such that for every S_i, there is exactly one j such that S_i equals S'_j or the reverse of S'_j. To prove the claim, let $X_0 (= X^*), X_1, \ldots, X_k (= Y^*)$ be sequences such that X_i is the sequence after i mutations have been performed on X^*. Let $X_i = a_1 a_2 \ldots a_n$ where $a_1, a_2, \ldots, a_n$ are characters in X^*. We call (a_i, a_{i+1}) a breakpoint if a_i and a_{i+1} are not adjacent at X^*. Note that a mutation can create at most 3 breakpoints and there are at most $3k$ breakpoints in Y^*. Thus, Y^* is a permutation of at most $3k+1$ substrings of X^*.

Let X and Y be the pair of k-mutated subsequences returned by MSS. The weight of X is at least the weight of the maximum weight common subsequence of A and B. Its weight is also at least the weight of the maximum weight common subsequence of A and the reversal of B. Thus, the weight of X is no less than the maximum weight substring among the $3k+1$ substrings described above. □

4. Locating conserved gene pairs

In this section, we show how to exploit the MSS algorithm to develop a software for locating conserved gene pairs, and show the result of using this software as well as two existing software MUMmer-3 and MaxMinCluster on aligning 15 pairs of human and mouse chromosomes. We find that our software is consistently more effective.

4.1. The software

The input is two DNA sequences.

Step 1: Finding MUM pairs.
We identify all MUM pairs of length at least 20 using a suffix tree. The MUM pairs with length less than 20 are likely to be noise [6]. The number of MUM pairs identified in this step is usually large and can be tens of thousands (see Table 1). Most of these pairs are believed to be noise and originated from intergenic regions.

Step 2: A simple filtering.
As Step 1 often reports a large number of isolated MUM pairs which are all likely to be noisy pairs, we apply the following filtering to remove some trivial isolated pairs: For each MUM pair m_i, we check for the existence of another pair within a pre-defined distance d on the genomes. To avoid removing correct MUM pairs, we set d to a reasonably large value. If no such a pair exists, we remove m_i. In our experiments, we set $d = 8000$ and find that the filtering is very effective.

Step 3: After filtering, we apply the MUM selection algorithm MSS to find an alignment of the remaining MUM pairs. Note that pairs not reported by MSS are considered as noise.

In the rest of this paper, we refer to the new software for locating conserved gene pairs as MSS alignment.

4.2. Experimental Results

We have selected 15 pairs of human and mouse chromosomes as our test cases. For each pair of chromosomes, the biological community has already identified a number of conserved genes; details are published in GenBank[1]. The set of published gene pairs will be the reference for our evaluation. The details of the data sets are given in Table 1. We implemented the MSS alignment software on a PC with 512M

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.

Mouse Chromosome No.	Human Chromosome No.	# of Published Gene Pairs	# of MUM pairs
2	15	51	96,473
7	19	192	52,394
14	3	23	58,708
14	8	38	38,818
15	12	80	88,305
15	22	72	71,613
16	16	31	66,536
16	21	64	51,009
16	22	30	61,200
17	6	150	94,095
17	16	46	29,001
17	19	30	56,536
18	5	64	131,850
19	9	22	62,296
19	11	93	29,814

Table 1: The data.

RAM and a 2.4GHz CPU. The actual running time of our system depends on the number of MUM pairs and the input parameter k. We have varied the values of k in our experiments. We first present the results for $k = 4$; the results for other values of k are presented next. The running times of our system are reasonable and range from several minutes to a couple of hours.

We have compared our new software MSS alignment with MUMmer-3 and MaxMinCluster from two perspectives: the *coverage* and the *sensitivity*. Coverage counts the percentage of published gene pairs for which some MUM pairs are reported. However, high coverage alone may not imply high quality in the output as one can simply output every MUM pair to achieve the maximum coverage. Thus, we also consider the percentage of reported MUM pairs that actually reside in a conserved gene pair. This percentage is referred as the *sensitivity* of the output. (Notice that the sensitivity is measured based on the published conserved gene pairs; it might be the case that some conserved gene pairs are not known in the literature, so the sensitivity is a rough indicator and should be used as a reference.)

