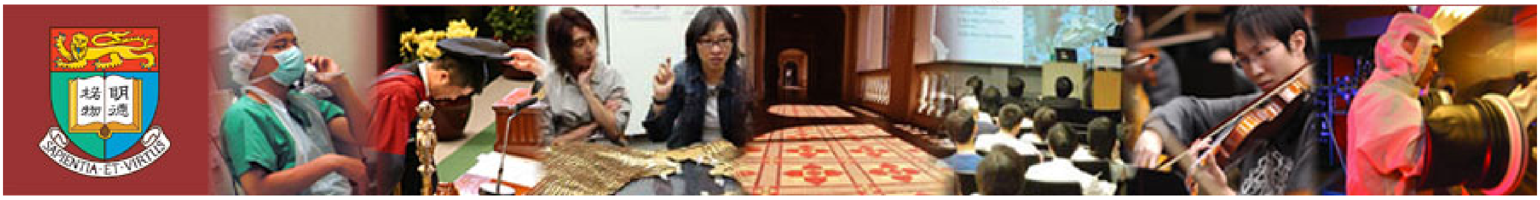

BIOINFORMATICS ORIGINAL PAPER

Vol. 21 no. 10 2005, pages 2271–2278
doi:10.1093/bioinformatics/bti371

Sequence analysis

The mutated subsequence problem and locating conserved genes

H. L. Chan[1,*], T. W. Lam[1], W. K. Sung[2], Prudence W. H. Wong[3], S. M. Yiu[1] and X. Fan[1]

[1]Department of Computer Science, University of Hong Kong, Hong Kong, China, [2]Department of Computer Science, National University of Singapore, Singapore and [3]Department of Computer Science, University of Liverpool, UK

Received on December 16, 2004; revised on February 21, 2005; accepted on March 1, 2005
Advance Access publication March 3, 2005

ABSTRACT

Motivation: For the purpose of locating conserved genes in a whole genome scale, this paper proposes a new structural optimization problem called the Mutated Subsequence Problem, which gives consideration to possible mutations between two species (in the form of reversals and transpositions) when comparing the genomes.

Results: A practical algorithm called mutated subsequence algorithm (MSS) is devised to solve this optimization problem, and it has been evaluated using different pairs of human and mouse chromosomes, and different pairs of virus genomes of Baculoviridae. MSS is found to be effective and efficient; in particular, MSS can reveal >90% of the conserved genes of human and mouse that have been reported in the literature. When compared with existing softwares MUMmer and MaxMinCluster, MSS uncovers 14 and 7% more genes on average, respectively. Furthermore, this paper shows a hybrid approach to integrate MUMmer or MaxMinCluster with MSS, which has better performance and reliability.

Availability: http://www.cs.hku.hk/~mss/

Contact: hlchan@cs.hku.hk

1 INTRODUCTION

As more and more genomes have been sequenced, there is a great desire to study and compare related species in a whole genome scale. Given the genomes of two related species, one important task is to uncover and locate the conserved genes, i.e. genes sharing similar functions (Baillie and Rose, 2000; Schwartz *et al*., 2000; Vincens *et al*., 1998). This task is non-trivial as most parts of a genome are non-coding areas and the locations of genes in each genome is often not available. Alignment software developed in an early stage, e.g. BLAST (Altschul *et al*., 1990) and FASTA (Pearson and Lipman, 1988), are not able to accomplish this task. In this paper, we propose an effective algorithm for identifying locations on the genomes that correspond to conserved genes.

MUMmer-1 (Delcher *et al*., 1999) is one of the earliest software that could perform genome comparisons in a whole genome scale. Since then, several other programs have been developed for large-scale genome comparison, e.g. ASSIRC (Vincens *et al*., 1998), PipMaker (Schwartz *et al*., 2000) and WABA (Baillie and Rose, 2000). In the process of uncovering conserved genes, most of these software are based on a very useful observation made by Delcher *et al*. (1999): A conserved gene rarely comprises the same entire sequence in the two genomes, yet 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 locate conserved genes is to identify pairs of matched substrings that appear uniquely in both genomes. This can be done in linear time using a suffix tree (Delcher *et al*., 1999). Such pairs of matched segments are called the MUM pairs. However, not every MUM pair corresponds to a conserved gene; 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 (Delcher *et al*., 1999) 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. It is commented in a recent survey (Chain *et al*., 2003) that all the software mentioned above are based on this assumption. MUMmer-2 (Delcher *et al*., 2002) and MUMmer-3 (Kurtz *et al*., 2004) adopt a different approach; they select MUM pairs that are close together, i.e. forming a cluster. Intuitively, a conserved gene 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 (Wong *et al*., 2004) 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 evolution. In other words, for the MUM pairs introduced by the conserved genes, the difference in the orderings in the two genomes should be related to the mutations that have occurred. For related species, the number of such mutations would be small. Thus, one should select those MUM pairs whose difference in orderings can be explained by a few mutations. Using this idea, we propose a new approach to select MUM pairs based on the structural optimization. Our approach has been shown to be significantly more effective than the previous ones when tested with real data.

