International Journal of Basic and Applied Biology
Print ISSN: 2349 – 5820; Online ISSN: 2349 – 5839; Volume. 2 No. 2, November 2014. pp. 23 – 27
© Krishi Sanskriti Publications
http://www.krishisanskriti.org/ijbab.html

An Improved Algorithm for Reducing the Solution Space of Sorting by Reversals

Beethika Tripathi[1], **Amritanjali**[2]
Department of Computer Science and Engineering
Birla Institute of Technology
Mesra, Ranchi, Jharkhand, India 835215.

Abstract: **The goal of this paper is to give an efficient algorithm for sorting the genome with respect to other genome using reversal operation and to generate the optimum traces. Usually the number of solutions is large so solution space can be represented in compact manner by grouping equivalent solutions into traces. To reduce the memory requirement depth first approach is employed. Still there are some traces having reversals that are biologically not possible so it is of no use to further study them so such traces are discarded by applying some biological constraints at each step of the process for generating traces. This approach not only reduces memory and time consumption but also makes it easier to parallelize and a better evolutionary scenario of genomes is characterized. This allows us to efficiently handle large permutations so that the exploration time is visibly reduced.**

1. INTRODUCTION

Genetic rearrangement over a period of time leads to evolution. Reversal is one of the most commonly observed evolutionary events [1]. In this a segment of a chromosome gets flipped, thereby reversing the order and orientation of the genes in that segment. By comparing the shared gene order in two genomes that have evolved by reversals events, their evolutionary scenario is reconstructed.

Sorting by reversals is defined as the problem of finding the minimal sequence of reversals that can transform the shared gene order of one of the given genome into that of another. Till now lots of study has been done on finding the reversal distance (minimum number of reversals) between two given genomes and generating corresponding sequence of reversals.

Genomes are represented as permutations where duplicate genes are not allowed. When the permutation is unsigned the sorting by reversal problem is NP-hard [2]. In 1995 Hannenhalli and Pevzner [3] for the first time gave a proof of solving it in polynomial time for the signed versions of the problem. Since then, many solutions were given with speed and memory utilization improvements. Later Siepel and Ajana et al. [5-6] proposed an algorithm that listed all sorting reversals (ASR) in $O(n^3)$ time complexity. By repetitively applying their algorithms, Siepel generated All Sorting Sequences by Reversals (ASSR) that transforms one genome into another. Later an improved version of their algorithm was given whose average-case complexity is $O(n^2)$ [7]. Bergeron et al. [8] grouped those sequences into equivalence classes and gave the concept of traces [9] to represent them into normal form thereby reducing the solution set. However, no algorithm was given. Braga et al. [10-11], developed an algorithm by combining the results of Siepel and of Bergeron et al. that enumerates the normal form of every trace and provides the count of the number of sorting sequences. Baudet et al. [12] used the approach of exploring the solution space in depth first manner using stack and listed normal forms of traces to represent classes of sorting sequences this drastically reduced the memory usage. But their algorithm could not find the total number of solutions. So Amritanjali and Sahoo [13] proposed a modified approach to count the total number of possible solutions while generating the traces are generated in depth first manner.

However, for most of the genomes there can several sequences and the solution space increases exponentially with reversal distance and number of shared genes. Each sequence describes a probable to actual scenario can be deduced by applying some biological constraints. Very large number of possible sequences makes their analysis difficult and listing them requires lots of memory and time. Application of biological constraints like common intervals [15] can help to discard reversals that are less probable, reducing the solution space. In the proposed algorithm, we are applying this constraint to generate solutions that are preserving the conserved segments. Also, the solutions are generated in depth first manner and they are grouped into equivalence classes for compact representation. Reduction of solution space not only decreases computation time but also makes their analysis easier. Moreover, depth first approach requires less memory and is easier to parallelize.

2. BACKGROUND

2.1. Basics

Unichromosomal genomes without duplication of genes are considered in our study. Genomes are represented by the list of homologous markers (usually genes or blocks of contiguous genes) between them. Signed integers are used to represent the markers where sign denotes the orientation of markers on one genome with respect to the other genome. The signed permutation $\pi = (\pi_1 \,.\,.\, \pi_n)$ of size n is used for representation where it can take values from –n to n without repetition of absolute value of integer. Here π_i represents the element at position i in π. π_T represents the identity permutation (1. . n).