Figure 2 shows the coverage and the sensitivity of MUMmer, MaxMinCluster, and MSS alignment in 15 test cases. On average, MSS alignment is able to uncover 91% of all the published gene pairs. In general, MSS alignment, has a better coverage and slightly higher sensitivity than both MUMmer and MaxMinCluster. Precisely, MSS alignment has an average coverage 15% and 7% higher than that of MUMmer and MaxMinCluster, respectively. For sensitivity, in 10 out of 15 test cases, MSS alignment has a higher sensitivity. For the other five cases, the sensitivity of MSS alignment is only lower than the other two software by at most 1%. On average, the sensitivity of MSS alignment is higher than that of MUMmer and MaxMinCluster by about 2.8% and 1.9%, respectively. It is also worth-mentioning that MSS alignment has a higher average number of MUM pairs reported for a known gene pair; the actual statistics are 22.5, 24.9, 28.3 for MUMmer, MaxMinCluster, and MSS alignment, respectively.

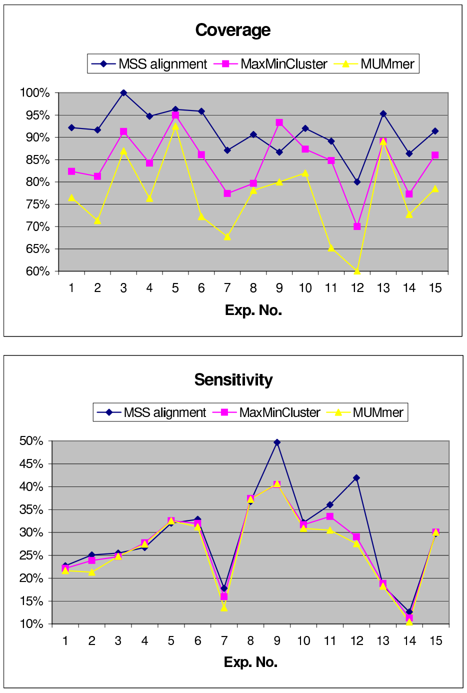


Figure 2: Comparison of MUMmer, MaxMinCluster and MSS alignment. (For MUMmer, we set the parameter gap = 2000; for MaxMinCluster, we set the noisy level to 3 and gap = 2000.)

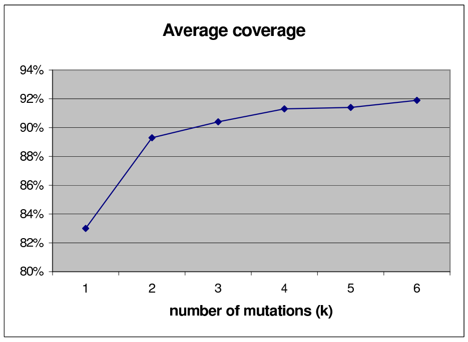
Figure 3 shows the average coverage and average sensitivity of MSS alignment for different values of k. The results show that the average coverage increases as k increases while the average sensitivity decreases slightly as k increases (except for $k = 0$).

We remark that the filtering procedure (Step 2) used by MSS alignment is very effective as it can reduce at least 80% of MUM pairs generated from Step 1, while the number of genes with all their MUM pairs being removed ranges from 0% to 2% among the 15 test cases.

5. NP-hardness

Though MSS performs very well in practice, yet from a computational viewpoint, MSS may not be able to find the pair of k-mutated subsequences with maximum weight. Nevertheless, the complexity of the MUM Selection Problem is open. Yet, if we restrict the mutation operations to be atomic, the MUM Selection Problem can be shown to be NP-hard.

Atomic mutations: Consider a sequence of characters $X = x_1 x_2 \cdots x_\ell$. Any mutation operation $r(i, j)$, $t(i, j, d)$ or $rt(i, j, d)$ moves and/or reverses a block of charac-



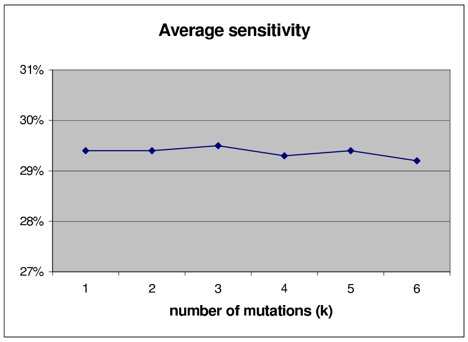


Figure 3: Average coverage and average sensitivity of MMS alignment for different values of k.

ters $x_i x_{i+1} \cdots x_j$ for some $1 \leq i \leq j \leq \ell$. We call this block of characters the *mutated block* of the mutation operation. A sequence of mutations on X is said to be *atomic* if for any mutation M in the sequence, the mutated block of M is disjoint from the mutated block of any previous mutation, and the mutated block of M is not moved to a position inside the mutated block of any previous mutation. We say that a pair of subsequences is a pair of k-mutated subsequences *restricted to atomic mutations* if one subsequence can be transformed to the other by at most k atomic mutations.