*To whom correspondence should be addressed.

In modeling mutations, the random breakage model (Nadeau and Taylor, 1984; Ohno, 1973) has been widely accepted, which assumes a random (i.e. uniform and independent) distribution of mutations. Yet, a recent conflicting result (Pevzner and Tesler, 2003b) suggests that mutations might be dependent on genomic features and are not uniformly distributed. This new observation reiterates that uncovering conserved genes with the presence of mutations is a non-trivial task.

In fact, the study of mutations between two related species is not a new topic. A closely related problem is the genome rearrangement problem (Bafna and Pevzner, 1996; Hannenhalli and Pevzner, 1999; Kaplan *et al*., 1999; Bafna and Pevzner, 1998; Gu *et al*., 1999; Eriksen, 2002). We consider the signed version of this problem: the conserved genes and their locations in the two genomes are given in advance. Each conserved gene is represented by a unique signed integer, and the orderings of the genes in the two genomes are 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 to π_2. The genome rearrangement problem has been studied intensively. There are several results on mutations restricted to reversals only (Bafna and Pevzner, 1996; Hannenhalli and Pevzner, 1999; Kaplan *et al*., 1999), while in some other studies, transpositions and reversed-transpositions are also included (Bafna and Pevzner, 1998; Gu *et al*., 1999; Eriksen, 2002). The genome rearrangement problem is solvable in polynomial time when the reversals only mutations are (Hannenhalli and Pevzner, 1999). 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 approximation algorithms have been proposed (Bafna and Pevzner, 1998; Eriksen, 2002; Gu *et al*., 1999). As for the unsigned version of the genome rearrangement problem, refer to Pevzner (2000) 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 to conserved genes. Our problem can be regarded as a generalization of the genome rearrangement problem. Note that a subset of MUM pairs induces two signed permutations σ_1 and σ_2, according to the orderings of the 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 the positional difference between the MUM pairs. Obviously, if we make no restriction on the number of 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 correspond to conserved genes. We call this problem the Mutated Subsequence Problem. (The definition is given in Section 2.) The genome rearrangement problem involving reversals, transpositions and reversed-transpositions can be reduced to the Mutated Subsequence Problem. As the former is believed to be NP-hard, the Mutated Subsequence Problem is likely to be even more NP-hard.

1.1 Main results

This paper gives an efficient algorithm called mutated subsequence algorithm (MSS) 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 algorithm 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 of at least $1/(3k+1)$ times the maximum weighted subset.

Table 1. Average coverage (and sensitivity) of different algorithms in locating conserved genes

	Mouse/ human (%)	Intergenus Baculoviridae (%)	Intergenus Baculoviridae (%)
MUMmer-3	77 (27)	66 (71)	43 (62)
MaxMinCluster	84 (27)	69 (75)	45 (59)
MSS	91 (29)	78 (87)	36 (53)
MUMmer-3 + MSS	91 (28)	79 (75)	48 (43)
MaxMinCluster + MSS	91 (27)	79 (82)	51 (53)

Based on MSS, we have implemented two software for locating conserved genes. The first one simply applies MSS directly to a given set of MUM pairs. It performs very well for species that are closely related and involve only a few mutations. We have tested the software using the DNA sequences of 15 pairs of mouse and human chromosomes, as well as the translated protein sequences of Baculoviridae genomes that are in the same genus (specifically, either pairs of *Nucleopolyhedrovirus* genomes or pairs of *Granulovirus* genomes). The performance is compared with that of MUMmer-3 and MaxMinCluster; the average figures are shown in the first two columns of Table 1. It is encouraging to see that MSS consistently achieves better coverage while preserving the sensitivity (coverage refers to the percentage of published genes that are reported by the software and sensitivity refers to the percentage of the reported MUM pairs that are known to reside in a conserved gene; note that sensitivity is an estimate as not all conserved genes have been identified).

We have also tested MSS with pairs of Baculoviridae genomes that are not in the same genus. As one may expect, MSS does not perform well in these cases as the number of mutations between a pair of such viruses is big. Also, for these genomes, there may be hot spots where a number of mutations cluster together in the same region. Applying MSS directly on the MUM pairs may not be appropriate for handling genomes with large number of mutations and hot spots (details are given in Section 4).

The second software we have developed adopts a hybrid approach. Our aim is to obtain a software that can handle genomes that are closely related as well as those with more mutations and hot spots. The hybrid approach first applies MaxMinCluster (or MUMmer-3) to identify some clusters that are obviously conserved genes; these clusters are each treated as a MUM pair and processed together with the remaining MUM pairs using MSS. For genomes that are closely related, the hybrid approach has almost the same performance as MSS alone; yet for genomes that are farther away, the hybrid approach differentiates itself from MSS alone and attains a coverage even better than MaxMinCluster and MUMmer-3 (see the last column of Table 1).

1.2 Organization of the paper

Section 2 gives the definition of the Mutated Subsequence Problem. Section 3 presents an algorithm for finding the maximum weight common subsequence (MWCS), which serves as a subroutine for the algorithm MSS. Details of MSS is given in Section 4. In Section 5, we present the new software for locating conserved genes, and the results of experiments on the real data.