A reversal ρ on an interval [i,j] of a permutation π where $1 \le i \le j \le n$, is defined as $\rho = \{|\pi_i|, |\pi_{i+1}|, . . . , |\pi_{j-1}|, |\pi_j |\}$,in sorted order. The reversal operation is denoted by $\pi \circ \rho$ where it reverses the order and flips the signs of the elements of ρ. The operation is represented as:

$$\pi \circ \rho = (\pi_1\ \pi_2 \ldots\ \pi_{i-1} -\pi_j -\pi_{j-1} \ldots -\pi_{i+1} -\pi_i\ \pi_{j+1}\ \ldots\ \pi_{n-1}\ \pi_n)$$

For example π={1 4 -3 2 -5}, ρ={2,3,4} then $\pi \circ \rho$={1 -2 3 -4 -5}.

A sequence or i−sequence of reversals is represented as $\rho_1\rho_2...\rho_i$ is valid sequence for a permutation π, then $\pi \circ \rho_1,\rho_2,...\rho_i$ denotes the successive application of the reversals $\rho_1, \rho_2,... \rho_i$ in the order in which they appear. An i−sequence of reversals $\rho_1\rho_2...\rho_i$ sorts a permutation π into a permutation π′ if $\pi \circ \rho_1\rho_2...\rho_i = \pi'$.

The length of the shortest sequence of reversals that sorts a permutation π into π_T is called the reversal distance and is denoted by d(π) which is calculated from the breakpoint graph[14].

An optimal i-sequence is represented as $s = \rho_1\rho_2...\rho_i$ (a valid i-sequence of reversals for π), if $d(\pi \circ s) = d(\pi)$-i. s is called an optimal sorting sequence for π and π_T when i = d(π). For example s = {1}{2}{1,2,3} is an optimal 3 sequence for π={-3 2 1 -4}.

Siepel gave an algorithm [5] to list all optimal 1-sequences in $O(n^3)$ called All Sorting Reversals. Using his algorithm those reversals are computed that will bring the given permutation one step closer to the target. All such 1-sequences are listed. Then each of them are applied to the original permutation to form the new set permutations and each of those permutations have reversal distance d(π)-1. Now new set of reversals are computed for each permutation. This 1-sequence when combined with the predecessor ρ optimal 2-sequence are generated. In this way the algorithm is repeated till d(π)-sequences are generated reducing the distance to zero and each of them sort permutation π to $\pi_{T.}$ For example {1}{2}{1,2,3}{4} is one such solution that sorts π=(-3 2 1 -4) to π_T=(1 2 3 4).

The solutions generated are huge in number and keeps on increasing as the size of permutation and the reversal distance increases. For example the permutation (−4 −11 6 −9 −2 1 −8 3 −10 7 −5) has 6345019 solutions. Though all the solutions could be easily obtained by Siepel's algorithm but it requires lot of time as well as memory to compute and store them.

The concept of traces was given by Bergeron *et al.* by grouping the similar sequences into equivalence class and representing them using normal form of traces. Two sequences are said to be equivalent if one can be obtained from another by sequence of commutations of non-overlapping reversals. For example, the sequences of reversals (words) {2} { 2, 3 ,4} { 3, 4, 5} and { 2, 3, 4} {2} {3, 4, 5} are equivalent because the reversals {2} and {2, 3 ,4} commute. As opposed to it none of these sequences of reversals are equivalent to {2} {3, 4, 5} {2, 3, 4} because the reversals {2,3,4} and {3,4,5} overlap.

An equivalence class of optimal sequences of reversals under this equivalence relation is called trace. For any trace there is a unique representation called the normal form. This is done by finding out the commuting pairs, these are the non-overlapping pairs or one set completely contained in another, their positions can be interchanged and belong to same equivalence class these when arranged in lexicographic order forms 1 sub-word of the normal form of trace. The other overlapping pairs are considered as another sub-word. All such sub-words are found out, these sub-words when arranged in lexicographic order forms the normal form of trace representing the equivalence class. For example for the permutation π={1 4 -3 2 -5} there are two normal form traces possible {2}{2,3,4}{4}{5} and {2,3,5}{3} > {2,4,5} > {3,4,5}. In first trace all {2}, {2,3,4}, {4} and {5} are commuting so position of any of them could be interchanged to give 24 different solutions. In the second trace {2,3,5},{2,4,5} and {3,4,5} overlap with each other resulting in three sub-words. Only reversal {3} commutes with all other reversals, so by interchanging {3} with others we get 4 solutions for this trace. In short total 28 solutions are represented just by two traces greatly reducing the solution set.

