Efficient Solution to the Problem Genome Sorting by Reversals

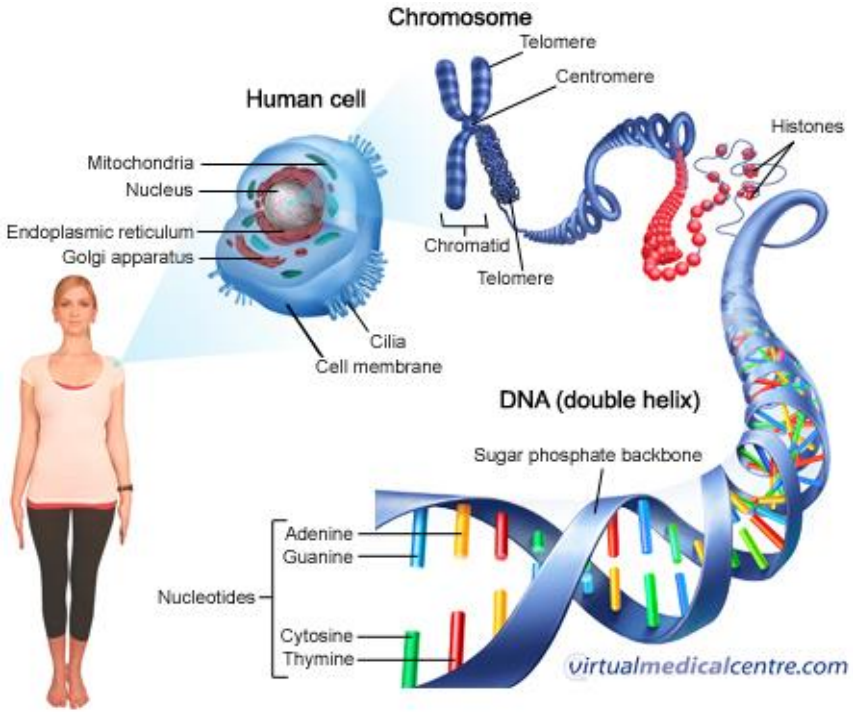
Guided by:
Dr. Amritanjali

Presented By:
Beethika Tripathi
MT/CS/10017/2013

Dept. of Computer Science and Engineering,
Birla Institute of Technology, Mesra,
Ranchi, Jharkhand

Introduction

- All species have said to be evolved from the common ancestor.
- When a genome replicates into another genome there can be some changes or errors this leads to various mutations and formation of a new species.