2 THE MUTATED SUBSEQUENCE PROBLEM

2.1 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 as $A = a_1a_2\cdots a_n$ and $B = b_1b_2\cdots b_n$, respectively, where each character represents the 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 a MUM pair. Both a_i and $b_{\delta(i)}$ are associated with the same weight $w(a_i)$, which is the length of the corresponding substring.

Each character in A and B is given a sign as follows. A DNA sequence is double stranded. When we extract MUM pairs from two DNA sequences, we consider MUM pairs from two strands of the same orientation as well as of opposite orientations. That is, given G_1 and G_2 representing two strands of the same orientation we need to perform the procedure for finding MUM pairs twice, first with G_1 in the given orientation and then with G_1 reversed. 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 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.

2.2 Common subsequences

A sequence $C = c_1c_2\cdots c_m$ is a subsequence of A if there exists indices $i_1, i_2, \ldots, i_m$ such that $i_1 < i_2 < \cdots < i_m$ and $c_j = a_{i_j}$ for $1 \le j \le m$. C is said to be an 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 all the involved characters to carry the same sign in both A and B.

2.3 Mutations

Given a sequence $X = x_1x_2\cdots x_\ell$, we consider the following three types of mutations.

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

2.4 The mutated subsequence problem

Given two sequences A and B and an integer k, we call a subsequence X of A and a subsequence Y of B, a pair of k-mutated subsequences if X can be transformed to Y by at most k mutations. The Mutated Subsequence 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 MWCS.

2.5 Reducing genome rearrangement to Mutated Subsequence Problem

Given two permutations of signed integers π_1 and π_2, the genome rearrangement problem involving reversals, transpositions and reversed-transpositions asks for the minimum number of mutations needed to transform π_1 to π_2. This problem can be polynomial-time reduced to the Mutated Subsequence Problem as follows. We associate a weight of 1 to each integer of π_1 and π_2. For $k = 1, 2, \ldots,$ we query the Mutated Subsequence Problem with input π_1 and π_2 for the pair of maximum weight k-mutated subsequences. Let k' be the smallest integer such that the pair of k'-mutated subsequences is exactly π_1 and π_2. π_1 can be transformed to π_2 using k' mutations but not $k'-1$ mutations; so k' is the minimum number of mutations needed to transform π_1 to π_2. k' is the maximum length of π_1; so at most a polynomial number of queries are made.

As the genome rearrangement problem involving reversals, transpositions and reversed-transpositions is believed to be NP-hard, the Mutated Subsequence Problem is likely to be even more NP-hard.

3 MAXIMUM WEIGHT COMMON SUBSEQUENCE

This section presents an $O(n \log n)$ time algorithm which, given two sequences of n distinct characters, finds the MWCS, or equivalently, solves the Mutated Subsequence Problem for the special case of $k = 0$ (i.e. no mutation is allowed). This algorithm also serves as a subroutine for the algorithm MSS given in the next section. The algorithm makes use of the techniques in the work of Cole *et al.* (2000) to compute the maximum agreement subtree.

LEMMA 1. *Given two sequences $A[1..n] = a_1a_2\cdots a_n$ and $B[1..n] = b_1b_2\cdots b_n$ of n distinct characters, we can find the MWCS in $O(n \log n)$ time. Furthermore, by the end of the algorithm, a data structure is built such that for any pair of prefixes $A[1..i]$ and $B[1..j]$, $1 \le i, j \le n$, the weight of their MWCS can be retrieved in $O(\log n)$ time.*

We denote MWCS (A, B) as the weight of the MWCS of A and B. Let $C_\ell[k]$ be MWCS $(A[1..\ell], B[1..k])$. Note that $C_\ell[k] = 0$ if $\ell = 0$ or $k = 0$. For other values of ℓ and k, we have the following equation. Recall that $w(A[i])$ is the weight of the character $A[i]$ and $\delta(i)$ is the index of the character in B that matches $A[i]$.

$$C_\ell[k] = \max \begin{cases} C_{\ell-1}[k] \\ w(A[\ell]) + C_{\ell-1}[\delta(\ell) - 1] \quad \text{if } k \ge \delta(\ell) \end{cases}. \qquad (1)$$

By computing the function C_ℓ for $\ell = 1, 2, \ldots, n$ incrementally, we can eventually compute MWCS (A, B), which equals $C_n[n]$. This simple approach takes $O(n^2)$ time.

We observe that the values in $C_\ell[1..n]$ are increasing, i.e. $C_\ell[1] \le C_\ell[2] \le \cdots \le C_\ell[n]$. Instead of storing the values in C_ℓ explicitly, we store only the boundaries at which the values change. Precisely, $C_\ell[1..n]$ can be represented by the pairs $(i, C_\ell[i])$ where $C_\ell[i] > C_\ell[i-1]$. Furthermore, we store these tuples in a binary search tree, denoted as T_ℓ, which allows us to efficiently retrieve the value of $C_\ell[i]$ for any i.

