Finding Sorting Traces of Reversals in the Presence of Hurdles

Beethika Tripathi
Department of Computer Science and Engineering, Birla Institute of Technology, Mesra, Ranchi, Jharkhand, India

Amritanjali
Department of Computer Science and Engineering, Birla Institute of Technology, Mesra, Ranchi, Jharkhand, India

ABSTRACT: Reversal is one of the most commonly observed evolutionary events that affect the order of genes in a single chromosome. By comparing the shared genes order in two genomes that have evolved by reversals events, their evolutionary scenario is reconstructed. The problem of finding the most plausible sequence of reversals that changed the genomes is complicated by the fact that for most of the cases several solutions are generated using the computational methods. Lot of work has been done to reduce the time and space complexities of these approaches. Another issue in the implementation is the problem of hurdles. We modify the existing approaches to enumerate the possible solutions of the sorting by reversals problem in the presence of hurdles and compare their performance.

1 INTRODUCTION

Evolutionary scenario between two species is described by the sequence of rearrangements that changed the arrangement of the shared markers (genes or segment of genes) in their genomes. From parsimony hypothesis most plausible evolutionary scenario is that which involves minimal number of rearrangements. Reversal is the most common rearrangement operation (McLysaght et al., 2000) that cuts a chromosome at two points and reverses the order and orientation of the genes in the segment between the two points. For two genomes that have evolved by reversals, finding minimal sequence of reversals that can transform the sequence of shared markers in one genome into that of another is the sorting by reversals problem. Analyzing these mutations can reveal the actual scenario that might have happened during a period of time. BaobabLuna (Braga, 2009) is freely available software package that can list all possible sorting sequences of reversals. It can also group the sorting sequences into equivalence class and represent the class by a unique sequence while counting the total number of sorting sequences in each class. Baudet and Dias (2010) proposed an improved method and generated traces in depth first manner. Another method to improve the performance is to group the traces according to the permutations they generate at every step. This reduces the amount of computation as different intermediate traces resulting in same permutation are not processed separately (Badr et al., 2011). All these approaches are based on Hannenhalli and Pevzner (1995) model of breakpoint graph. However, if bad components are present in the breakpoint graph of the input signed permutation, the current implementations fails to produce solutions. Braga and Stoye (2010) used a different model employing double cut and join (DCJ) operation which is free of bad components. However, finding all possible solutions using this model is still an open problem. In this paper, we modify the existing approaches based on the breakpoint graph model to enumerate the solution space of the sorting by reversals problem in the presence of hurdles and compare their performance.

2 GETTING STARTED

2.1 *Notations*

The order of the shared genes in one genome, say target, is represented by an identity permutation π_T of integers 1 to n, where n is the number of shared genes. The relative order and orientation of these genes in the chromosome of the other genome, say source, is represented by a signed permutation of n integers, $\pi = (\pi_1 \ldots \pi_n)$. A reversal ρ on an interval $[i, j]$ of a permutation π where $1 \leq i \leq j \leq n$, is defined as $\rho = \{|\pi_i|, |\pi_{i+1}|, \ldots, |\pi_{j-1}|, |\pi_j|\}$, in sorted order. Application of reversal operation on a permutation π is represented as:

$$\pi.\rho = \left(\pi_1\pi_2...\pi_{i-1} - \pi_j - \pi_{j-1}... - \pi_{i+1} - \pi_i\pi_{j+1}...\pi_{n-1}\pi_n\right) \quad (1)$$

An $i-$sequence of reversals $\rho_1\rho_2...\rho_i$ transforms the permutation π into a permutation π' if $\pi.\rho_1\rho_2...\rho_i = \pi'$. 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.

2.2 *Enumerating Solution Space of Sorting by Reversals Problem*

The BFA method (Braga et al., 2007) enumerates the solution space of the sorting by reversals problem in breadth first manner. The set of optimal reversals that can be applied to the input permutation is called as 1-sequences of optimal reversal. On applying them to the source permutation we get new permutations that are one step closer to the target permutation. For each of these, next 1-sequences of optimal reversal are computed. When these 1-sequences are concatenated to their corresponding preceding sequence, we get 2-sequences of optimal reversals. Finally, after d-iterations all the d-sequences of optimal reversals are obtained. Each sequence is one solution in the solution space. The solution space is usually huge and contains many equivalent sorting sequences, so for compact representation those intermediate sequences are merged into same equivalence class, using the concept of traces (Diekert and Rozenberg, 1995). The baobabLuna program (Braga, 2009) outputs one trace per class along with the number of sorting sequences in each class. The process of enumerating sorting traces is similar. At any step if two i-traces are similar then they are represented by single i-trace. But a running count is maintained to keep track of number of similar traces merged, which gives the count of total number of sorting sequences in each trace generated.

