The Mutated Subsequence Problem and Locating Conserved Genes

HL Chan [1], TW Lam [1], WK Sung [2], Prudence WH Wong [3], SM Yiu [1], X Fan [1]

[1] Department of Computer Science, University of Hong Kong, Hong Kong; supported in part by Hong Kong RGC Grant HKU-7139/04E.

[2] Department of Computer Science, National University of Singapore, Singapore

[3] Department of Computer Science, University of Liverpool, UK

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 MSA 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 Baculovirdae. MSA is found to be effective and efficient; in particular, MSA can reveal over 90% of the conserved genes of human and mouse that have been reported in the literature. When compared to existing software MUMmer (Delcher et al., 2002; Kurtz et al., 2004) and MaxMinCluster (Wong et al., 2004), MSA uncovers 14% and 7% more genes on average, respectively. Furthermore, this paper shows a hybrid approach to integrate MUMmer or MaxMinCluster with MSA, which has better performance and reliability.

Availability The software is available upon request.

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 nontrivial as most parts of a genome are non-coding areas and the information 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, for example, 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. (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 the evolution. In other words, for the MUM pairs introduced by the conserved genes, their 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.

In modelling 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 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 into π_2. The genome rearrangement problem has been studied intensively. There are several results on mutations restricted to reversals only (e.g., Bafna and Pevzner, 1996; Hannenhalli and Pevzner, 1999; Kaplan et al., 1999), while in some other studies, transpositions and reversed-transpositions are also included (e.g., Bafna and Pevzner, 1998; Gu et al., 1999; Eriksen, 2002). The genome rearrangement problem is solvable in polynomial time when the only mutations are reversals (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, see (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 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 correspond to conserved genes. We call this problem the Mutated Subsequence Problem. (The formal 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 even more likely to be NP-hard.

Main result: This paper gives an efficient algorithm called MSA (mutated subsequence algorithm) 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 MSA has a total length at least $1/(3k+1)$ times the maximum weighted subset.

Based on MSA, we have implemented two software for locating conserved genes. The first one simply applies MSA directly to a given set of MUM pairs. It performs very well for species that are closely related and involve few mutations. We have tested the software using the DNA sequences of fifteen pairs of mouse and human chromosomes, as well as using the translated protein sequences of Baculovirdae 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 MSA 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 MSA with pairs of Baculovirdae genomes that are not in the same genus. As one may expect, MSA does not perform well in these cases as the number of mutations between a pair of such viruses is big.

	Mouse/Human	Intra-genus Baculovirdae	Inter-genus Baculovirdae
MUMmer-3	77% (27%)	66% (71%)	43% (62%)
MaxMinCluster	84% (27%)	69% (75%)	45% (59%)
MSA	91% (29%)	78% (87%)	36% (53%)
MUMmer-3 + MSA	91% (28%)	79% (75%)	48% (43%)
MaxMinCluster + MSA	91% (27%)	79% (82%)	51% (53%)

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

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

Organization of the paper: Section 2 gives the formal definition of the Mutated Subsequence Problem. Section 3 presents an algorithm for finding the maximum weight common subsequence, which serves as a subroutine for the algorithm MSA. Details of MSA 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

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 $A = a_1a_2\cdots a_n$ and $B = b_1b_2\cdots b_n$, respectively, where each character represents the matched substring of an 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 $w(a_i)$, 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 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 $C = c_1c_2\cdots c_m$ 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 $1 \le j \le m$. 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 $X = x_1x_2\cdots x_\ell$, 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* $rt(i,j,d)$, 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.

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 maximum weight common subsequence.

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 transposition 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 at most the length of π_1, so at most a polynomial number of queries are made.

As the genome rearrangement problem involving reversals, transpositions and reversed transposition is believed to be NP-hard, the Mutated Subsequence Problem is even more likely to be 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 maximum weight common subsequence, 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 MSA to be given in the next section. The algorithm makes use of the techniques in the work of Cole et al. (Cole et al., 2000) on computing the maximum agreement subtree.

LEMMA 1. Given two sequences $A[1..n]{=}a_1 a_2 \cdots a_n$ and $B[1..n] = b_1 b_2 \cdots b_n$ of n distinct characters, we can find the maximum weight common subsequence 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 maximum weight common subsequence can be retrieved in $O(\log n)$ time.