Given $T_1, \ldots, T_{\ell-1}$, we can make use of Equation (1) to compute $C_\ell[\delta(\ell)]$ in $O(\log n)$ time. Then we can build T_ℓ from $T_{\ell-1}$ as follows. Notice that $C_\ell[\delta(\ell)] \geq C_{\ell-1}[\delta(\ell)]$, so either $C_\ell[\delta(l)] = C_{\ell-1}[\delta(\ell)]$ or $C_\ell[\delta(\ell)] > C_{\ell-1}[\delta(\ell)]$. Lemma 2 shows that in either case, all the values in the array C_ℓ can be computed easily.

LEMMA 2. *(a) If $C_\ell[\delta(\ell)] = C_{\ell-1}[\delta(\ell)]$, then $C_\ell[k] = C_{\ell-1}[k]$ for all $k = 1, 2, \ldots, n$.*
(b) If $C_\ell[\delta(\ell)] > C_{\ell-1}[\delta(\ell)]$, let k_0 be the smallest integer greater than $\delta(\ell)$ such that $C_\ell[\delta(\ell)] < C_{\ell-1}[k_0]$. Then, $(i) C_\ell[k] = C_{\ell-1}[k]$ for all $k < \delta(\ell)$ and $k \geq k_0$; and $(ii), C_\ell[k] = C_\ell[\delta(\ell)]$ for $\delta(\ell) \leq k < k_0$.

Hence, we can build T_ℓ from $T_{\ell-1}$ as follows. If $C_\ell[\delta(\ell)] = C_{\ell-1}[\delta(\ell)]$, then by Lemma 2(a), T_ℓ is same as $T_{\ell-1}$. Otherwise, by Lemma 2(b), we can construct T_ℓ from $T_{\ell-1}$ by deleting all tuples $(i, C_{\ell-1}[i])$ where $i \geq \delta(\ell)$ and $C_{\ell-1}[i] \leq C_\ell[\delta(\ell)]$, followed by inserting the tuple $(\delta(\ell), C_\ell[\delta(\ell)])$. Denote α_ℓ as the number of pairs being deleted. The time for computing T_ℓ is $O((\alpha_\ell + 1)\log n)$.

Apparently, the above method implies that $T_{\ell-1}$ is erased once T_ℓ is obtained. Nonetheless, by exploiting a persistent data structure (Sarnak and Tarjan, 1986), both T_ℓ and $T_{\ell-1}$ can coexist after the insert and delete operations, while retaining the same time complexity for construction and accession. In summary, the total time for constructing $T_1, \ldots, T_n$ is $O\left(\sum_{\ell=1}^{n}(\log n + \alpha_\ell \log n)\right)$. As we insert at most n pairs into these trees, we can delete at most n pairs, and $\sum_{\ell=1}^{n} \alpha_\ell \leq n$. Hence, $T_1, \ldots, T_n$ can all be computed in $O(n \log n)$ time. The weight of the MWCS of A and B is given by $C_n[n]$. The required subsequence can be found in $O(n \log n)$ time using the standard backtracking method. Also, for any pair of prefixes $A[1..i]$ and $B[1..j]$, where $1 \leq i, j \leq n$, the weight of their MWCS is given by $C_i[j]$, which can be accessed in $O(\log n)$ time.

4 A PRACTICAL ALGORITHM FOR SELECTING MUM PAIRS

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 genes. We will show, in the next section, that MSS performs well in all test cases of closely related genomes.

To find a pair of k-mutated subsequences of two sequences A and B that have a large weight, we first find the MWCS of A and B, which we call the backbone. Then we attempt to identify which parts of the backbone should be replaced with other shorter common subsequences corresponding to different mutations so as to increase the overall weight. Roughly speaking, a good candidate should be heavy-weight common subsequence outside the backbone and should replace only a small portion of the backbone. Details are as follows.

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

DEFINITION 1. *An interval $A[i, j]$, where $i \leq j$, 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 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 step can be very time consuming if one simply examines every k interval; 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 $B[\delta(i'), \delta(j')]$ 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 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 in a similar manner.

Remark. Note that the above algorithm considers only the genes that have been moved at most once and thus it searches for mutations that involve non-overlapping regions. This assumption is reasonable for closely related species. In fact, the experimental results also show that MSS outperforms the others for closely related species, but it does not work very well for Baculoviridae genomes that are not in the same genus. It is expected that there could be more mutations involved in these genomes and some of these mutations may cluster on the same region, called hot spots, which degrade the performance of MSS. In Section 5.4, we will show that it is easy to integrate MUMmer-3 or MaxMinCluster with MSS to obtain a better software which can handle genomes with more mutations and hot spots.

4.1 Implementation details of MSS

Step 1 takes $O(n \log n)$ time by applying the algorithm presented in Section 3. A brute force way to implement Step 2 would require executing the MWCS algorithm n^2 times, using $O(n^3 \log n)$ time.

The following shows how to perform Step 2 in $O(n^2 \log n)$ time. First, we perform the following preprocessing.

> 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 the MWCS 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. After 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. 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 $\mathtt{Score}[i, j]$. The dynamic programming is based on the following recurrence.