In a recent paper, Baudet and Dias (2010) observed that only those i-traces are new in which the next reversal appends at the end of the previous (i-1)-trace. All the remaining i-traces can be discarded without checking for similarity with the existing traces. This saves the time spent in comparing a newly formed i-trace with all the existing i-traces. However, using this concept it is not possible to count the number of sorting sequences in each trace. To make it more memory-efficient the d-traces are generated using depth first approach (DFA) instead of breadth first approach. The initial set of 1-sequences is processed in lexicographic order. Starting with first 1-sequence of reversal, corresponding 2-sequences are generated. Continuing like this up to d-level, first d-trace is obtained, it backtracks to next reversal in the previous level. As all the reversals at each level get processed, it backtracks to the previous level. Finally, it comes back to the first level and proceeds to next reversal at that level. In this way, all the d-traces are obtained in depth first manner. Both the approaches, BFA and DFA, use breakpoint graph for calculating reversal distance and identifying sorting reversals at each step.

2.3 *Breakpoint Graph*

The breakpoint graph is constructed from the source and the target permutations. It contains two types of edges. Gray edge connects vertices corresponding to adjacent genes in target and black edge connects vertices corresponding to adjacent genes in source. Figure 1(a) describes an example breakpoint graph. The arcs are gray edges and straight lines along the axis are the black edges. A cycle is a closed path of alternate gray and black edges and component is set of overlapping cycles. Cycles are said to overlap when their gray edge(s) intersect. A trivial cycle consists of a single pair of gray and black edges. If two adjacent genes in the target have oppo-

site signs then the corresponding gray edge is called oriented. And the cycle or the component containing one or more oriented gray edge(s) is also oriented. A benign component is either oriented or trivial. Hurdle is an unoriented component that does not separate other unoriented components. If removing the hurdle transforms a non-hurdle into hurdle then it is called super hurdle otherwise simple. Such a non-hurdle is said to be protected. A fortress is present in the graph if all the hurdles are super hurdles and they are odd in number.

Cycles and their interleaving structure give clues to past rearrangement events. By Hannenhalli and Pevzener duality theorem, the reversals distance is computed as: $d = n + 1 - c + h + f$. Here, n is the size of the permutation, c is the number of cycles, h is number of hurdles and parameter f is 1 if fortress is present in the graph otherwise 0.Performing a reversal operation can result in splitting a cycle, merging two cycles or no change in the cycle. Let Δc denote the change in the number of cycles after performing a rearrangement operation, then $\Delta c = \{-1, 0, +1\}$. Based on the value of Δc, a reversal operation is classified as split ($\Delta c = 1$), neutral ($\Delta c = 0$) or joint ($\Delta c = -1$). The reversal operation is valid if $\Delta d = -1$, i.e. $\Delta(h + f - c) = -1$. An optimal (minimal) sorting sequence consists of only valid reversals.

3 HANDLING HURDLES AND FORTRESS

To make the implementation simpler, the existing program fails to produce solution if the input permutation has hurdle or fortress. In the proposed work, Siepel's method (2002) is used for finding the possible positions of cut points for applying valid reversals in their presence.

3.1 *Hurdle Graph*

Hurdle graph is constructed for the unoriented components of the breakpoint graph in order to determine their type as simple hurdle or super hurdle or protected non-hurdle. An example of a hurdle graph is given in Figure 1(b).