Braga *et al.* combined the algorithm of Siepel and with the concept of traces [10]. By using the normal form representation, it is capable of enumerating the set of all traces that represents all possible solutions.

2.2. Biological Constraints

While dealing with the Sorting by Reversal problem the solution set keeps on increasing as the size of the permutation or reversal distance increases. These large solution sets are difficult to be handled and have to be logically reduced to be of any practical importance. So various biological constraints could be applied on the reversals to logically limit the reversals that are practically not possible or whose probability of occurrence is very less. Common Interval is one such phenomenon in which the clusters of co-localised genes between the two genomes are listed. This common interval is said to have been inherited from the common ancestor and the chances of occurrence of sequence that breaks this block is very less. So we could discard solutions having such reversals, resulting into a much reduced solution set and biologically more feasible.

A pair interval ($[x_\pi ,y_\pi]$, $[x_{\pi T} ,y_{\pi T}]$) is called a common interval between the permutation π and π_T if it satisfies the following condition:

$$\{ \pi_i \,|\, i \in [x_\pi ,y_\pi]\} = \{ \pi_{Ti} \,|\, i \in [x_{\pi T} ,y_{\pi T}]\}$$

If i^th element of a permutation π is j i.e. $\pi_i = j$ then $\pi^{-1}j = i$ means that element j is present at i^th index of π. $I_{\pi\pi T}$ is denoted as $\pi_T^{-1}\,\pi$ (i.e. $I_{\pi\pi T}(i)= \pi_T^{-1}(\pi(i))$. So $I_{\pi\pi T}(i)=j$ means i^th element of π is located at j^th position of $\pi_{T.}$

For interval [x,y] of π:

$$l(x, y) = \min I_{\pi\pi T}(i) \text{ where } i \in [x ,y]$$

$$u(x,y) = \max I_{\pi\pi T}(i) \text{ where } i \in [x,y]$$

$$f(x, y) = u(x , y) - l(x , y) - (y - x)$$

When $f(x , y) = 0$ then ($[x_\pi ,y_\pi]$, $[x_{\pi T} ,y_{\pi T}]$) represents the common interval. An efficient algorithm to enumerate all common intervals between two permutations has been proposed in [16].

For example the set of common interval between the permutations π = {-7 4 5 6 -3 8 -2 -1} and π_T = {1...8} are I={{1,2},{2..8},{3…6},{3…8},{4,5},{4…7} {5,6}}

2.3. Filter Process

The filter process is done as soon as the 1-sequences are generated. Each of the reversals is checked if they break the common interval or not. A common interval breaks when one end of reversal lie inside the interval and other end outside the interval and the interval is not completely contained within the reversal, such that after performing the reversal the elements of the common interval are not contiguous. Those reversals that break the interval are not added to the set of 1-sequences. This filtration process is done at each step of the algorithm as soon as the 1-sequences are generated. So the process keeps on reducing the solution set generating the solutions that define most probable evolutionary scenario.

Let θ represent one of the intervals amongst the set of common intervals. A reversal ρ breaks an interval θ if ρ overlaps with θ and θ is not completely contained in ρ. Considering, for instance, the permutation (−5 −2 −7 4 −8 3 6 −1) having θ = {2,…,8} we observe that the reversal ρ = {1, 3, 4, 6, 7, 8} breaks the interval.

The detection of common intervals is done at the beginning of the analysis. An optimal sequence of reversals sorting a permutation π into π_T that does not break any common interval initially detected between π and π_T is called a perfect sorting sequence.

3. PROPOSED WORK

In order to generate (i+1)-trace it is not necessary to calculate all i-trace [12]. From here the concept of depth first approach came into existence, when the tree was processed in depth first manner instead of breadth first manner, this improved the memory usage.

