Pairwise Alignment with Rearrangements

Le Sy Vinh [1] vle@amnh.org
Andrés Varón[1,2] avaron@amnh.org
Ward C. Wheeler[1] wheeler@amnh.org

[1] Division of Invertebrate Zoology, American Museum of Natural History, USA
[2] Computer Science Department, The City University of New York, USA

Abstract

The increase of available genomes poses new optimization problems in genome comparisons. A genome can be considered as a sequence of characters (loci) which are genes or segments of nucleotides. Genomes are subject to both nucleotide transformation and character order rearrangement processes. In this context, we define a problem of so-called *pairwise alignment with rearrangements* (PAR) between two genomes. The PAR generalizes the ordinary pairwise alignment by allowing the rearrangement of character order. The objective is to find the optimal PAR that minimizes the total cost which is composed of three factors: the edit cost between characters, the deletion/insertion cost of characters, and the rearrangement cost between character orders. To this end, we propose simple and effective heuristic methods: *character moving* and *simultaneous character swapping*. The efficiency of the methods is tested on Metazoa mitochondrial genomes. Experiments show that, pairwise alignments with rearrangements give better performance than ordinary pairwise alignments without rearrangements. The best proposed method, simultaneous character swapping, is implemented as an essential subroutine in our software POY version 4.0 to reconstruct genome-based phylogenies.

Keywords: genome comparison, pairwise alignment, gene rearrangements

1 Introduction

A large number of complete genomes has been sequenced thanks to the development of efficient sequencing techniques and many genome projects (`http://www.ncbi.nih.gov/Genomes/`). The genetic divergence between genomes reflects both small-scale and large-scale evolutionary processes. An essential problem in genome comparison is the construction of pairwise alignments between two genomes [9, 18].

The small-scale processes that act on the nucleotide level can be divided into two classes: nucleotide substitution and nucleotide deletion/insertion (indel). In addition to the small-scale processes, the large-scale processes act on locus level. Common operations are character loss, character duplication, and character rearrangement. The first two operations cause the presence and absence of characters. Thus, a character is present in some genomes, but is not in others. Character rearrangements are operations that change the order of characters and can be inversion, transposition, and inverted transposition [13, 14]. Figure 1 illustrates these rearrangement operations on a genome with six characters.

Although traditionally research has been focused primarily at the nucleotide level, recent developments have allowed for analysis of rearrangement at the locus level [5, 6, 11, 12, 13, 14, 15, 17]. The number of rearrangements to convert the character order of one genome to the character order of another genome tracks the genome divergence. The rearrangement distance between two genomes can be simply calculated as the number of breakpoints between two genomes [16]. A more sophisticated measure of rearrangement distance is the number of inversions needed to transform the character order of one genome to the character order of another genome [11].

	Original order of characters	g_1	g_2	g_3	g_4	g_5	g_6
Inversion:	Characters from position 3 to 5 are inverted	g_1	g_2	$-g_5$	$-g_4$	$-g_3$	g_6
Transposition:	Characters from position 3 to 5 are moved to position 1	g_3	g_4	g_5	g_1	g_2	g_6
Inverted transposition:	Characters from position 3 to 5 are moved to position 1, then inverted	$-g_5$	$-g_4$	$-g_3$	g_1	g_2	g_6

Figure 1: Three types of rearrangements on a genome with six characters

A typical approach to compare the evolutionary relationships among species is to analyze the nucleotide transformations among homologous sequences. The approach is sufficient for most sequences, except distantly related sequences where nucleotide variations might be saturated. It is possible that character orders of species are more conserved than nucleotides, thus, they can be used properly to overcome this problem [4, 5]. However, character rearrangement-based approach is not applicable to closely related species because their character orders are usually identical. Moreover, it ignores all nucleotide transformations which might be orders of magnitude more information than character orders [23].