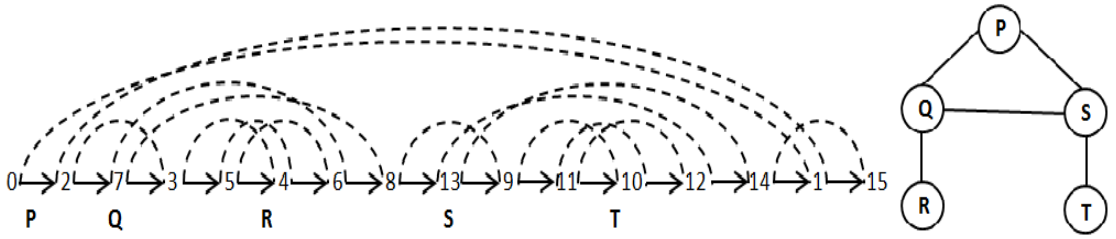

Figure 1. (a) Breakpoint graph for a signed (source) permutation (2, 7, 3, 5, 4, 6, 8, 13, 9, 11, 10, 12, 14, 1, 15). Arrow shows the direction in which black edges are traversed when traversing the edges in each cycle. It has 5 components P, Q, R, S and T. (b) Its hurdle graph, where P is a simple hurdle, Q and S are protected non-hurdles, R and T are super hurdles.

Algorithm to identify hurdles and protected non-hurdles from the hurdle graph is given below:

1. For every vertex in the hurdle graph
   a. If it has degree 2 and belongs to a cycle or has degree less than 2 and does not belong to a cycle, then the vertex is marked as simple hurdle. Else if its degree is 1, then check the degree of its adjacent vertices if any of them has degree 3 and belongs to a cycle or has degree 2 and does not belongs to a cycle, then the vertex is marked as super hurdle.
   b. If it has degree more than 2 and belongs to a cycle or has degree 2 and does not belong to a cycle, then it is not a hurdle and vertex is marked as protected non-hurdle.

3.2 *Sorting Reversals to eliminate unoriented components*

There are two types of reversals that can overcome hurdle and fortress, namely cut reversal and merge reversal.

Cut Reversal: The extremities of the reversal are in the same cycle of the unoriented component. It is neutral and transforms unoriented component into oriented component. It either eliminates hurdle or fortress but does not change number of cycle, such that $\Delta h+\Delta f =-1$. The possible positions of cut points are determined as follows-

1. Fortress is absent (f = 0)
   Perform cut on every pair of black edges of the cycles of the simple hurdle so that they can be converted into oriented component. Number of simple hurdle must be either more than 1 or can be 1 when number of super hurdle is even, so that fortress formation can be avoided. Here Δh is reduced by 1 and Δf remains unchanged so that $(\Delta h + \Delta f)$ is reduced by 1.
2. Fortress is present (f = 1)
   Perform cut on all the pair of black edges of the cycles of the super hurdle or protected non-hurdle so that super hurdle can be converted into simple hurdle component and fortress existing can be resolved. Here Δf is reduced by 1 and Δh remains unchanged so that $(\Delta h + \Delta f)$ is reduced by 1.

Merge Reversal: The extremities of the reversal are in the cycles of different components, thus rearranging the components such that number of cycles and hurdles decrease. This is an example of joint reversal. Here $\Delta c=-1$, $\Delta h+\Delta f =-2$ so that $-\Delta c+\Delta h+\Delta f =-1$. The cut points are taken on all pair of black edges between two components determined as follows-

1. Fortress is absent (f = 0)
   a. If number of simple hurdle is less or equal to 1, apply reversals between pairs of:
      i. Super hurdles.
      ii. Super hurdle and benign component with a separating hurdle.
      iii. Benign component and separating hurdle.
   b. If number of simple hurdles is 2, apply reversals between pairs of:
      i. Super hurdles.
      ii. Super hurdle and a benign component with a separating hurdle.
      iii. Benign component and a separating hurdle.
      iv. Simple hurdles when number of super hurdle is even.
   c. If number of simple hurdles is more than 2, apply reversals between pairs of:
      i. Super hurdles.
      ii. Super hurdle and a benign component with a separating hurdle.
      iii. Benign component and a separating hurdle.
      iv. Simple hurdles.
2. Fortress is present (f = 1)
   a. When double super hurdle is present, apply reversals between pairs of:
      i. Super hurdles which form double super hurdle.
      ii. Super hurdle and a benign component with a separating super hurdle which form double super hurdle.
   b. When double super hurdle is not present, apply reversals between pairs of super hurdle and a protected non-hurdle.
   c. Apply reversals between pairs of super hurdle and benign component in the same protected non-hurdle.

