An Improved Algorithm for Genome Rearrangements

Thang N. Dinh, Huan X. Hoang, Le Sy Vinh
CS Department, College of Technology,
Vietnam National University, Hanoi, Vietnam
Email: {thangdn, huanhx, vinhls}@coltech.vnu.edu

***Abstract*— A remarkable pattern of evolutionary is that many species have closely related gene sequences but differ dramatically in gene order. It raises a new challenge in aligning two genome sequences that we have to consider changes at both the nucleotide level and the locus level such as gene rearrangements, duplication or loss. Finding the series of rearrangements at the same time with changes at nucleotide level has been formulated as a combinatorial optimization problem so called *Pairwise alignment with rearrangements* by Vinh et. al. The best presented algorithm for this problem, however, still has high computational cost and is impractical for large genome sequences. We propose an improved algorithm for aligning two genome sequence considering rearrangements of segments in genomes. We experiments our algorithm on Metazoa mitochondrial and artificially generated genomes to show the improvement in both alignment quality and computing complexity. Finally, the proposed algorithm is integrated into an evolutionary framework to further reduce the gap to the optimal solution.**

***Index Terms*— Genome rearrangement, Incremental algorithm, Metazoa mitochondrial genomes, Memetic algorithm**

I. Introduction

The alignment cost between two genomes can be used as a measure of genetic divergence between them. Besides local evolution within individual genes, genomes also undergo large-scale events such as horizontal gene transfer, gene rearrangement, gene duplication, gene loss and insertion. Recently it was realized that micro rearrangements are abundant in genomic DNA [3]. They have been shown to be responsible for numerous heritable diseases, called genomic disorders and also in cancer [1] [10] [17] [20].

Such large-scale operations break our previous understanding about alignments and have demanded new methods of aligning whole genomes. Some methodologies for pairwise genome alignments have developed for comparing the human and mouse genomes. These approaches were based either on local alignment [14] [19] or on a local/global technique, in which the mouse contigs are mapped on the human genome by a local aligner initially, and then the homology is confirmed and refined by a global aligner [7].

Pairwise genome alignment tools such as MUMmer [9], GLASS [4], and WABA [12] use anchoring for the alignment process. The aligner identifies a set of local alignments in regions of high similarity among the genomes and anchors are used to restrict the number of possible alignments and then alignment is performed using dynamic programming.

Previous approaches use only information at nucleotides level and genomes are presented as sequences of nucleotides. However, information from both nucleotide and locus levels needs to be incorporated together to infer the genome alignment. We present genomes as sequences of characters that 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 [19], Glocal [6], or Mauve [8].

We consider the small-scale processes that act on the nucleotide level: nucleotide substitution and nucleotide deletion/insertion (indel) and the large-scale processes act on locus level: character loss, character duplication, and character rearrangement. Given two genomes, the objective is to minimize the total cost to transform from one genome to the other using operations from both small-scale and large-scale levels. The total cost consists of two parts: the edit cost between characters, insertion/deletion(indel) cost of characters, and rearrangement cost between character orders. The rearrangement cost between two genomes can be calculated as the number of breakpoints between two genomes or the number of inversions need to transform the character order of one genome to the character order of another genome.

The optimal pairwise alignment with rearrangements of two genomes can be found by examining all permutations of characters in genomes. However, this exact approach is intractable and intolerable even for small instances.

II. Problem Formulation

Given two genomes formulated as two sequences of characters

$$X = (x_1, x_2, \ldots, x_p) \text{ and } Y = (y_1, y_2, \ldots, y_q)$$

In an alignment of X and Y, gaps are inserted between the characters so that identical or similar characters in X and Y are aligned in successive columns. A sequence X is a gapless sequence of X' if X can be obtained from X' by removing all gap characters. For example $X = (1, 2, 4, 3)$ is a gapless sequence of $X' = (1,'-', 2, 4,'-', 3,'-')$. We denote the set of gapless sequences of X by $\mathcal{G}(X)$. Formally an alignment of a pair of sequences (X, Y) is

defined as a pair of sequences (X', Y') satisfies $X' \in \mathcal{G}(X), Y' \in \mathcal{G}(Y)$ and $|X'| = |Y'|$.