Recently, Ward Wheeler proposed an approach incorporating information from both small-scale and large-scale processes to reconstruct genome-based phylogenies [23]. Characters can be determined by annotations from Genbank at NCBI (`http://www.ncbi.nih.gov/`), i.e., segments within and between annotated regions. They can be also determined automatically by algorithms BLASTZ [18], Glocal [7] or Mauve [8].

In this paper, we formulate a generalized version of pairwise alignment between two genomes presented as two sequences of characters where the rearrangement of character order is allowed. The objective is to construct the optimal pairwise alignment with rearrangements (PAR) between two genomes that minimize the total cost to convert one genome into another genome. The total cost is composed of the edit cost between characters, insertion/deletion (indel) cost of characters, and rearrangement cost between character orders.

To motivate the necessary of constructing pairwise alignment with rearrangements, the total cost can be used as a comprehensive measure of the genetic divergence between two genomes. Moreover, constructing pairwise alignments with rearrangements is a crucial task in constructing genome-based phylogenies using dynamic homology [22, 23, 24, 25]. It also can be used as a central subroutine to construct multiple alignment with rearrangements using progressive multiple alignment algorithms [19].

The rest of paper is organized as follows: section 2 provides a detailed description of the PAR problem. Since an exact solution for the PAR problem is likely intractable, we propose simple heuristic methods to search efficiently optimal PAR in section 3. All methods are described explicitly in an algorithm-based style in order to estimate their time complexity. In section 4, we give a short introduction to Metazoa mitochondrial genomes in which character rearrangements have been reported [3]. Then the efficiency of methods based on 760 Metazoa mitochondrial genomes collected from Genbank at NCBI is examined using different criteria. Finally, discussions and open questions are addressed in section 5.

2 Pairwise Alignment with Rearrangements

In this section, we mathematically define the PAR problem between two genomes as two sequences of characters. To do that, we first introduce some notations and the ordinary pairwise alignment problem. Let $\alpha = \{x_1, \ldots, x_p, y_1, \ldots, y_q, \text{`}-\text{'}\}$ be an alphabet of $(p+q)$ characters as well as a special gap character '$-$' (we also denote x_0/y_0 as the gap character). Let $X = (x_1, x_2, \ldots, x_p)$ be a sequence of p characters, $Y = (y_1, y_2, \ldots, y_q)$ be a sequence of q characters . Let $C(x_i, y_j) \in R^+$ be the edit cost

x_1	x_2	x_3	$-$	x_4	x_5
y_4	$-$	y_3	y_1	y_2	$-$

Figure 2: A pairwise alignment with rearrangements between two sequences $X = (x_1, x_2, x_3, x_4, x_5)$ and $Y = (y_1, y_2, y_3, y_4)$.

to transform character x_i into character y_j for $i = 1 \ldots p,\ j = 1 \ldots q$. The edit cost matrix C is metric. Note that $C(x_i, y_0)$ and $C(x_0, y_j)$ are costs to delete/insert characters x_i and y_j, respectively. Since x_i and y_j are segments of nucleotides, the edit cost $C(x_i, y_j)$ can be calculated as the minimum number of nucleotide transformations between x_i and y_j. Let $R(Y, Y_r)$ be the rearrangement cost function between sequence Y and its permutation Y_r. Typically, $R(Y, Y_r)$ is computed as the breakpoint distance or inversion distance [16, 11].

A sequence $X' = (x'_1, x'_2, \ldots, x'_l)$ is called an *edited sequence* of X if and only if X is obtained from X' by deleting all gap characters. In other words, sequence X is called the *gapless sequence* of X'. For example, $X' = (\text{`}-\text{'}, 1, 2, \text{`}-\text{'}, 3, 4)$ is an edited sequence of $X = (1, 2, 3, 4)$. A pair $A(X, Y) = (X', Y')$ of two edited sequences $X' = (x'_1, x'_2, \ldots, x'_l)$ and $Y' = (y'_1, y'_2, \ldots, y'_l)$ is called an *ordinary pairwise alignment* of sequences X and Y. The cost $\mathbf{C}(A)$ of alignment A is the sum of edit cost between characters and indel cost of characters as shown below:

$$\mathbf{C}(A = (X', Y')) = \sum_{i=1}^{l} C(x'_i, y'_i). \tag{1}$$

The optimal ordinary pairwise alignment $A^*(X, Y) = \text{argmin}_{A(X,Y)}\{\mathbf{C}(A)\}$ can be constructed in $O(pq)$ time using dynamic program technique [21]. An ordinary pairwise alignment is constructed by inserting gap characters into both sequences conditioned that the order of characters in both sequences must be kept unchanged. Since the order of characters (loci) can be rearranged in real data, this unrealistic condition needs to be relaxed.

A sequence $X'_r = (x'_1, x'_2, \ldots, x'_l)$ is called an *edited rearrangement sequence* of X if the gapless sequence X_r of X'_r is a permutation of X. For instance, $X'_r = (\text{`}-\text{'}, 1, 4, 2, \text{`}-\text{'}, 3)$ is an edited rearrangement sequence of $X = (1, 2, 3, 4)$. We are now ready to formulate the pairwise alignment with rearrangements problem. A pair $A_r = (X'_r, Y'_r)$ of edited rearrangement sequences $X'_r = (x'_1, x'_2, \ldots, x'_l)$ and $Y'_r = (y'_1, y'_2, \ldots, y'_l)$ is called a *pairwise alignment with rearrangements* (PAR) of two sequences X and Y. The cost $\mathbf{C}_r(A_r)$ of PAR A_r is the sum of edit cost between characters, indel cost of characters and rearrangement cost between character orders. Precisely:

$$\mathbf{C}_r(A_r = (X'_r, Y'_r)) = \sum_{i=1}^{l} C(x'_i, y'_i) + R(X, X_r) + R(Y, Y_r) \tag{2}$$

The objective is to find the optimal PAR A^*_r which minimizes the total cost. Mathematically,

$$A^*_r = \text{argmin}_{A_r}\{\mathbf{C}_r(A_r)\}. \tag{3}$$

Figure 2 illustrates a PAR A_r between two sequences $X = (x_1, x_2, x_3, x_4, x_5)$ and $Y = (y_1, y_2, y_3, y_4)$. Note that the PAR problem does not require genomes to have the same number of characters. Since an exact solution for PAR problem is likely intractable, heuristic approaches which compromise between computational expense and alignment quality need to be developed.

3 Algorithms

In this section, we propose hill climbing methods to search the optimal PAR A_r^* sufficiently. In general, the methods compose two phases. In the first parse, an initial PAR A_r is constructed. Then the second phase searches the optimal PAR A_r^* by improving the quality of A_r in a step-by-step manner.

Algorithm 1: Stepwise addition method

```
1.1  begin
1.2      Y_r ← ∅
1.3      for j = 1 to |Y| do
1.4          bestCost ← +∞;  bestP ← nil
1.5          foreach position p in Y_r do
1.6              Insert y_j into Y_r at position p
1.7              A_r ← A*(X, Y_r)
1.8              if C_r(A_r) < bestCost then
1.9                  bestCost ← C_r(A_r);  bestP ← p
1.10             Remove y_j from Y_r
1.11         end
1.12         Insert y_j into Y_r at position bestP
1.13     end
1.14     return A*(X, Y_r)
1.15 end
```

In particular, the first phase creates an initial PAR A_r by employing *stepwise addition* algorithm. The algorithm starts from an incomplete PAR $A_r = (X_r' = X, Y_r' = \emptyset)$ and sequentially inserts characters $y_j \in Y, j = 1 \ldots |Y|$ into the nascent Y_r' to construct a final PAR A_r. Character y_j is inserted into the position such that the cost of new PAR A_r is minimized. The method is explained in details in algorithm 1.

The second phase is the modification mechanism to improve the quality of current A_r [1]. To this end, we propose character moving and simultaneous character swapping techniques.