Below, we prove that the MUM Selection Problem restricted to atomic mutations is NP-hard. Technically speaking, we will show a polynomial time reduction from the Maximum 2-Satisfiability (Max 2-SAT) problem [9] to the decision version of the MUM Selection Problem.

The MUM Selection Problem restricted to atomic mutations (decision version): Let $A[1..n]$ and $B[1..n]$ be two strings that are permutations of n distinct weighted characters. Given integers $k(\leq n)$ and w, the problem is to decide if there is a pair of k-mutated subsequences restricted to atomic mutations between A and B with weight at least w.

The Max-2SAT Problem: Given a set of variables U, a collection C of clauses with each clause being a disjunction of exactly two literals over U, and a positive integer $r \leq |C|$, the problem is to decide if there is a truth assignment for U that satisfies at least r clauses in C.

Given an instance of the Max 2-SAT problem where $U = \{u_1, u_2, \cdots, u_n\}$ is the set of variables, $C = \{c_1, c_2, \cdots, c_m\}$ is the set of clauses, and an integer r, we construct an instance of the MUM Selection Problem as follows. Recall that the instance consists of two strings A and B, and two positive integers k and w. We first define $2mn + 4m + 2(r+1)m$ distinct weighted characters which are used to form A and B. Each character is associated with a positive sign.

1. For each variable u_i, define $2m$ characters as $u_i[1], \cdots, u_i[m]$ and $\overline{u_i}[1], \cdots, \overline{u_i}[m]$. Each of these characters has weight 1.
2. For each clause c_j, define 4 characters as $c_j[1]$, $c_j[2]$, $c_j[3]$, $c_j[4]$. Each of these characters has weight $2mn$.
3. Define m sets of separator characters each containing $2(r+1)$ characters. Denote the i-th set of separator characters as $s_1[i], \cdots, s_{2(r+1)}[i]$. Each of these characters has weight $2mn + 4m \times 2mn + 1$.

The string A can be regarded as the concatenation of $n + 1 + 2m$ substrings, denoted by $A_{V_1} \cdots A_{V_n} A_{S_1} A_{C_1} A_{S_2} A_{C_2} \cdots A_{S_m} A_{C_m}$. The first n substrings A_{V_i} correspond to n variables, the substrings A_{C_j} correspond to m clauses. The substrings A_{S_j} is to separate the clauses from one another. The details of these substrings are given as follows. $A_{V_i} = u_i[1] \cdots u_i[m] \overline{u_i}[1] \cdots \overline{u_i}[m]$, $A_{C_j} = c_j[1] c_j[2] c_j[3] c_j[4]$, $A_{S_j} = s_1[j] \cdots s_{2(r+1)}[j]$. The signs of all characters in A_{V_i}, A_{C_j} are all positive. For A_{S_j}, all the characters $s_{2p-1}[j]$ carry positive sign while all the characters $s_{2p}[j]$ carry negative sign.

For the string B, we first construct a string B' which is the concatenation of $n + m$ substrings, $B_{V_1} \cdots B_{V_n} B_{S_1} \cdots B_{S_m}$. The details of these substrings are given as follows. $B_{V_i} = \overline{u_i}[1] \cdots \overline{u_i}[m] u_i[1] \cdots u_i[m]$, $B_{S_j} = s_2[j] s_1[j] s_4[j] s_3[j] \cdots s_{2(r+1)}[j] s_{2r+1}[j]$.

To construct B from B', we insert the characters $(c_j[1], c_j[2], c_j[3], c_j[4])$ into B' depending on the literals used in the clause c_j. Let $c_j = \ell_a + \ell_b$ where ℓ_q represents the literal u_q or $\overline{u_q}$. Note that we require $a < b$. We insert $c_j[2]$ and $c_j[4]$ into B' to enclose $\ell_a[j]$, more precisely, $c_j[2] \ell_a[j] c_j[4]$ forms a substring in B. We insert $c_j[1]$ and $c_j[3]$ into B' such that $c_j[1] \ell_b[j] c_j[3]$ forms a substring in B. All characters in B carry positive sign. The structures of A and B are illustrated in Figure 4.

Finally, we let k be $r + m(r+1)$ and w be $mn + r \times 4mn + m(2r+2)(2mn + 4m \times 2mn + 1)$. We can easily verify that A, B, k and w satisfy the input requirement for the MUM Selection Problem and they can be constructed in $O(mn)$ time. The following lemmas are the main results of this section.