The cost of an alignment (X', Y') is defined as

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

where $c(x_i, y_j) \in R^+$ be the edit cost to transform character x_i to character y_j. For convenient, we also denote $c(x_i, y_0) = c(x_i, '-')$ the cost to delete/insert character x_i and $c(x_0, y_j) = c('-', y_j)$ the cost to indel character y_j. The cost $c(x_i, y_j)$ is often calculated as the edit distance between two characters x_i and y_j. Recall that each character is a gene or a segment of nucleotides in a genome.

There is an infinite number of alignments of two sequences (X, Y). However, we are only interested in the *ordinary alignment* of (X, Y) i.e the alignment (X_o, Y_o) with the smallest cost

$$(X_o, Y_o) = \underset{(X', Y') \in \mathcal{G}(X) \times \mathcal{G}(Y)}{\arg\min} C(X', Y')$$

We also denote $O(X, Y)$ the cost of the ordinary alignment of X, Y that can be computed using dynamic program technique [15] with a computational complexity $O(pq)$.

Denote $\mathcal{PM}(X)$ the set of all permutations of the sequences X. Let $R(X, X_r)$ be the rearrangement cost function between sequences of characters X and its permutation $X_r \in \mathcal{PM}(X)$. We can think of the rearrangement cost as the number of needed operations to transform X to X_r. Popular measurements of rearrangements cost $R(X, X_r)$ are breakpoint distance and inversion distance [2] [11] [18].

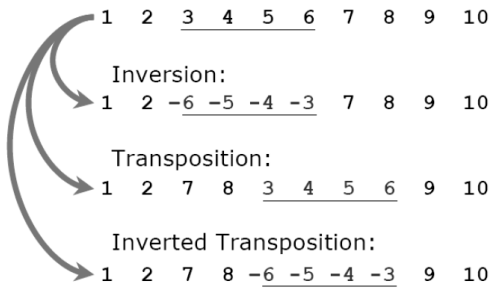

Figure 1. Three types of rearrangements on a genome, pictures from [22]

The problem *pairwise alignment with rearrangement* (PAR) of two genome sequences has been defined by Vinh et al. [21]. A PAR of two sequences X and Y is a pair of sequences $(X'_r = (x'_1, x'_2, \ldots, x'_l), Y'_r = (y'_1, y'_2, \ldots, y'_l))$ such that there exists permutations X_r, Y_r of X and Y respectively and (X_r, Y_r) are gapless sequences of (X'_r, Y'_r). We denote $\mathcal{PAR}_{(X,Y)}$ the set of all PAR of (X, Y) The cost of a PAR $A_r = (X'_r, Y'_r)$ will be sum of submission/insertion/deletion cost of characters and rearrangement cost between character orders:

$$\mathcal{C}(A_r(X'_r, Y'_r)) = \sum_{i=1}^{l} C(x'_i, y'_i) + [\ R(X, X_r) + R(Y, Y_r)\ ] \quad (1)$$

The goal is to find an optimal PAR $A^*_r = (X'_r, Y'_r)$ with the minimum total cost. Finding the optimal solution by considering all pairs of permutation of X and Y will take $O(p!q! \times p \times q)$ and is conjectured to be in the NP-Complete class. When p, q are about 20, that will be out of computing capacity of current computer systems.

A. Breakpoint Distance

Given two sequences with same number of characters

$$X = (x_1, x_2, ..., x_p) \text{ and } X' = (x'_1, x_2, ..., x'_p)$$

We say (x_i, x_{i+1}) is an adjacent pair in $X = (x_1, x_2, ..., x_p)$. We also say (Δ, x_1) and (x_p, ∇) are adjacent pairs, where Δ and ∇ are special characters that do not appear in X. If two characters x and y form an adjacent pair in X but not in X', they determine a breakpoint in X. The breakpoint distance between X and X' is the number of breakpoints in X. and is denoted by $B(X, X')$. The number of breakpoint between two genomes has some relation to the edit distance as half the number of breakpoints is a lower bound on the reversal distance. Both Breakpoint distance (and Inversion Distance) can be calculated in linear time [2] [11] [18].