4 EMPIRICAL RESULTS

Both breadth first approach and depth first approach were implemented to generate all the optimal sorting traces. First the handling of hurdles was incorporated into the algorithm proposed by Braga et al. (2007). Then, to improve its efficiency with respect to time and memory the BFA method was modified. The comparison with the existing traces is avoided by using the concept given by Baudet and Dias (see section 2.3). Next, DFA method was modified to take care of hurdles and recursion is used for the depth first expansion of the solution space (Amritanjali and Sahoo, 2013). A comparative test was carried out with random permutations generated with two test cases, keeping $n=d$ and $n=2d$. Testing was also done with artificial permutations containing hurdles and fortress. All the tests were carried out on an AMD A-10 2.5 GHz

processor with 8 GB RAM running Windows 8.1 64 bit. JVM was allocated with maximum memory 1.5 GB (using parameter -Xmx1550m). The correctness of the implementation has been verified by comparing its output with that of the baobabLuna package for permutations without containing hurdles. The average of the output obtained on these random permutations was taken to get a generalized result. Elapsed time (in seconds) is computed by taking the system's clock value before and after execution with the help currentTimeMillis() method and the resulting graphs are shown in Figure 2.

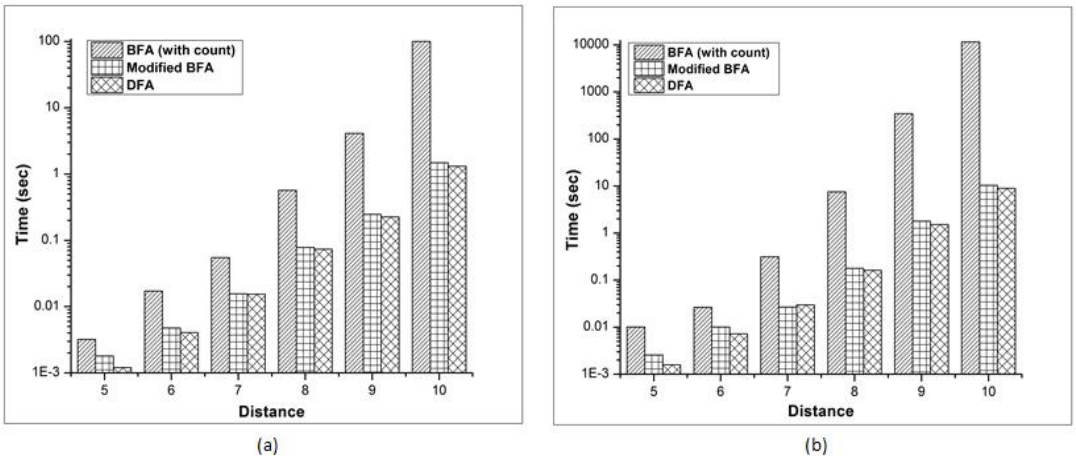

Figure 2. Average execution time for (a) n=2d and (b) n=d.

Modified BFA and DFA take lesser time than original BFA, as in both the methods exploring of those branches are avoided which is not leading to new trace. The difference in execution time becomes significantly larger as size of the permutation and reversal distance increase. The modified BFA and DFA are more or less similar with respect to time required for computation. Memory analysis was done between the modified BFA and DFA to see the improvement in memory usage when exploring the solution space in depth first manner in comparison to the breadth-first approach. Memory used (in bytes) is computed with the help of separate thread measuring memory at frequent intervals using the methods- totalMemory() and freeMemory(). The graphs for the results obtained are shown in Figure 3.