We denote *MWCS*(A, B) as the weight of the maximum weight common subsequence of A and B. Let $C_\ell[k]$ be *MWCS*$(A[1..\ell], B[1..k])$. $C_\ell[k] = 0$ 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 $A[i]$.

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

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., $C_\ell[1] \le C_\ell[2] \le \ldots \le C_\ell[n]$. 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 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 $C_\ell[\delta(\ell)] \ge C_{\ell-1}[\delta(\ell)]$, so either $C_\ell[\delta(\ell)] = C_{\ell-1}[\delta(\ell)]$ 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 $k \ge k_0$; and *(ii)*, $C_\ell[k] = C_\ell[\delta(\ell)]$ for $\delta(\ell) \le k < k_0$.

PROOF. **Case (a)**. By equation (1), $C_\ell[k] = C_{\ell-1}[k]$ for all $k < \delta(\ell)$. As $C_\ell[\delta(\ell)] = C_{\ell-1}[\delta(\ell)]$, we observe from Equation (1) that $C_{\ell-1}[\delta(\ell)] \ge w(A[\ell]) + C_{\ell-1}[\delta(\ell) - 1]$. For all $k \ge \delta(\ell)$, $C_{\ell-1}[k] \ge C_{\ell-1}[\delta(\ell)] \ge w(A[\ell]) + C_{\ell-1}[\delta(\ell) - 1]$. Thus, by Equation (1), $C_\ell[k] = C_{\ell-1}[k]$ for all $k \ge \delta(\ell)$.

Case (b). If $C_\ell[\delta(\ell)] > C_{\ell-1}[\delta(\ell)]$, we observe from Equation (1) that $C_\ell[\delta(\ell)] = w(A[\ell]) + C_{\ell-1}[\delta(\ell) - 1]$, and the equation can be rewritten as:

$$C_\ell[k] = \max \begin{cases} C_{\ell-1}[k] \\ C_\ell[\delta(\ell)] & \text{if } k \ge \delta(\ell) \end{cases}$$

Thus, for all $k < \delta(\ell)$, $C_\ell[k] = C_{\ell-1}[k]$. Also, for all $k \ge k_0$, $C_{\ell-1}[k] \ge C_{\ell-1}[k_0] > C_\ell[\delta(\ell)]$, so $C_\ell[k] = \max\{C_{\ell-1}[k], C_\ell[\delta(\ell)]\} = C_{\ell-1}[k]$. On the other hand, for all $\delta(\ell) \le k < k_0$, $C_{\ell-1}[k] \le C_{\ell-1}[k_0] \le C_\ell[\delta(\ell)]$, so $C_\ell[k] = C_\ell[\delta(\ell)]$.

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 $i \ge \delta(\ell)$ and $C_{\ell-1}[i] \le C_\ell[\delta(\ell)]$, then 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 constructing and accessing. In summary, the total time for constructing $T_1, \ldots, T_n$ is $O(\sum_{\ell=1}^{n} (\log n + \alpha_\ell \log n))$. As we insert at most n pairs into these trees, we can delete at most n pairs, and $\sum_{\ell=1}^{n} \alpha_\ell \le n$. Hence, $T_1, \ldots, T_n$ can all be computed in $O(n \log n)$ time. The weight of the maximum weight common subsequence 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 $1 \le i, j \le n$, the weight of their maximum weight common subsequence 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 called MSA (mutated subsequence algorithm) 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 MSA 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 maximum weight common subsequence (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 maximum weight common subsequence (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. 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 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 $B[\delta(i'), \delta(j')]$ 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 similarly.

4.1 Implementation Details of MSA

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 $1 \leq i \leq j \leq n$, 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 $j \geq i$ 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 3. *If* $c = 0$ *or* $j = 0$, $\mathtt{OPT}[c,j] = 0$. *Otherwise*, $\mathtt{OPT}[c,j] =$

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

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 MSA 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, MSA has a bounded performance.

LEMMA 4. *Given two sequences A and B and an integer k, the weight of the pair of k-mutated subsequences found by MSA is at least $1/(3k+1)$ times that of any k-mutated subsequences.*

PROOF. 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 each S_i, there is exactly one distinct j such that S_i equals S'_j or the reverse of S'_j. To prove the claim, we let $X_0(=X^*), X_1, \ldots, X_k(=Y^*)$ be sequences such that X_i is the sequence obtained 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_j, a_{j+1}) a breakpoint if a_j and a_{j+1} are not adjacent at X^*. 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 MSA. 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.