3.1 Character Moving

Consider a PAR $A_r = (X_r', Y_r')$, an operation $M(i, j, t \mid i \leq j < t - 1)$ on gapless sequence $Y_r = (y_1, y_2, \ldots, y_q)$ moves characters $(y_i, \ldots, y_j)$ to position t resulting in new gapless sequence $\mathcal{Y}_r$. Thus, we have a new PAR $\mathcal{A}_r = A^*(X_r, \mathcal{Y}_r)$ with new cost $\mathbf{C}_r(\mathcal{A}_r)$. A move $M(i, j, t)$ is said to be *possible* if and only if $\mathbf{C}_r(\mathcal{A}_r) < \mathbf{C}_r(A_r)$. In other words, the move $M(i, j, t)$ leads to a new PAR $\mathcal{A}_r$ with smaller cost than current A_r. The character moving algorithm applies sequentially possible moves to improve the current PAR. The process is repeated until no possible move is available (see Algorithm 2).

3.2 Simultaneous Character Swapping

Here we describe a more efficient and complicated technique to refine current PAR $A_r = (X_r', Y_r')$. An operation $S(k, t)$ on gapless sequence $Y_r = (y_1, y_2, \ldots, y_q)$ swaps two characters y_k and y_t to create a new gapless sequence $\mathcal{Y}_r = (y_1, \ldots, y_{k-1}, y_t, y_{k+1}, \ldots, y_{t-1}, y_k, y_{t+1}, \ldots, y_q)$. The swapping cost of operation $S(k, t)$ is the cost $\mathbf{C}_r(\mathcal{A}_r)$ of new PAR $\mathcal{A}_r = A^*(X_r, \mathcal{Y}_r)$. Similar to character moving, we can simply apply possible swaps to refine current A_r. However, an observation is that a swap $S(k, t)$ can be possible at the current time, but might not be possible at the next time due to the performance

Algorithm 2: Character moving method

```
2.1  begin
2.2      Build an initial PAR A_r = (X'_r, Y'_r) by stepwise addition algorithm
2.3      iteration ← 0
2.4      repeat
2.5          possibleMove ← false
2.6          foreach triple positions (i, j, t | i ≤ j < t − 1) in Y_r do
2.7              if M(i, j, t) is a possible move then
2.8                  Move characters (y_i, ..., y_j) in Y_r to position t
2.9                  possibleMove ← true
2.10             end
2.11         end
2.12         iteration ← iteration + 1
2.13     until possibleMove = false
2.14     return A*(X_r, Y_r)
2.15 end
```

of another swap. If multiple possible swaps are available at the same time, a ranking function is needed to determine their priority. The swapping cost of $S(k,t)$ can be used as a proper ranking function for swaps. More precisely, the swap with the smallest cost will be executed first. However, execution time is a problem if we perform only the smallest cost swap at each time. Inspired by the success of simultaneous nearest neighbor interchanges technique in phylogenetic tree reconstruction [10, 20], we propose a simultaneous character swapping algorithm to overcome the problem.

Consider two possible swaps $S(k_1,t_1)$ and $S(k_2,t_2)$, without losing the generality we assume that $k_1 < t_1$, $k_2 < t_2$, and $k_1 < k_2$. $S(k_1,t_1)$ and $S(k_2,t_2)$ are said to be *independent* if either $k_2 > t_1$ or $t_2 < t_1$ as illustrated in Figure 3 (left). Experiments show that performing simultaneously two independent possible swaps typically leads to a better PAR (data not shown). If two possible swaps are not independent, the swap with higher cost is ignored.

The idea is to swap simultaneously a set of r best independent possible swaps (see Figure 3). If multiple r independent possible swaps does not lead to a better PAR, the number swaps r is reduced to $r/2$. The swapping process is performed iteratively until $r = 1$. It is guaranteed that a better PAR will be obtained when $r = 1$. The simultaneous character swapping process terminates when no possible swap is available. The final PAR is considered as the optimal one. The method is described in details in algorithm 3.