B. Previous Work

Vinh et al. introduced three hill-climbing algorithms to find the PAR of two genome sequences (X, Y), namely *Stepwise Addition*, *Character Moving* and *Simultaneous Character Swapping*. Among those, Simultaneous Character Swapping(SCS) outperforms other algorithms. The technique to reduce the running time in SCS is performing a set of best independent possible swaps at once. It considers a set of simultaneously swappings of characters at each iteration until no swapping can reduce the cost of PAR. However, SCS still has the running time of $O(n^4 \times \text{iteration})$ where $n = p + q$ because it has to recalculate the ordinary alignment of two sequences after every swap of characters in sequences.

Then cost of the optimal PAR can be calculated as:

$$\mathcal{C}(A^*_r(X^*_r, Y^*_r)) = \min\{O(X_r, Y_r) + R(X, X_r) + R(Y, Y_r) \mid X_r, Y_r \text{ permutations of } X, Y\ \} \quad (2)$$

C. A Naive Local Search Algorithm

One simple algorithm to find a PAR is that we start with sequences (X, Y) and try to perform character swappings in X or Y that lead to reduction of the total cost. After a swap, assume that we have a new pair of sequences (X', Y'). We need to recalculate the ordinary alignment of (X', Y') to see the cost reduction. This nave algorithm also require $O(n^4)$ time for each iteration. Precisely it will

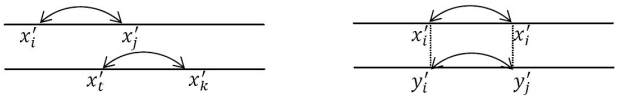

Figure 2. Two single swaps(left) and a couple-swap(right)

examine $O(pq)$ possible swaps, and each one will take $O(pq)$ for recalculation of ordinary alignment. However, we will avoid recalculation of ordinary alignment of two sequences after each swap of characters and consider different types of character swapping to obtain a much better algorithm in next section.

III. Fast Swapping Algorithm

A. Cost estimating after swapping

Let (X', Y') be the ordinary pairwise alignment of (X, Y). We now look for characters swapping in (X', Y') rather than in (X, Y) like previous algorithms. Because (X', Y') contain gap characters '-', we now allow swapping between gap characters and characters in X and Y. Swapping between gaps characters can be ignored because it has no effect on sequences. We come up with the first observation that if we swap two characters in X' or Y' the optimal ordinary pairwise alignment of new sequences are not much different from old ones'. For example, if we swap y'_i and y'_j in Y' the new optimal ordinary alignment cost can be well approximated by:

$$\begin{aligned}\text{NewCost} \approx \text{OldCost} &- [C(x'_i, y'_i) + C(x'_j, y'_j)] \\ &+ [C(x'_i, y'_j) + C(x'_j, y'_i)] \end{aligned} \quad (3)$$

In fact, the cost of ordinary alignment of new sequences after performing a swap is smaller or equal the cost in the equation (3). Moreover, the rearrangement cost can be updated easily in $O(1)$ as the swapping characters may affects only adjacent characters. Here, to simplify the time complexity estimation, we calculate rearrangement cost as the breakpoint distance. Hence, the total time required for estimating the new total cost after a swapping is now $O(1)$ that is much faster than $O(pq)$ in previous algorithms.

B. Swapping Characters

We consider two kinds of swap: single swap and couple-swap. The first type of swap called Single swaps exchange positions of two characters in X' or in Y'. A swapping of x'_i and x'_j in X' will be denoted by $S_{X'}(i, j)$. The second type of swap is couple-swaps that exchange simultaneously characters in both sequences. A coupe-swap $CS(i, j)$ exchanges x'_i with x'_j and y'_i with y'_j. In other word, we can obtain same result of a couple-swap $CS(i, j)$ with two consecutive single swaps: $S_{X'}(i, j)$ in X' and $S_{Y'}(i, j)$in Y'. A swap is said to be *good* if it makes estimated total cost decreases that implies the decreasing of actual cost. Note that if we consider only single-swaps, we may skip good characters swapping. It is possible that both $S_{X'}(i, j)$ and $S_{Y'}(i, j)$ lead to worse (higher cost) alignment but combination of them, $CS(i, j)$, leads to a much better one. Single swaps and couple-swaps have very different effects in improving quality of an alignment. Single swaps usually produce alignment with better edit/indel cost while couple-swaps often reduce rearrangement cost of the alignment. Every couple-swap will decrease or at least remain the ordinary alignment cost.