We are using tree structure to store the 1-sequenes where each node is composed of its elements containing a reversal ρ and the number of solutions generated following the reversals in the path from the root node to the current node represented as c. All the 1-sequences are generated and they are checked for common interval preserving property so that those which do not preserve the property are filtered. We store all these filtered sequences in lexicographic manner in the root node of the tree and c for each element is initialized as 0. Now take the first reversal apply it on the given permutation π and obtain new permutation π' again compute 1-sequences for it filter it and store them in lexicographic order in the next node. Similarly take the first reversal from it and process repeat this till the height of the tree becomes equal to $d(\pi)$ then take all the 1-sequence in the given order and it's the new trace.

While storing the 1-sequence in the node of the tree it should be done in the manner that it preserves the normal form order with respect to the sub-trace generated so for, this is done to ensure that the first trace generated is in normal form. Once the branch is explored perform backtracking to explore the new branch. Repeat this till all the branches have been explored.

1. Generate all the 1-sequence of reversals preserving the CI and store in lexicographic order in root node.
2. For each reversal
   a) Find the number of solutions generated by the reversal by calling recursive procedure EXPAND.
   b) Add the number of solutions of each element to get the total number of solutions.

Figure 1. TRACE_GENERATION procedure.

The routine TRACE_GENERATION initializes the root node with the filtered 1-sequences in the lexicographic order. It then calls the recursive procedure EXPAND to explore the path originating from here and returns the number of possible solutions it could have with that particular element as the first element of the trace. Adding them gives the size of the solution set.

1. Apply reversal on given genome to get new genome sequence.
2. If level is equal to d(π) print the trace and return number of solutions as 1.
3. Generate next 1-sequence of reversals preserving the CI and store in normal form order with respect to the predecessor sequence in new node at next level.
4. For each 1-sequence in node
   a) If appending 1-sequence to the predecessor sequence leads to a new trace then call EXPAND to find number of solutions otherwise find the number of solution from the traces generated so far.
   b) Add number of solutions to total number of solutions for the node.
5. Return total number of solutions.

Figure 2. EXPAND procedure.

A sequence is appended to the predecessor sequence in a particular order i.e. first all the commuting reversals gets arranged in lexicographic order then the non-commuting reversals in lexicographic order. Therefore while appending the 1-sequence to the predecessor sequence if it gets appended at the end then it might lead to a new trace so we could expand further otherwise if it gets appended somewhere in the middle it means that the sequence had already been processed and no need to expand further. In the latter case just determine the number of solution it could have by following the path that has been processed.

Since many of the reversals are redundant so identifying them at an earlier stage and terminating the path there reduces the amount of computation and memory consumption. After exploring each branch, the normal form of the trace found along with the count of each element is stored separately and the memory is freed before exploring next branch.

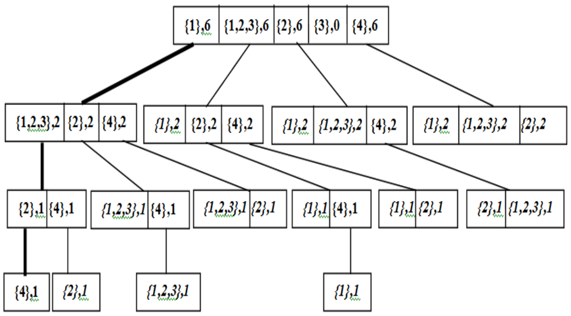

Figure 3. Tree structure of the traces that sort the permutation (−3 2 1 −4).

As in Figure 3 which represents the tree structure for the permutation {-3 2 1 -4}. In this representation, the values in plain text are reversals. The values in italics are reversals which are optimal 1-sequences but, when combined with predecessor trace, lead to traces that were inserted in another branch of the tree, which means they are redundant. The broader edge represents the trace {1}{1,2,3}{2}{4} that sort the permutation.

4. DISCUSSIONS

We reduced the universe of sequences and class by applying the biological constraints and filtering those sequences which are biologically less probable to occur. This could help the biologists to work on large permutations or permutations with large reversal distance, and a better evolutionary scenario is characterized for the two genomes. The solution space is explored in depth first manner to list the normal form of possible traces while providing the count of the total number of solutions in the solution space.

This version could further be extended to form a parallel version of the algorithm. Data Parallelism can be employed where each branch is handled by a different processor and all the processors are doing the same work. Each branch is explored separately by different processors independently starting from each reversal in the optimal 1-sequences of reversals of the input permutation. This results in better time and space complexity.