5 EXPERIMENTAL RESULTS

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

5.1 A Simple Software

The first software we have implemented simply applies MSA directly to find out which MUM pairs are likely 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 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 MSA directly to select the MUM pairs that are likely to correspond to conserved genes. MSA 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 to determine an appropriate value dynamically.

The software has been implemented on a PC with 512M RAM and a 2.4GHz CPU. The actual running time of the system depends on the number of MUM pairs and the input parameter k, which range from a few to tens of minutes.

Measurement. We compared the software based on MSA 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 as the *sensitivity* of the output. It gives us an indicator 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 use fifteen pairs of human and mouse chromosomes as our test cases. The size of the chromosomes ranges 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[1]. The set of published genes will be the reference for our evaluation.

We generate the MUM pairs of the DNA sequences and we require that each MUM pair has length ℓ at least 20. MUM pairs with length less than 20 are likely to be noise (Delcher et al., 1999). These MUM pairs serve as input data to our algorithm as well as MUMmer-3 and MaxMinCluster. Details of the data sets are given in Table 2 of the appendix.

The findings: We have tested MSA using different values of k (i.e., number of mutations allowed), and finding that $k=4$ is a sensible setting. Figure 1 shows the coverage and sensitivity of MUMmer-3, MaxMinCluster, and MSA ($k=4$) in the 15 test cases. In general, MSA, has a better coverage and slightly higher sensitivity than both MUMmer-3 and MaxMinCluster. Precisely, MSA has an average coverage of 91%, which is 14% and 7% higher than that of MUMmer-3 and MaxMinCluster, respectively. The average sensitivity of MSA is 29%, which is higher than that of MUMmer-3 and MaxMinCluster by about 3% and 2%, respectively. It is also worth-mentioning that MSA has a higher average number of MUM pairs reported for each known conserved gene; the actual statistics are 23, 25, 28 for MUMmer-3, MaxMinCluster, and MSA, respectively.

In summary, MSA, using a mutation sensitive approach to select the MUM pairs, is able to locate the conserved genes more effectively.

Different values of k: Figure 2 shows the average coverage and sensitivity of MSA for different values of k (i.e., number

[1] http://www.ncbi.nlm.nih.gov/Homology

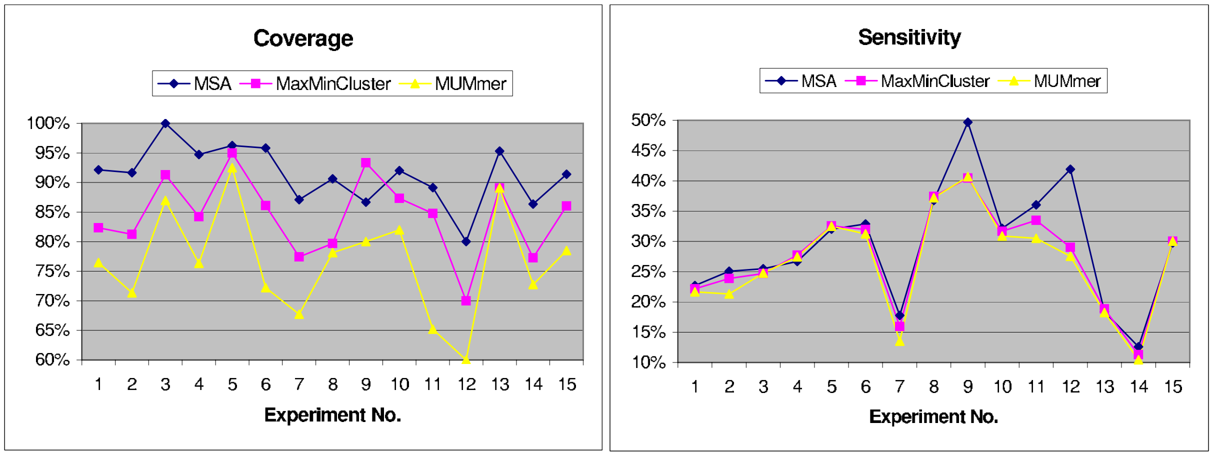


Fig. 1. Performance of MUMmer-3, MaxMinCluster and MSA for aligning mouse and human chromosomes.