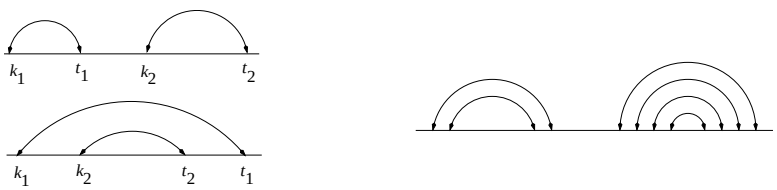

Figure 3: Two possible swaps $S(k_1,t_1)$ and $S(k_2,t_2)$ are independent (left). A simultaneous swap of independent possible swaps presented by half circles (right).

Algorithm 3: Simultaneous character swapping method

```
3.1  begin
3.2      Build an initial PAR A_r = (X'_r, Y'_r) by stepwise addition algorithm
3.3      minCost ← C_r(A_r)
3.4      iteration ← 0
3.5      repeat
3.6          possibleSwap ← false
3.7          Find independent possible swaps L and sort L increasingly
3.8          if |L| > 0 then
3.9              r ← |L|
3.10             repeat
3.11                 Swap simultaneously L_1, ..., L_r in Y_r to get 𝒴_r
3.12                 𝒜_r ← A*(X_r, 𝒴_r)
3.13                 if C_r(𝒜_r) < minCost then
3.14                     possibleSwap ← true
3.15                     minCost ← C_r(𝒜_r);  Y_r ← 𝒴_r
3.16                 else
3.17                     r ← r/2
3.18             until possibleSwap = true
3.19         end
3.20         iteration ← iteration + 1
3.21     until possibleSwap = false
3.22     return A*(X_r, Y_r)
3.23 end
```

3.3 Complexity

To estimate the time complexity of algorithms, we assume that the edit costs $C(x_i, y_j)$ and the indel cost of characters are computed in advance. To simplify the time complexity estimation, the rearrangement cost $R(Y, Y_r)$ is computed as the breakpoint distance [16]. Thus, the time complexity of calculating the rearrangement cost $R(Y, Y_r)$ is $O(q)$.

First, we compute the time complexity of constructing a PAR A_r using the stepwise addition method described in algorithm 1. Since the time complexity of lines 1.6 to 1.10 is $O(pq)$, the time complexity from line 1.5 to line 1.11 is $O(pq^2)$. Therefore, the algorithm can be completed in $O(pq^3)$ time.

Second, we estimate the time of character moving in algorithm 2. Line 2.2 requires $O(pq^3)$ time. Since lines 2.7 to 2.10 can be done in $O(pq)$ time, lines 2.6 to 2.11 take $O(pq^4)$ time. In short, the time complexity of character moving method is $O(pq^4 \times iteration + pq^3) = O(pq^4 \times iteration)$.

Finally, we calculate the time complexity of the simultaneous character swapping method in algorithm 3. Line 3.2 takes $O(pq^3)$ time. The time complexity of line 3.7 is $O(pq^3)$. Since $|L| \leq q$, lines 3.8 to 3.19 are completed in $O(pq^2)$. Therefore, lines 3.6 to 3.20 are done in $O(pq^3 + pq^2) = O(pq^3)$. In summary, the time complexity of the simultaneous character swapping algorithm is $O(pq^3 \times iteration + pq^3) = O(pq^3 \times iteration)$.

4 Experiments

4.1 Data

To date, thousands of mitochondrial genomes (mtDNA) have been sequenced. In this paper, we collected 760 Metazoa (animals) mtDNA genomes from Genbank at NCBI (`http://www.ncbi.nih.gov/`) to test the method. Metazoa mtDNA genomes are typically closed-circular, about 16kb long, and usually contain the same 37 genes [3]. It is known that mtDNA genomes undergo not only nucleotide transformations but also gene arrangement processes. A survey and summary of Metazoa mtDNA genomes is given by Boore [3].