Other different types of constraints could be applied depending upon the type of application. The concept of near-perfect trace could be used which allows bounded number of breaks per trace. This is done no perfect sorting sequence exist so a near-perfect sorting sequence is generated.

The use of biological constraints has some important limitations. First, there is no guarantee that a sequence that respects the given constraints exists, thus this approach may lead to empty results, which is undesirable. Relaxing the biological constraints in order to obtain a non-empty result is generally possible, but this approach may require some work for relaxing the parameters, which costs computation time.

5. REFERENCES

[1] McLysaght, A., Seoighe, C. and Wolfe K. H.: “High frequ-ency of inversions during eukaryote gene order evolution”, in: *Comparative Genomics: Empirical and Analytical Approaches to Gene Order Dynamics, Map Alignment and the Evolution of Gene Families,* D. Sankoff and J. H. Nadeau (Eds.), 2000, pp. 47–55.

[2] Caprara, A.: "Sorting by reversals is difficult", in: *Proc. 1st Ann. Int’l Conf. Comput. Mol. Biol. (RECOMB 1997),* ACM Press, New York (1997), pp. 75–83.

[3] Hannenhalli, S., Pevzner, P.A.: "Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals)", in: *Proc. 27th Ann. ACM Symp. Theory of Comput. (STOC 1995),* ACM Press, New York (1995), pp. 178–189.

[4] Tannier, E., Bergeron, A., Sagot, M.-F.: "Advances on sorting by reversals.", Disc. Appl. Math. (2007), 155(6-7), pp. 881–888.

[5] Ajana, Y., Lefebvre, J.-F., Tillier, E.R.M., El-Mabrouk, N.: "Exploring the set of all minimal sequences of reversals - an application to test the replication-directed reversal hypothesis.", in: *Guig´o, R., Gusfield, D. (eds.) WABI 2002. LNCS, vol. 2452,* Springer, Heidelberg (2002), pp. 300–315.

[6] Siepel, A.C.: "An algorithm to find all sorting reversals.", in: *Proc. 6th Ann. Int’l Conf. Comput. Mol. Biol. (RECOMB 2002),* ACM Press, New York (2002), pp. 281–290.

[7] Swenson, K.M., Badr, G., Sankoff, D.: "Listing all sorting reversals in quadratic time.", in: *Singh, M. (ed.) WABI 2010. LNCS, vol. 6293,* Springer, Heidelberg (2010), pp. 102–110.

[8] Bergeron, A., Chauve, C., Hartman, T., Saint-Onge, K.: "On the properties of sequences of reversals that sort a signed permutation.", i*n: JOBIM,*(June 2002), pp. 99–108

[9] Diekert, V., Rozenberg, G.: "The Book of Traces",. World Scientific, Singapore (1995)

[10] Braga, M.D.V.: Baobabluna: "The solution space of sorting by reversals.", Bioinformatics 25(14) (2009).

[11] Braga, M.D.V., Sagot, M., Scornavacca, C., Tannier, E.: "The solution space of sorting by reversals", in: *M˘andoiu, I.I., Zelikovsky, A. (eds.) ISBRA 2007. LNCS (LNBI), vol. 4463,* Springer, Heidelberg (2007), pp. 293–304.

[12] Baudet, C., Dias, Z.: "An improved algorithm to enumerate all traces that sort a signed permutation by reversals", in: Proc. *The 2010 ACM Symposium on Applied Computing*, 2010, pp. 1521–1525

[13] Amritanjali , G. Sahoo: "Exploring the Solution Space of Sorting By Reversals: A New Approach", in: *International Journal of Information Technology, Issue. 2, Vol.3* (June 2013), pp. 98–104.

[14] Bergeron A., “A very elementary presentation of the Hannenhalli-Pevzner theory”, Discrete Applied Mathematics, vol. 146, 2005, pp. 134–145.

[15] Braga, M.D.V., Gautier, C., Sagot, M.: "An asymmetric approach to preserve common intervals while sorting by reversals", Algorithms for Molecular Biology 4(16), (2009).

[16] T. Uno, M. Yagiura,: "Fast algorithms to enumerate All Common Intervals of two Permutations", Algorithmica 26(2), 2000, pp. 290-309.