Now we present the fast swapping method in the Algorithm 1. The main idea is that we perform 'good' swaps on alignment ordinary alignment (X', Y') of (X, Y). In addition, we update the optimal alignment using dynamic algorithm only after each iteration. The algorithm stops when no swaps are performed. Since the time complexity

Algorithm 1 Fast Swapping Algorithm

```
1:  Initialize (X'_r, Y'_r) ← ordinary alignment of (X, Y)
2:  while true do
3:    for i = 1 to |X'_r| do
4:      for j = 1 to |Y'_r| do
5:        if ∃ good swapping S ∈ { S_{X'_r}(i,j), S_{Y'_r}(i,j), CS_{X_r Y_r}(i,j) } then
6:          Perform swapping S;
7:          Evaluate cost of new sequences;
8:        end if
9:      end for
10:   end for
11:   (X_r, Y_r) ← gapless sequences of (X'_r, Y'_r);
12:   (X'_r, Y'_r) ← ordinary alignment of (X_r, Y_r);
13:   if No Swaps Performed then
14:     Return (X'_r, Y'_r) and terminate;
15:   end if
16: end while
```

to perform a swap and evaluate new cost in line 6 and 7 is $O(1)$ and the time for scanning and performing possible swaps from lines 3 to line 10 is $O(n^2)$ where $n = p + q$. Lines 11 to 12 take $O(n^2)$ for finding the optimal ordinary alignment. Therefore, each loop from line 3 to line 15 can be completed in $O(n^2)$. In summary, the time complexity of the fast swapping algorithm is $O(n^2 \times \text{iteration})$ where iteration is the number of loops. In practice, the number of iteration is bounded by a constant and running time of the fast swapping algorithm is quadratic as shown in the experiment section.

IV. Memetic Algorithm

Memetic algorithms are the combinations of Evolutionary algorithms with local search. The hybridization takes the advantages of both evolutionary adaptation of a population and individual learning in lifetimes of its members. By combining global and local search, Memetic Algorithms have been shown to be orders of magnitude faster and more accurate than EAs on many problems. The general scheme for a memetic algorithm is shown in Figure 3

All individuals in the population represent local optima, which is ensured by applying local search after generation

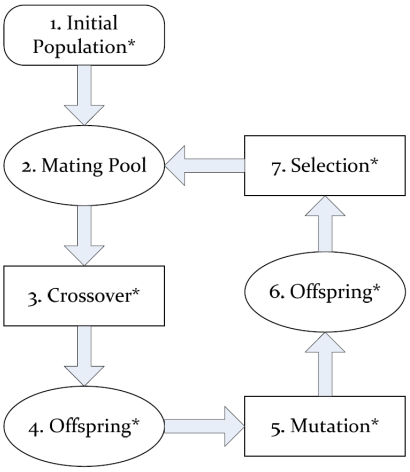

Figure 3. Genetic algorithm. Parts marked with (*) are subjected to changes in a Memetic approach.

and after application of the evolutionary operators. The algorithm is presented in Algorithm 2.

Algorithm 2 Memetic Algorithm

1: Set of population $P \leftarrow \phi$
2: Return the best individual in P
3: **for** $i = 1$ to $popSize$ do **do**
4: $p \leftarrow$ generatePAR()
5: $p \leftarrow$ FastSwapping(p)
6: $P \leftarrow P \cup \{p\}$
7: **end for**
8: **while** not converged **do**
9: Generate set of offsprint C using Crossover
10: **for** $i_c \in C$ **do**
11: $i_c \leftarrow$ FastSwapping(i_c)
12: $P \leftarrow P \cup \{i_c\}$
13: **end for**
14: Apply mutation operator on P
15: $P \leftarrow$ Selection(P)
16: **end while**

A. Initialization

The function generatePAR() in line 2 returns a randomly initiated PAR of (X, Y). Both X and Y are shuffled and their ordinary alignment are found using dynamic programming algorithm.