We divide each genome into sequence segments based on annotations from Genbank at NCBI, e.g., segments within and between annotated regions. Segments without annotations are also kept in the analysis. Each segment is now considered as a character. On average, each genome contains a sequence of about 50 characters. The cost between two characters x and y is calculated as the minimum number of nucleotide transformations between x and y. More precisely, the nucleotide substitution cost, nucleotide indel cost are set to 1 and 2, respectively. The indel cost of a character x or y equals to the number of its nucleotides.

Since we are not able to test all genome pairs ($\sim$ 32 million), 760 genomes are divided randomly into 38 groups each containing 20 genomes. The PAR between any two genomes in the same group is constructed. Precisely, 7980 pairs of genomes were analyzed.

4.2 Measurements

The quality of a PAR $A_r = (X'_r, Y'_r)$ is measured by both criteria: the cost $\mathbf{C}_r(A_r)$ and the percentage of correctly aligned pairs $H(X'_r, Y'_r)$ as calculated below:

$$H(X'_r, Y'_r) = \frac{\sum_{i=1}^{|A_r|} \sigma(X'_r(i), Y'_r(i))}{\text{Number of pairs } (x_i, y_j) \text{ where } x_i \text{ and } y_j \text{ are annotated the same}} \times 100 \tag{4}$$

where

$$\sigma(X'_r(i), Y'_r(i)) = \begin{cases} 1 & \text{If } X'_r(i) \text{ and } Y'_r(i) \text{ are characters with the same annotations} \\ 0 & \text{Otherwise} \end{cases} \tag{5}$$

Note that, the smaller cost denotes the better alignment. In contrast, the higher percentage of correctly aligned pairs $H(X'_r, Y'_r)$ is the better alignment.

4.3 Results

We compared the quality of PAR constructed from different methods: stepwise addition, character moving, and simultaneous character swapping. To test the effect of rearrangements, we also compared the quality of PAR to the best ordinary pairwise alignment $A^*(X, Y)$.

Table 1(a) shows that the PAR gives better result than the ordinary pairwise alignment. For example, the cost of the PAR constructed by simultaneous character swapping method is approximately 16% less than that of the ordinary pairwise alignment. The percentage of correctly aligned pairs of simultaneous character swapping method is approximately 14% higher than that of the ordinary pairwise alignment. Table 1(a) shows that simultaneous character swapping is the best method. It is better than character moving method both in terms of alignment cost and percentage of correctly aligned pairs.

Table 1: The performance of different methods based on Metazoa mitochondrial genomes.

Name	Alignment cost	Percentage of correctly aligned pairs
Ordinary alignment	10154	66
Stepwise addition	8826	66
Character moving	8684	66
Simultaneous character swapping	8490	75

(a) The rearrangement cost is calculated as the breakpoint distance

Name	Alignment cost	Percentage of correctly aligned pairs
Ordinary alignment	10154	66
Stepwise addition	8824	66
Character moving	8687	66
Simultaneous character swapping	8489	75

(b) The rearrangement cost is calculated as the inversion distance

We tested the effect of rearrangement cost function $R(Y, Y_r)$ to the PAR. To this end, the cost function $R(Y, Y_r)$ is measured as the inversion distance between Y and Y_r using the GRAPPA program [2]. Table 1(b) shows similar results to the Table 1(a). That is to say, in these experiments inversion distance does not boost the quality of PAR compared to breakpoint distance.

Both experiments show that simultaneous character swapping performs better than all other methods tested. Moreover, the simultaneous character swapping needs only approximately 5 refining iterations to obtain the optimal PAR (see Algorithm 3). That means the multiple character swapping is much faster than character moving and as fast as the stepwise addition algorithm.

5 Conclusion

Since number of available genomes is increasing quickly, using total information, i.e., nucleotide transformation, character loss, character duplication, and character rearrangement information, is necessary in genome comparisons. We defined the pairwise alignment with rearrangements problem between two genomes presented as two sequence of characters. The objective is to find the optimal alignment that minimizes the cost to transform one genome to another genome.