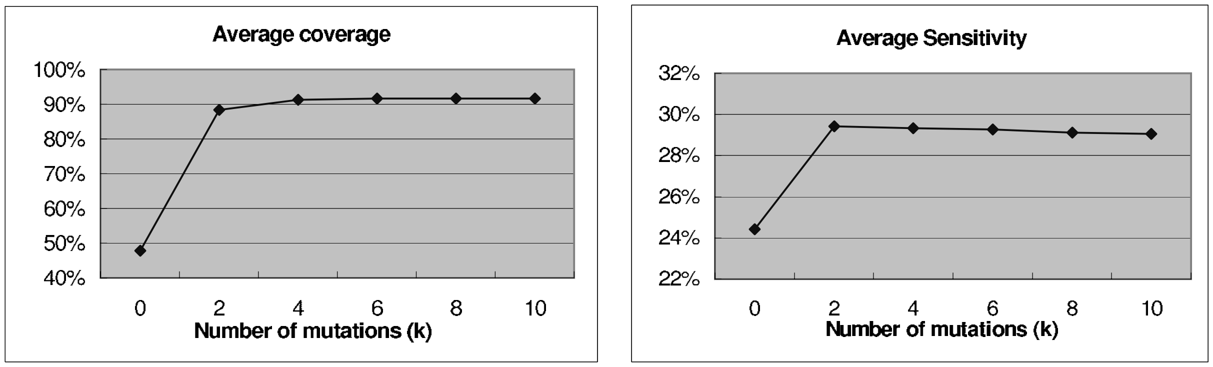


Fig. 2. Average coverage and sensitivity of MSA for different values of k.

of mutations allowed). We observe that $k = 4$ seems to be a sensible setting as it gives high coverage while preserving the sensitivity. 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 thought. It is also known that there are about 100 pairs of human-mouse chromosomes that are related (Mouse Genome Informatics, 2004). The value of 4 seems to be a reasonable estimate on the number of mutations between a pair of mouse and human chromosomes.

5.3 Aligning Translated Protein Sequences

We use pairs of virus genomes from the family *Baculovirdae* as our test cases. The virus genomes are of length 100 thousand to 200 thousand nucleotides and their corresponding conserved genes have been published in the literature (Herniou et al., 2001). Mutations occur more frequently in virus and their DNA sequences show much lower degree of similarity than that of mouse and human. Comparing the translated protein sequences is more useful in analyzing these distant species.

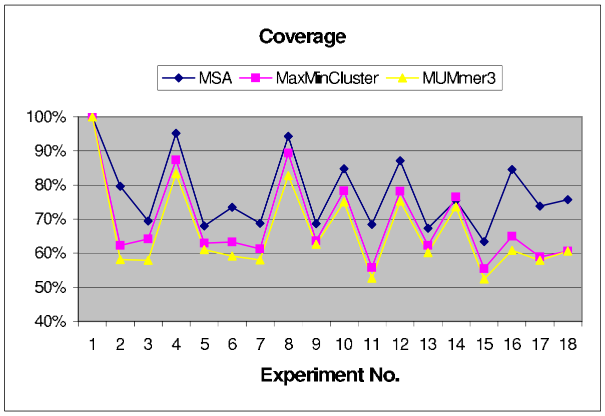
We generate MUM pairs of length at least 3 amino acids. These MUM pairs serves as input to MSA and also as input to MUMmer-3 and MaxMinCluster. Details of the data sets are given in Table 3 of the appendix.

The findings: We first use 18 pairs of Baculovirdae genomes that are within the same genus (either Nucleopolyhedroviruses or Granulovirus). As mutations are often very frequent among viruses, we set k (the number of mutations allowed) to a larger value and $k = 20$ seems to be a sensible setting. Figure 3 shows the coverage and sensitivity of MSA in these 18 test cases. MSA achieves the highest coverage in all except one test case, and it has the highest sensitivity in all the eighteen cases. Specifically, the average coverage of MSA is 78%, while the coverage of MaxMinCluster and MUMmer-3 are 69% and 66% respectively. The sensitivity of the three software are 87%, 75% and 71% for MSA, MaxMinCluster and MUMmer-3 respectively.