B. Crossover

Assume that we need to combine a 'male' PAR $(X^m = (x_1^m, x_2^m, \ldots, x_n^m), Y^m = (y_1^m, y_2^m, \ldots, y_n^m))$ with a 'female' PAR $(X^f = (x_1^f, x_2^f, \ldots, x_n^f), Y^f = (y_1^f, y_2^f, \ldots, y_n^f))$. First we set $(X^c, Y^c) \leftarrow (X^m, Y^m)$ and perform Couple-swaps on (X^c, Y^c) such that after swapping the gapless sequences of X^c equals to the gapless sequences of X^f. By doing so, the child will inherit the order of the sequence X^f in the 'female' PAR and have the edit cost at most the edit cost of the 'male' PAR. We create an another child in a similar way with 'male' PAR in place of 'female' PAR and vice versa.

C. Mutation

The mutating is performed by applying some random single-swaps on the PAR. The number of swaps is large enough to guarantee that the PAR will escape from the current local optimum.

V. Experimental Analysis

A. Data Sets

As no local search algorithm has a polynomial guarantee in the worst-case, extensive testing is required. We use two main sources of data. One is a collection of 760 Metazoa4 mtDNA genomes from Genbank at NCBI, which was used, in the previous research by Vinh et al. [21]. The second is collection of various size genomes artificially generated through a simulated evolving process. Metazoa mtDNA genomes are usually found in form of closed circles. They undergo not only nucleotide transformations but also gene rearrangements [5]. Each genome contains about 16,000 base pairs and about same 37 genes [5].

Genomes are divided into segments6based on annotations from Genbank at NCBI and each segment is now considered as a character. After segmenting, a genome has 50 characters on average. We use the same testing method as in [21] for the first data source: 760 genomes are divided randomly into 38 groups of size 20. The PARs between any two genomes in the same group are constructed. Thus, the number of evaluated pairs is 7220. The results presented are averages taken over all the instances in the data source.

To create artificial genomes, at the first step we generate randomly genomes. Then, the evolving process simulates both small-scale and large-scale operations on genomes to produce final pairs. The mechanism is quite simple; however, results are quite consistent between two instance sources in term of both running time and quality of PAR.

B. Optimality Measurement

To evaluate the optimality of PAR returned by the algorithm, we compare the cost of solution with an lower bound of the optimal solution. Recall that the cost of the optimal solution

$$\begin{aligned} OPT &= \mathcal{C}(A_r^*(X_r^*, Y_r^*)) \\ &= \min_{X_r', Y_r'} \{\ C(X_r', Y_r') + R(X, X_r) + R(Y, Y_r)\ \} \\ &< \min\{\ C(X_r', Y_r')\ \} \end{aligned} \quad (4)$$

For every PAR (X_r', Y_r') of (X, Y) we can make (X_r', Y_r') have length $n = p + q$ by adding/removing pairs of gap characters '-' into it. Because there are exactly $n = p + q$ non-gap characters in the PAR. Therefore by introducing two new sequences of size $n = p + q$

$$\begin{aligned} X^+ &= (x_1^+, x_2^+, \ldots, x_p^+, \text{“}-\text{”}, \text{“}-\text{”}, \ldots, \text{“}-\text{”}) \\ Y^+ &= (y_1^+, y_2^+, \ldots, y_q^+, \text{“}-\text{”}, \text{“}-\text{”}, \ldots, \text{“}-\text{”}) \end{aligned}$$

We have:

$$\begin{aligned} OPT &< \min\{C(X'_r, Y'_r) \mid (X'_r, Y'_r) \in \mathcal{PAR}_{(X,Y)}, \\ &\quad |X'_r| = |Y'_r| = n\} \\ &= \min\{\sum_{i=1}^{n} C(x_i^+, y_{\pi(i)}^+) \\ &\quad \mid \pi \text{ is a permutation of } \{1, 2, \ldots, n\} \} \\ &= M(X, Y) \end{aligned} \quad (5)$$