The ordinary pairwise alignment between two sequences X and Y can be done in $O(pq)$ time where p and q are numbers of characters in sequences X and Y, respectively. However, the pairwise alignment with rearrangements is much more difficult. It can be solved exactly by examining ordinary pairwise alignments between sequence X and all permutation sequences Y_r of Y. The time complexity of the exact approach is $O(pq \times q!)$. Thus, it is not applicable to sequences with more than 20 characters.

To solve the PAR problem, we proposed simple and efficient heuristic methods. The combination of stepwise addition method and simultaneous character swapping technique is the most efficient approach to search the optimal PAR. The approach needs less than one minute to construct an optimal PAR for Metazoa mtDNA genomes on a 3.0 GHz PC.

Two rearrangement cost functions, i.e., breakpoint distance and inversion distances were tested. Interestingly, the inversion distance does not overcome the simple breakpoint distance in these experiments. Of course, proposed algorithms are not restricted to breakpoint and inversion distance, they can work with any rearrangement cost function.

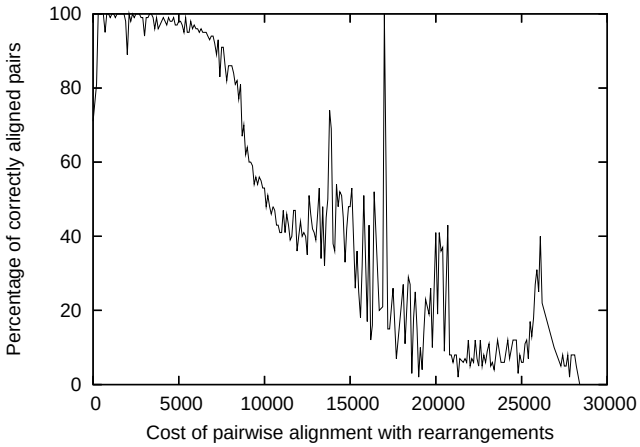

Figure 4: The relationship between the cost of pairwise alignment with rearrangements and the percentage of correctly aligned pairs. The result is from simultaneous character swapping method.

It is not surprising to see from Figure 4 that the percentage of correctly aligned pairs typically increases with a decreasing alignment cost. When the alignment cost is smaller than 5000, we get a high probability that characters with the same annotations are paired correctly.

There are some open problems involving with the rearrangements of character orders. Although we have proposed simple heuristic methods, more efficient approaches to construct the pairwise alignment with rearrangements certainly must be developed. Moreover, comparisons between phylogenies which are constructed with and without character rearrangements are necessary to examine. Last but not least, analysis of multiple alignment with rearrangements which is much more difficult than analyzing the pairwise alignment with rearrangements needs to be studied in the near future.

Acknowledgments

We would like to thank Megan Harrison for her carefully reading the manuscript. This paper is based upon work supported by the U.S. Army Research Laboratory and the U.S. Army Research Office under grant "Novel analytical and empirical approaches to the origin and prediction of pathogenicity" (#W911NF-05-1-0271) and NSF-ITR grant "Information technology research: Phyloinformatics."

References

[1] Aarts, E. and Lenstra, J. K., *Local Search in Combinatorial Optimization*, Wiley, Chichester, UK, 1997.

[2] Bader, D. A., Morret, B. M., and Yan, M., A linear-time algorithm for computing inversion distance between signed permutations with an experimental study, *Journal Computational Biology*, 8:483–491, 2001.

[3] Boore, J. L., Animal mitochondrial genomes, *Nucleic Acids Res.*, 27:1767–1780, 1999.

[4] Boore, J. L. and Brown, W. M., Big trees from little genomes: mitochondrial gene order as a phylogenetic tool, *Curr Opin Genet Dev*, 8:668–674, 1998.