Next, we consider 18 pairs of Baculovirdae genomes that are not within the same genus. As one may expect, MSA cannot handle genomes that involve too many mutations and the performance of MSA is significantly inferior to MaxMinCluster and MUMmer-3. The average coverage of MSA is 36%, while MaxMinCluster and MUMmer-3 achieve 45% and 43%, respectively.

5.4 A Better Software

The second software we have implemented adopts a hybrid approach. The aim is to obtain a software that can outperform the clustering algorithms and MSA consistently even for genomes that involve many mutations. The hybrid



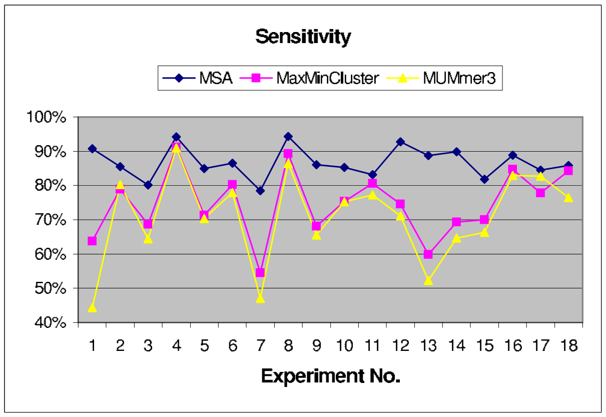
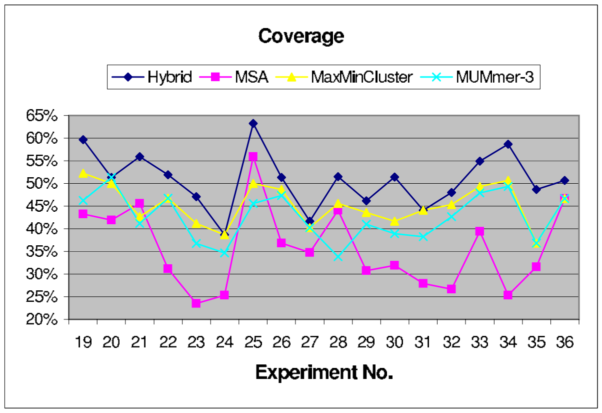


Fig. 3. Performance of MUMmer-3, MaxMinCluster and MSA for aligning the translated protein sequences of 18 pairs of Baculovirdae genomes that are in the same genus.



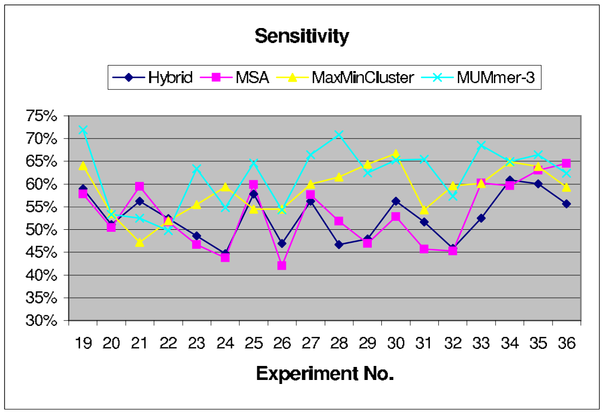


Fig. 4. Performance of the hybrid approach (MaxMinCluster + MSA) for aligning Baculovirdae genomes that are **not** in the same genus.

approach first applies MaxMinCluster to identify some clusters that are obviously conserved genes; these clusters are each treated as an MUM pair with a bigger weight and processed together with the remaining MUM pairs using the MSA. For species that are close, the hybrid approach has the same performance as MSA alone; more specifically, the average coverage is 91% for the case of human and mouse, and 79% for the viruses. For species that might involve a large number of mutations, the hybrid approach differentiates itself from MSA 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 Baculovirdae genomes that are not in the same genus. The hybrid approach can achieve an average coverage of 51% (MUMmer-3, MaxMinCluster, and MSA alone can only attain 43%, 45%, and 36%, respectively), while maintaining the sensitivity at a satisfactory level ($\sim$53%). We have also tested the hybrid approach based on MUMmer-3 plus MSA, the performance is slightly worst than MaxMinCluster plus MSA, achieving an average coverage of 48%.