It is easy to see that (5) is a linear sum assignment problem(LSAP) [16] or minimum cost perfect matching in bipartite graph. Hence the lower bound $M(X, Y)$ can be computed in $O(n^3)$ with Hungarian algorithm or Kuhn-Munkres algorithm [13]

The optimality of a PAR (X'_r, Y'_r) with cost C_r is now can be measured by:

$$\frac{M(X, Y)}{C_r} \times 100\% \quad (6)$$

The transformation cost between two characters x and y is calculated as the minimum cost to change x into y with nucleotide substitutions, insertions, and deletions. The nucleotide substitution cost and the 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.

The algorithm to calculate inversion distance in $O(n)$ is rather complex [2]. However, it is known from previous work of Vinh et al. [21] that when trying different rearrangement functions, both inversion distance and breakpoint distance show similar results. Thus, we measure rearrangement cost $R(X, X_r)$ as breakpoint distance between X and X_r.

The memetic algorithm was implemented using the GALib library, a C++ library of genetic algorithm components written by Matthew Wall. This 7 year old library is now showing its maturity, as the code is elegant object oriented, which allows us to easily customize it and combine it with the fast swapping method. The library can be found at the author web site http://lancet.mit.edu/ga/.

We use Simple GA with typical parameters: $popSize = 100$, $p_{crossover} = 0.9$, $p_{mutation} = 0.006$ to run the algorithm through 100 generations. The result for MA is very promising; however, we have to pay a high price for the running time. On metazoa genomes, it takes up to 5 minutes to align a pair of average size genomes.

Results were taken average on 7220 pairs of metazoa mtDNA genomes. The result for MA is very promising; however, we have to pay a high price for the running time. On metazoa genomes, it takes up to 5 minutes to align a pair of average size genomes

The average results when testing on Metazoa mtDNA is shown in table I. Both algorithms SCS and fast swapping obtain high optimality ratio and fast swapping has slightly better optimality ratio. However, fast swapping is a big improvement in term of running time comparing to SCS. Fast swapping algorithm runs about 30 times faster than SCS and it needs on average only 3 iterations to align genome sequences. The number of iteration in case of SCS is 5 [21]. The second source data contains larger genomic sequences. The sizes of genomes pairs were chosen from powers of 10. As SCS takes too long time when running on large genomes, results for genome sequences with size larger than $10^{2.5}$ is not available. The quadratic running time of fast swapping enable it to align very large genome sequences. Note that each character in generated genomes is a sequence of 500 to 1000 nucleotides. Hence, genomes with millions of nucleotides can be aligned in minutes on a personal computer. All the experiments in this section are performed on a normal Desktop 4 PC.

TABLE I.
AVERAGE RESULTS ON METAZOA MTDNA

	Ave. Optimality	**Ave. Running Time**
SCS	99.39 %	0.46 s
FS	99.41 %	0.01 s
MA	99.79 %	5 m

TABLE II.
AVERAGE RUNNING TIME ON ARTIFICIALLY GENERATED GENOMES

Genome size	10^2	$10^{2.5}$	10^3	$10^{3.5}$
SCS	3.13 s	3 m 29 s	n/a	n/a
FS	0.03 s	0.31 s	2.02 s	27.27 s

Fast swapping also gives better optimality ration when aligning larger genome sequences as shown in table III.

VI. CONCLUSION

The Fast Swapping algorithm gives out better PARs than the best-known algorithm Simultaneous Swapping. Moreover, it is much faster with empirical running time as low as the size of the input. SCS is only suitable for genome sequence as large as 100 characters whereas fast swapping can be applied for multi-thousands characters genomes that contain millions of base pairs. However, both algorithms are local search methods which depend heavily on initial alignment configuration. Lightweight fast swapping algorithm can be also combined in global search method such as Simulated Annealing, Genetic Algorithm, Ant Colony and so on to obtain even better optimality ratio.