[5] Boore, J. L., Lavrov, D. V., and Brown, W. M., Gene translocation links insects and crustaceans, *Nature*, 392:667–668, 1998.

[6] Bourque, G. and Pevzner, P. A., Genome-scale evolution: Reconstructing gene orders in the ancestral species, *Genome Res.*, 12:26–36, 2004.

[7] Brudno, M., Malde, S., and Poliakov, A., Do, C. B., Couronne, O., Dubchak, I., and Batzoglou, S., Glocal alignment: finding rearrangements during alignment, *Bioinformatics*, 19:i54–i62, 2003.

[8] Darling, A. C., Mau, B., Blattner, F. R., and Perna, N. T., Mauve: Multiple alignment of conserved genomic sequence with rearrangements, *Genome Res.*, 14:1394–1403, 2004.

[9] Delcher, A. L., Kasif, S., Fleishman, R., Peterson, J., White, O., and Salzberg, S. L., Alignment of whole genomes, *Nucleic Acids Res.*, 27:2369–2376, 1999.

[10] Guindon, S. and Gascuel, O., A simple, fast and accurate algorithm to estimate large phylogenies by maximum likelihood, *Syst. Biol.*, 52:696–704, 2003.

[11] Hannenhalli, S. and Pevzner, P. A., Transforming cabbage into turnip. (polynominal algorithm for sorting signed permutations by reversals, *Proceedings of the 27th Annual ACM-SIAM Symposium on the Theory of Computing*, 178–189, 1995.

[12] Larget, B., Simon, D. L., Kadane, J. B., and Sweet, D., A bayesian analysis of metazoan mitochondrial genome arrangements, *Mol. Biol. Evol.*, 22:486–495, 2004.

[13] Moret, B. M., Tang, J., Wang, L.-S., Warnow, T., Steps toward accurate reconstructions of phylogenies from gene-order data, *J. Comput. Syst. Sci.*, 65:508–525, 2002.

[14] Nadeau, J. H. and Talor, B. A., Lengths of chromosome segments conserved since divergence of man and mouse, *Proc. Natl. Acad. Sci. USA*, 81:814–818, 1984.

[15] Sankoff, D., Genome rearrangement with gene families, *Bioinformatics*, 15:909–917, 1999.

[16] Sankoff, D. and Blanchette, M., Multiple genome rearrangement and breakpoint phylogeny, *J. Comput. Biol.*, 5:555–570, 1998.

[17] Sankoff, D. and Nadeau, J. H., *Comparative Genome: Empirical and Analytical Approaches to Gene Order Dynamics, Map Alignment and the Evolution of Gene Families*, Kluwer, first edition, 2000.

[18] Schwartz, S., Kent, W. J., Smit, A., Zhang, Z., Baertsch, R., Hardison, R. C., Haussler D., and Miller, W., Human mouse alignments with BLASTZ, *Genome Res.*, 13:103–107, 2003.

[19] Thompson, J. D., Higgins, D. G., and Gibson, T. J., CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice, *Nucleic Acids Res.*, 22, 1994.

[20] Vinh, L. S. and von Haeseler, A., IQPNNI: Moving fast through tree space and stopping in time, *Mol. Biol. Evol.*, 21:1565–1571, 2004.

[21] Waterman, M. S., *Introduction to Computational Biology*, Chapman and Hall, London, UK, 1995.

[22] Wheeler, W. C., Alignment, dynamic homology, and optimization, Albert, V., ed., *Parsimony, Phylogeny, and Genomics*, Oxford University Press, 73–80, 2005.

[23] Wheeler, W. C., Chromsomal character optimization, *Mol. Biol. Evol.*, Submitted.

[24] Wheeler, W. C., Fixed character states and the optimization of molecular sequence data, *Cladistics*, 15(4):379–385, 1999.

[25] Wheeler, W. C., Optimization alignment: The end of multiple sequence alignment in phylogenetics?, *Cladistics*, 12(1):1–9, 1996.