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

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

Notice that $\mathtt{OPT}[k, n]$ is the total weight of the k intervals that maximize the total score. We can use a two-level for loop to compute $\mathtt{OPT}[k, n]$ in $O(kn^2)$ time, and recover the positions of 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 complexity 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.

4.2 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, MSS has a bounded performance.

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

5 EXPERIMENTAL RESULTS

In this section, we show how to exploit the algorithm MSS to develop two software for locating conserved genes of two given genomes. We test the software on 15 pairs of human and mouse chromosomes and also on 36 pairs of virus genomes (from the family Baculoviridae). The results are compared with two existing software MUMmer-3 (Kurtz *et al*., 2004) and MaxMinCluster (Wong *et al*., 2004). Table 1 gives a summary of the comparison, showing that our new software is more effective.

5.1 A simple software

The first software we have implemented simply applies MSS directly to find out which MUM pairs are likely to be a part of some conserved genes. Details are as follows:

The input is two DNA sequences. Depending on the user's choice, the software can generate MUM pairs from the DNA sequences or from the translated protein sequences. By using a suffix tree, we can identify in linear time all MUM pairs of length of at least ℓ, where the default value of ℓ is 20 for DNA sequences and 7 for translated protein sequences. After generating the MUM pairs, we apply MSS directly to select the MUM pairs that are likely to correspond to conserved genes. MSS requires a user parameter k (i.e. the number of mutations allowed). A user can choose a particular value of k or let the software determine an appropriate value for the given dataset. In the latter case, the software will try to estimate the evolutionary distance between the species based on the located MUM pairs and detect the effectiveness of allowing more mutations, then set the value of k accordingly. The software has been implemented on a PC with 1 G RAM and a 2.4 GHz CPU.

5.1.1 Measurement We compared the software based on MSS with MUMmer-3 and MaxMinCluster from two perspectives: the coverage and the sensitivity. For coverage, we count the percentage of published conserved genes for which some MUM pairs are reported. We note that 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. This percentage is referred to as the sensitivity of the output. It gives us an indication of 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 select a set of MUM pairs with high coverage and reasonable sensitivity.

5.2 Aligning DNA sequences

We used 15 pairs of human and mouse chromosomes as our test cases. The size of the chromosomes ranged from 14 to 65 million nucleotides. For each pair of chromosomes, the biological community has already identified a number of conserved genes; details are published in GenBank (http://www.ncbi.nlm.nih.gov/Homology). The set of published genes will be the reference for our evaluation.

We generated the MUM pairs of the DNA sequences and it was required that each MUM pair had length ℓ of at least 20. MUM pairs with length <20 are likely to be noise (Delcher *et al*., 1999). These MUM pairs served as input data to our algorithm as well as MUMmer-3 and MaxMinCluster. (Details of the datasets can be found in our website.)

5.2.1 The findings We let the software determine the value of k (the number of mutations allowed) automatically and k is found to be four. In fact, we have also tried other values of k and found that the coverage and sensitivity are more or less the same when we increased the value of k (see below for more discussion). Figure 1 shows the coverage and sensitivity of MUMmer-3, MaxMinCluster and MSS ($k = 4$) in the 15 test cases (refer to our website for detailed experimental results). In general, MSS has a better coverage and slightly higher sensitivity than both MUMmer-3 and MaxMinCluster. Precisely, MSS has an average coverage of 91%, which is 14 and 7% higher than that of MUMmer-3 and MaxMinCluster, respectively.

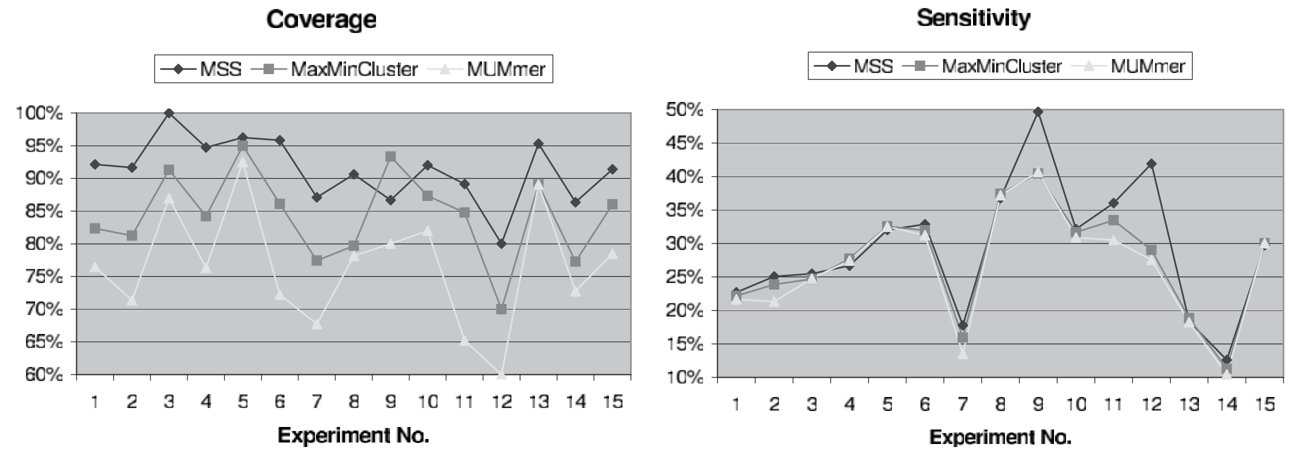

Fig. 1. Performance of MUMmer-3, MaxMinCluster and MSS for aligning mouse and human chromosomes.