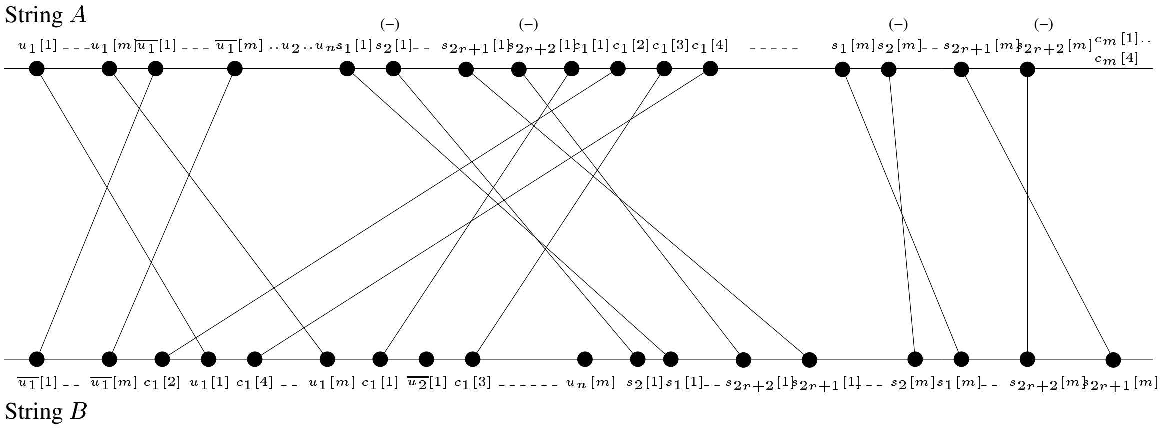


Figure 4: Strings A and B (For the clauses, we show the case for $c_1 = u_1 + \overline{u_2}$).

Lemma 3. *If there is a truth assignment for U that satisfies at least r clauses in C, then there is a pair of* $(r+m(r+1))$*-mutated subsequences restricted to atomic mutations between* A *and* B *of weight at least* $mn + r \times 4mn + m(2r+2)(2mn + 4m \times 2mn + 1)$.

Proof. To be given in the full paper. □

Lemma 4. *If there is a pair of* $(r+m(r+1))$*-mutated subsequences restricted to atomic mutations between* A *and* B *of weight at least* $mn + r \times 4mn + m(2r+2)(2mn+4m \times 2mn+1)$*, then there is a truth assignment for U that satisfies at least r clauses in C.*

Proof. To be given in the full paper. □

Theorem 5. *The MUM Selection Problem restricted to atomic mutations is NP-hard.*

References

[1] V. Bafna and P. Pevzner. Genome rearrangements and sorting by reversals. *SIAM Journal on Computing*, 25(2):272–289, 1996.

[2] V. Bafna and P. Pevzner. Sorting by transpositions. *Journal on Discrete Mathematics*, 11(2):224–240, 1998.

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

[4] A. Caprara. Sorting permutations by reversals and eulerian cycle decompositions. *SIAM Journal on Discrete Mathematics*, 12(1):91–110, 1999.

[5] Richard Cole, Martin Farach-Colton, Ramesh Hariharan, Teresa M. Przytycka, and Mikkel Thorup. An $O(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees. *SIAM Journal on Computing*, 30(5):1385–1404, 2000.

[6] 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.

[7] 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.

[8] N. Eriksen. $(1+\epsilon)$-approximation of sorting by reversals and transpositions. *Theoretical Computer Science*, 289:517–529, 2002.

[9] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman & Company, 1979.

[10] Q. P. Gu, S. Peng, and H. Sudborough. A 2-approximation algorithm for genome rearrangements by reversals and transpositions. *Theoretical Computer Science*, 210:327–339, 1999.

[11] S. Hannenhalli and P. Pevzner. Transforming cabbage into turnip: Polynomial algorithm for sorting signed permutations by reversals. *Journal of ACM*, 46(1):1–27, 1999.

[12] T. W. Lam, N. Lu, H. F. Ting, Prudence W. H. Wong, and S. M. Yiu. Efficient algorithms for optimizing whole genome alignment with noise. In *Proceedings of the Fourteenth International Symposium on Algorithms and Computation*, pages 364–374, 2003.

[13] William J. Masek and Michael S. Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20(1):18–31, 1980.

[14] P. Pevzner. *Computational Molecular Biology – An Algorithmic Approach*. The MIT Press, Cambridge, 2000.

[15] 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.

[16] The MUMMER Web Site, 2003. http://www.tigr.org/software/mummer/.

[17] T.G. Szymanski. A special case of the maximal common subsequence problem. Technical Report TR-170, Computer Science Laboratory, Princeton University, 1975.

[18] 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.