The proposed algorithms were tested only with breakpoint distance as rearrangement cost function. However, algorithm can be extended easily to work with any rearrangement cost functions. Then, required time to perform

TABLE III.
AVERAGE OPTIMALITY ON ARTIFICIALLY GENERATED GENOMES

Genome size	10^2	$10^{2.5}$	10^3	$10^{3.5}$
SCS	98.59 %	98.46 %	n/a	n/a
FS	99.07 %	99.56 %	99.82 %	99.95 %

a move in the fast swapping algorithm will mainly depend on the time to update the new rearrangement cost.

REFERENCES

[1] Albertson DG, Collins C, McCormick F, Gray JW: Chromosome aberrations in solid tumors. *Nat. Genet.* 2003, 34(4):369-376.

[2] Bader, D.A, Moret, Bernard M.E and Yan, Mi. A Linear-Time Algorithm for Computing Inversion Distance Between Signed Permutations with an Experimental Study. *Journal of Computational Biology*, 2001, Vol. 8, pp. 483-491.

[3] Batzoglou, Serafim. The many faces of sequence alignment. Briefings in *Bioinformatics Oxford Journal*, 2005, Vol. 6.

[4] Batzoglou, S., Pachter, L., Mesirov, J, P., Berger, B., and Lander, E, S. (2000). Human and mouse gene structure: comparative analysis and application to exon prediction. *Genome Res*, 10(7):9508.

[5] Boore, J. L. Animal mitochondrial genomes. s.l. : *Oxford University Express*, 1999, Vol. 27. 1767-1780

[6] 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.

[7] Couronne, O., Poliakov, A., Bray, N., Ishkhanov, T., Ryaboy, D., Rubin, E.M., Pachter, L., and Dubchak, I. 2003. Strategies and tools for whole genome alignments. *Genome Res*. 13: 73-80.

[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., Fleischmann, R, D., Peterson, J., White, O., and Salzberg, S, L. (1999). Alignment of whole genomes. *Nucleic Acids Res*, 27(11):236976

[10] Eichler EE, Sankoff D: Structural dynamics of eukaryotic chromosome evolution. *Science* 2003, 301(5634):793-797.

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

[12] Kent,W, J. and Zahler, A,M. (2000). Conservation, regulation, synteny, and introns in a large-scale C. briggsae-C. elegans genomic alignment. *Genome Res*, 10(8):111525.

[13] Kuhn, H. W. The Hungarian method for the assignment problem. *Naval Research Logistic Quarterly*, 1955, Vol. 2, pp. 83-97

[14] Ma, B., Tromp, J., and Li, M. 2002. PatternHunter: Faster and more sensitive homology search. *Bioinformatics* 18: 440-445.

[15] Needleman, S. B. and Wunsch, C. D. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 1970, Vol. 48, pp. 443-453

[16] Rainer Burkard, Mauro Dell'Amico, Silvano Martello. Assignment Problems. s.l. : *SIAM Monographs on Discrete Mathematics and Applications*. pp. 75-166

[17] Rieseberg L: Chromosomal rearrangements and speciation. *Trends Ecol. Evol.* 2001, 16(7):351-358.

[18] Sanko, D. and Blanchette. Multiple genome rearrangement and breakpoint phylogeny. *Journal of Comput. Biol.*, 1998, Vol. 5, pp. 555-570

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

[20] Stankiewicz P, Lupski JR: Genome architecture, rearrangements and genomic disorders. *Trends Genet* 2002, 18(2):74-82.

[21] Vinh L.S, Andres Varon and Ward Wheeler, Pairwise alignment with rearrangements, *Genome Informatics* 17(2):141-151

[22] Wang, LiSan, et al. Distance-based genome rearrangement phylogeny. 4, *Journal of Molecular Evolution*, 2006, Vol. 63, pp. 473-83

[23] Waterston, R.H., Lindblad-Toh, K., Birney, E., Rogers, J., Abril, J.F., Agarwal, P., Agarwala, R., Ainscough, R., Alexandersson, M., An, P., et al. 2002. Initial sequencing and comparative analysis of the mouse genome. *Nature* 420: 520-562.