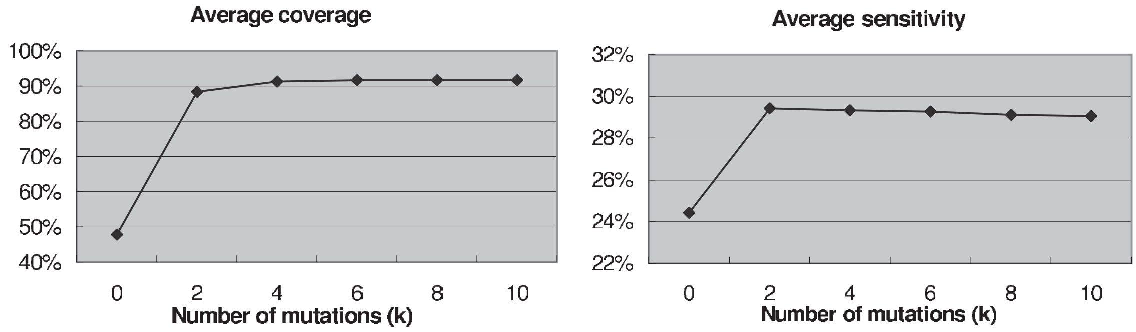

Fig. 2. Average coverage and sensitivity of MSS for different values of k.

The average sensitivity of MSS is 29%, which is higher than that of MUMmer-3 and MaxMinCluster by about 3% and 2%, respectively. The number of conserved genes located by MSS ranges from 19 to 176 and the number of published genes ranges from 22 to 192 in this set of test cases (refer to our website for more details). It is also worth mentioning that MSS has a higher average number of MUM pairs reported for each known conserved gene; the actual statistics are 23, 25 and 28 for MUMmer-3, MaxMinCluster and MSS, respectively.

In summary, MSS, using a mutation sensitive approach to select the MUM pairs, is able to locate the conserved genes more effectively. As a remark, the average running times for MUMmer-3, MaxMinCluster and MSS are 1.3, 2.5 and 3.4 min, respectively, which are quite reasonable.

5.2.2 Different values of k Figure 2 shows the average coverage and sensitivity of MSS for different values of k (i.e. number of mutations allowed). We observe that both the coverage and the sensitivity converge after $k = 4$. Biologically, it was suggested that only 178 ± 39 mutations have occurred between mouse and human (Nadeau and Taylor, 1984), a more recent work (Pevzner and Tesler, 2003a) provided evidence for a larger number of mutations (281) than previously known. It is also known that there are ∼100 pairs of human–mouse chromosomes that are related (Mouse Genome Informatics, 2004, http://www.informatics.jax.org/). The value of 4 seems to be in line with the number of mutations between a pair of mouse and human chromosomes that were predicted in some previous study.

5.3 Aligning translated protein sequences

We used pairs of virus genomes from the family Baculoviridae as our test cases. The virus genomes are of length 100 000–200 000 nt and their corresponding conserved genes have been published in the literature (Herniou *et al.*, 2001, http://www.bio.ic.ac.uk/research/dor/research/eah). Mutations occur more frequently in virus and their DNA sequences show much lower degree of similarity than those of mouse and human. Comparison of the translated protein sequences is more useful in analyzing these distant species.

We generated MUM pairs of length for at least three amino acids. These MUM pairs served as input to MSS and also as input to MUMmer-3 and MaxMinCluster. (Details of the datasets can be found in our website.)

5.3.1 The findings We first used 18 pairs of Baculoviridae genomes that were within the same genus (either *Nucleopolyhedrovirus* or *Granulovirus*). We let the software determine the value of k (the number of mutations allowed) automatically. The value of k was found to be 20. This seems to be reasonable as we expect more mutations to exist in viruses. Figure 3 shows the coverage and sensitivity of MSS in these 18 test cases. MSS achieves the highest coverage in all except one test case, and it has the highest sensitivity in all the 18 cases. Specifically, the average coverage of MSS is 78%, while