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

REFERENCES

Altschul, S. F., W. Gish, W. Miller, E. W. Myers, and D. J. Lipman (1990). Basic local alignment search tool. *Journal of Molecular Biology 215*, 403–410.

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

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

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

Chain, P., S. Kurtz, E. Ohlebusch, and T. Slezak (2003). An application-focused review of comaprative genomic tools: Capabilities, limitations and future challenges. *Briefings in Bioinformatics 4*(2), 105–123.

Cole, R., M. Farach-Colton, R. Hariharan, T. M. Przytycka, and M. Thorup (2000). An $O(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees. *SIAM Journal on Computing 30*(5), 1385–1404.

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

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

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

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

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

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

Kaplan, H., R. Shamir, and R. Tarjan (1999). A faster and simpler algorithm for sorting signed permutations by reversals. *SIAM Journal on Computing 29*(3), 880–892.

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

Mouse Genome Informatics (2004). Mammalian orthology and comparative maps. http://www.informatics.jax.org/.

Nadeau, J. H. and B. A. Taylor. (1984). Lengths of chromosomal segments conserved since divergence of man and mous. In *Proceedings of the National Academy of Sciences USA*, Volume 81, pp. 814–818.

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

Pearson, W. R. and D. J. Lipman (1988). Improved tools for biological sequence comparison. In *Proceedings of the National Academy of Sciences USA*, Volume 85, pp. 2444–2448.

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

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

Pevzner, P. and G. Tesler (2003b). Transforming men into mice: the Nadeau-Taylor chromosal breakage model revisited. In *RECOMB*, pp. 244–256.

Sarnak, N. and R. E. Tarjan (1986). Planar point location using persistent search trees. *Communications of the ACM 29*(7), 669–679.

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

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

Wong, P. W. H., T. W. Lam, N. Lu, H. F. Ting, and S. M. Yiu (2004). An efficient algorithm for optimizing whole genome alignment with noise. *Bioinformatics 20*(16), 2676–2684.

APPENDIX

Exp. No.	Mouse Chr. No.	Human Chr. No.	# of Published Conserved Genes	# of MUM pairs
1	2	15	51	96,473
2	7	19	192	52,394
3	14	3	23	58,708
4	14	8	38	38,818
5	15	12	80	88,305
6	15	22	72	71,613
7	16	16	31	66,536
8	16	21	64	51,009
9	16	22	30	61,200
10	17	6	150	94,095
11	17	16	46	29,001
12	17	19	30	56,536
13	18	5	64	131,850
14	19	9	22	62,296
15	19	11	93	29,814

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

Experiment No.	Virus	Virus	# of Published Conserved Genes	# of MUM pairs
1	Ac	Bm	134	20,006
2	Ac	Ha	98	36,630
3	Ac	Ld	95	32,340
4	Ac	Op	126	33,075
5	Ac	Se	100	37,035
6	Bm	Ha	98	35,122
7	Bm	Ld	93	31,012
8	Bm	Op	122	31,750
9	Bm	Se	99	35,881
10	Ha	Ld	92	29,156
11	Ha	Op	95	28,939
12	Ha	Se	101	36,527
13	Ld	Op	98	38,753
14	Ld	Se	102	33,854
15	Op	Se	101	32,771
16	Cp	Px	97	29,661
17	Cp	Xc	107	40,027
18	Px	Xc	99	34,889
19	Ha	Px	67	30,087
20	Ha	Xc	74	41,357
21	Ld	Px	68	27,986
22	Ld	Xc	77	35,554
23	Op	Px	68	27,614
24	Op	Xc	75	35,781
25	Px	Se	68	31,278
26	Se	Xc	76	41,718
27	Ac	Cp	72	34,668
28	Ac	Px	68	31,233
29	Ac	Xc	78	42,680
30	Bm	Cp	72	33,320
31	Bm	Px	68	30,484
32	Bm	Xc	75	40,887
33	Cp	Ha	71	32,687
34	Cp	Ld	75	31,243
35	Cp	Op	76	32,273
36	Cp	Se	75	34,255

Table 3. Details of the data sets for virus genomes from the family *Baculovirdae*.