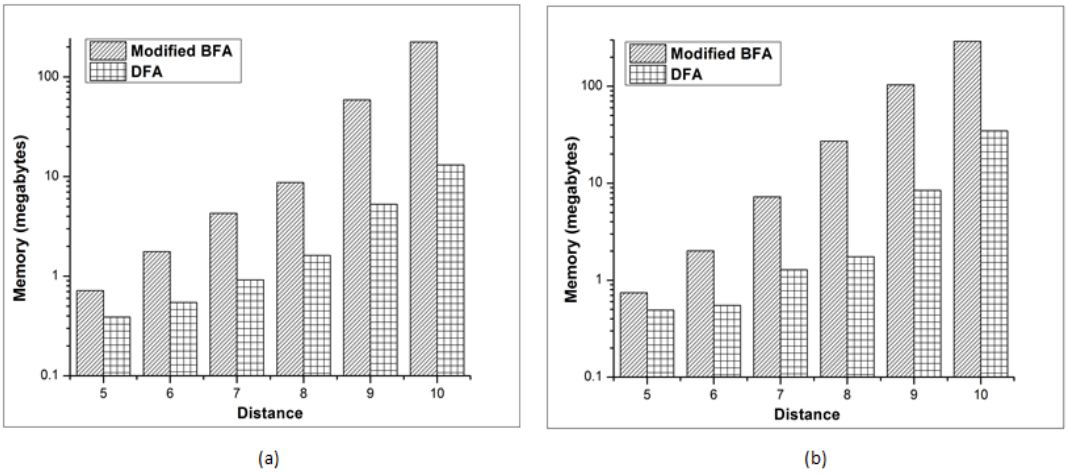

Figure 3. Average memory used for (a) n=2d and (b) n=d.

Memory consumption is more in BFA than in DFA. This is due to the fact at a time only one branch is explored and memory is freed before moving to next branch. Moreover, the plotted graphs show decrease in average execution time and used memory as size of the permutation increase for a given distance, as there is decrease in the average number of traces.

Table 1 shows the output for various permutations with different types of hurdle(s).

Table 1. Computation results for processing random permutations with hurdles. The time and memory measurements are for DFA.

Input permutation	d	Hurdle	Fortress	Traces	Time (sec)	Memory (bytes)
2,1,3,-4,5,7,9,8,10,6,11,13, 14, 12	12	Simple, protected nonhurdle	0	172	2.421	903704
2,7,3,5,4,6,8,13,9,11,10,12,14,1	13	Simple, Super, protected nonhurdle	0	1048	8.790	2893504
17,1,3,8,4,6,5,7,9,11,13,12,14,10,15,2,16	16	Super, protected nonhurdle	1	1048	120.631	3850136

5 CONCLUSIONS

Our implementation removes the problem of hurdles and generates the possible evolutionary scenarios that eliminate them. As the implementation is based on the Siepel's work, so the complete set of sorting traces are obtained. For efficiency, the double super hurdles are not implemented as the chances of their occurrence are negligible. The hurdle version has been implemented using both BFA and DFA approaches. As expected, DFA is more memory efficient than BFA. This version could further be extended to form a parallel version of the algorithm. To reduce the solution space, biological constraints like the common interval constraint can be applied.

REFERENCES

Amritanjali & Sahoo, G. 2013. Exploring the Solution Space of Sorting By Reversals: A New Approach, International Journal of Information Technology, 3(2): 98-104.

Badr, G., Swenson, K.M. & Sankoff, D. 2011. Listing All Parsimonious Reversal Sequences: New Algorithms and Perspectives. *Journal Of Computational Biology*, 18(9): 1201–1210.

Baudet, C. & Dias, Z. 2010. An improved algorithm to enumerate all traces that sort a signed permutation by reversals. *Proc. ACM Symposium on Applied Computing*. 1521–1525.

Braga, M.D.V., Sagot, M., Scornavacca, C. & Tannier, E. 2007. The solution space of sorting by reversals, *Proc. Int'l Symp. Bioinformatics Research and Applications (ISBRA 2007)*. 4463: 293–304.

Braga, M.D.V. 2009. baobabLuna: The solution space of sorting by reversals. *Bioinformatics*. 25(14):1833-1835.

Braga M.D.V. & Stoye, J. 2010. The solution space of sorting by DCJ. *Journal of Comput. Biol*. 17(9): 1145-1165.

Diekert, V. & Rozenberg G. 1995. *The Book of Traces*. World Scientific: Singapore.

Hannenhalli, S. & Pevzner, P.A. 1995. Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals). *Proc. 27th Ann. ACM Sym on the Theory of Comput*., 178–189.

McLysaght, A., Seoighe, C. & Wolfe, K.H. 2000. High frequency of inversions during eukaryote gene order evolution. In Sankoff, D., and Nadeau, J.H., eds., *Comparative Genomics*, 47–58, Kluwer Academic Press, NY.

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