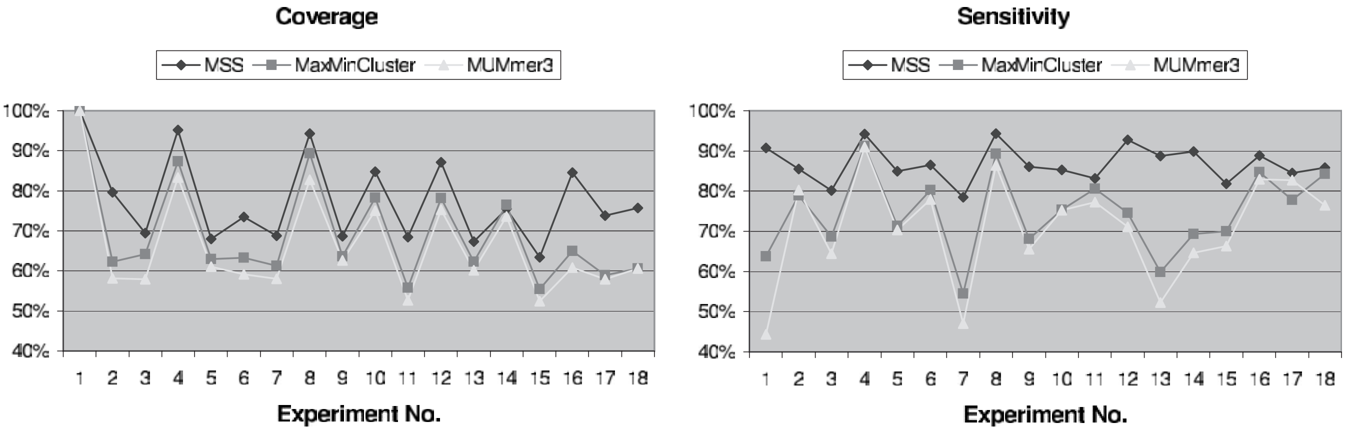

Fig. 3. Performance of MUMmer-3, MaxMinCluster and MSS for aligning the translated protein sequences of 18 pairs of Baculoviridae genomes that are in the same genus.

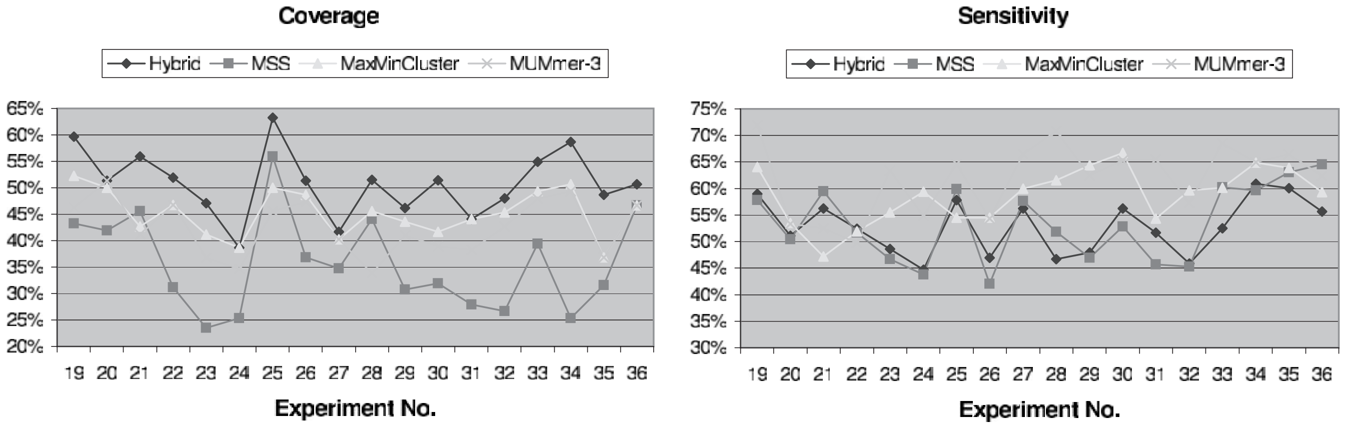

Fig. 4. Performance of the hybrid approach (MaxMinCluster + MSS) for aligning Baculoviridae genomes that are not in the same genus.

the coverage of MaxMinCluster and MUMmer-3 are 69 and 66%, respectively. The number of conserved genes located by MSS ranges from 63 to 134 and the number of published genes ranges from 92 to 134 for these test cases (refer to our website for more details). The sensitivity of the three software are 87, 75 and 71% for MSS, MaxMinCluster and MUMmer-3, respectively.

Next, we considered 18 pairs of Baculoviridae genomes that were not within the same genus. As one may expect, MSS cannot handle genomes that involve too many mutations and possibly with hot spots. The performance of MSS is significantly inferior to MaxMinCluster and MUMmer-3. The average coverage of MSS is 36%, while MaxMinCluster and MUMmer-3 achieve 45 and 43%, respectively. As a remark, the number of conserved genes located by MSS ranges from 16 to 38 and the number of published genes ranges from 68 to 77 in these test cases.

5.4 A better software

The second software we have implemented adopts a hybrid approach. The aim is to obtain a software that can handle genomes that are closely related as well as those with more mutations and hot spots. The hybrid approach first applies MaxMinCluster to identify some clusters that are obviously conserved genes. These clusters are each treated as a MUM pair with a bigger weight and processed together with the remaining MUM pairs using the MSS.

For species that are close, the hybrid approach has the same performance as MSS alone; more specifically, the average coverage is 91% for the case of human–mouse, and 79% for the viruses. For species that might involve a large number of mutations, the hybrid approach differentiates itself from MSS alone and attains a performance even better than MaxMinCluster and MUMmer-3. Figure 4 compares the coverage and sensitivity of this hybrid approach against other software on those pairs of Baculoviridae genomes that are not in the same genus. The hybrid approach can achieve an average coverage of 51% (MUMmer-3, MaxMinCluster and MSS individually can attain only 43, 45 and 36%, respectively), while maintaining the sensitivity at a satisfactory level ($\sim$53%). The number of conserved genes located by the hybrid approach ranges from 29 to 44. Note that the range using MSS alone is only 16–38 (the number of published genes ranges from 68 to 77 for these test cases, refer to our website for more details).

We have also tested the hybrid approach based on MUMmer-3 plus MSS, the performance is slightly worse than MaxMinCluster plus MSS, achieving an average coverage of 48%. We believe that the hybrid approach in general performs better because it can exploit the local clustering algorithm (MUMmer-3 or MaxMinCluster) to handle those mutations that are close together over hot spots. In other words, the mutations (that are close together) over hot spots are handled by MUMmer-3 and MaxMinCluster while the mutations involving disjoint regions (that can be far away) are located by MSS.

In conclusion, we find that the hybrid algorithm (in particular, MaxMinCluster plus MSS) is the most effective algorithm to locate conserved genes in all cases.

ACKNOWLEDGEMENT

This work was supported in part by Hong Kong RGC Grant HKU-7139/04E.

REFERENCES

Altschul,S.F. *et al*. (1990) Basic local alignment search tool. *J. Mol. Biol.*, **215**, 403–410.

Bafna,V. and Pevzner,P. (1996) Genome rearrangements and sorting by reversals. *SIAM J. Sci. Comput.*, **25**, 272–289.

Bafna,V. and Pevzner,P. (1998) Sorting by transpositions. *J. Discrete Mathematics*, **11**, 224–240.

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

Chain,P. *et al*. (2003) An application-focused review of comparative genomic tools: capabilities, limitations and future challenges. *Brief. Bioinformatics*, **4**, 105–123.

Cole,R. *et al*. (2000) An $O(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees. *SIAM J. Sci. Comput.*, **30**, 1385–1404.

Delcher,A.L. *et al*. (1999) Alignment of whole genomes. *Nucleic Acids Res.*, **27**, 2369–2376.

Delcher,A.L. *et al*. (2002) Fast algorithms for large-scale genome alignment and comparison. *Nucleic Acids Res.*, **30**, 2478–2483.

Eriksen,N. (2002) $(1 + \epsilon)$-approximation of sorting by reversals and transpositions. *Theor. Comput. Sci.*, **289**, 517–529.

Gu,Q.P. *et al*. (1999) A 2-approximation algorithm for genome rearrangements by reversals and transpositions. *Theor. Comput. Sci.*, **210**, 327–339.

Hannenhalli,S. and Pevzner,P. (1999) Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. *J. ACM*, **46**, 1–27.

Herniou,E.A. *et al*. (2001) Use of whole genome sequence data to infer baculovirus phylogeny. *J. Virol.*, **75**, 8117–8126.

Kaplan,H. *et al*. (1999) A faster and simpler algorithm for sorting signed permutations by reversals. *SIAM J. Sci. Comput.*, **29**, 880–892.

Kurtz,S. *et al*. (2004) Versatile and open software for comparing large genomes. *Genome Biol.*, **5**, R12.

Mouse Genome Informatics (2004) Mammalian orthology and comparative maps.

Nadeau,J.H. and Taylor,B.A. (1984) Lengths of chromosomal segments conserved since divergence of man and mouse. *Proc. Natl Acad. Sci. USA*, **81**, 814–818.

Ohno,S. (1973) Ancient linkage groups and frozen accidents. *Nature*, **244**, 259–262.

Pearson,W.R. and Lipman,D.J. (1988) Improved tools for biological sequence comparison. *Proc. Natl Acad. Sci. USA*, **85**, 2444–2448.

Pevzner,P. (2000) *Computational Molecular Biology—An Algorithmic Approach*. The MIT Press, Cambridge.

Pevzner,P. and Tesler,G. (2003a) Genome rearrangements in mammalian evolution: Lessons from human and mouse genomic sequences. *Genome Res.*, **13**, 13–26.

Pevzner,P. and Tesler,G. (2003b) Transforming men into mice: the Nadeau–Taylor chromosal breakage model revisited. In *proceedings of RECOMB* 2003, Berlin, Germany, ACM Press, pp. 247–256.

Sarnak,N. and Tarjan,R.E. (1986) Planar point location using persistent search trees. *Commun. ACM*, **29**, 669–679.

Schwartz,S. *et al*. (2000) Pipmaker—a web server for aligning two genomic dna sequences. *Genome Res.*, **10**, 577–586.

Vincens,P. *et al*. (1998) A strategy for finding regions of similarity in complete genome sequences. *Bioinformatics*, **14**, 715–725.

Wong,P.W.H. *et al*. (2004) An efficient algorithm for optimizing whole genome alignment with noise. *Bioinformatics*, **20**, 2676–2684.