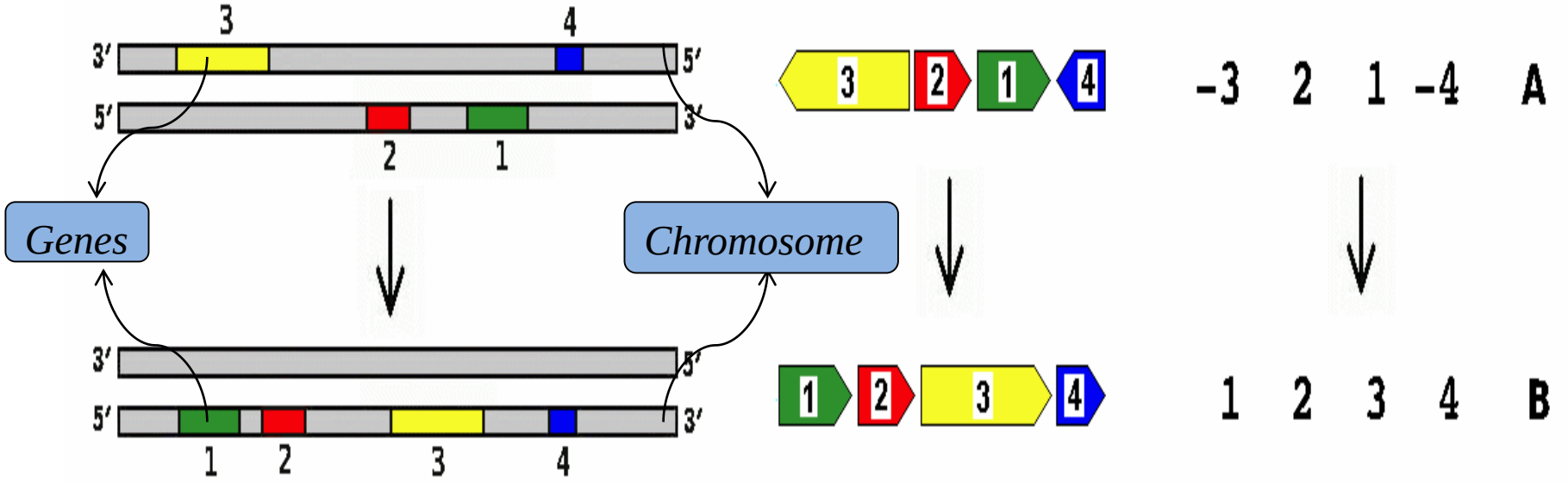


Genome Rearrangements and Evolution

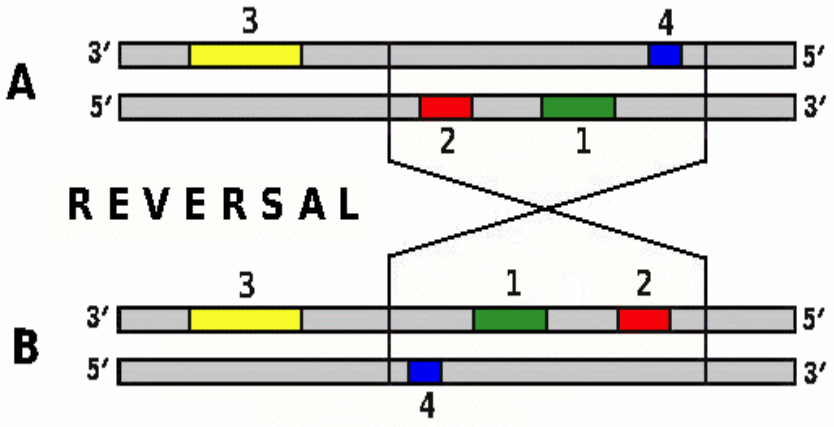
- Large-scale rearrangements in the genomes accompany evolution of species
- It can change number of chromosomes or number and arrangement of genes in the chromosomes
- Closely related species have more similar genomes than distantly related species
- Evolutionary scenario between two species is described by the sequence of rearrangements that changed the arrangement of the shared markers in their genomes
- Reversal is the most common rearrangement operation that reverses the order and orientation of the genes in a chromosomal segment

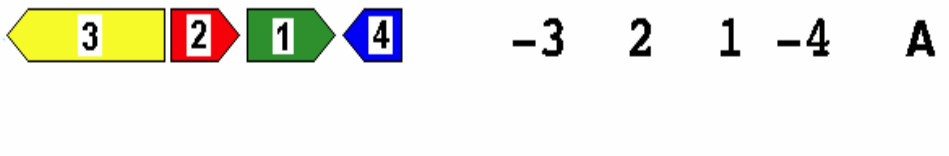
Genome Representation

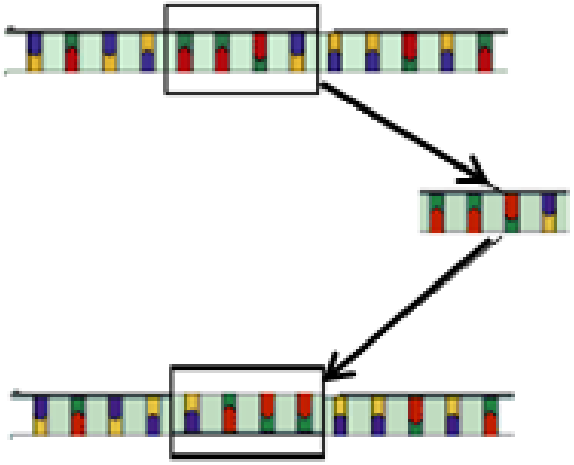
Genomes are represented by the list of homologous markers between them.



Reversal Operation







π={-3 2 1 -4}

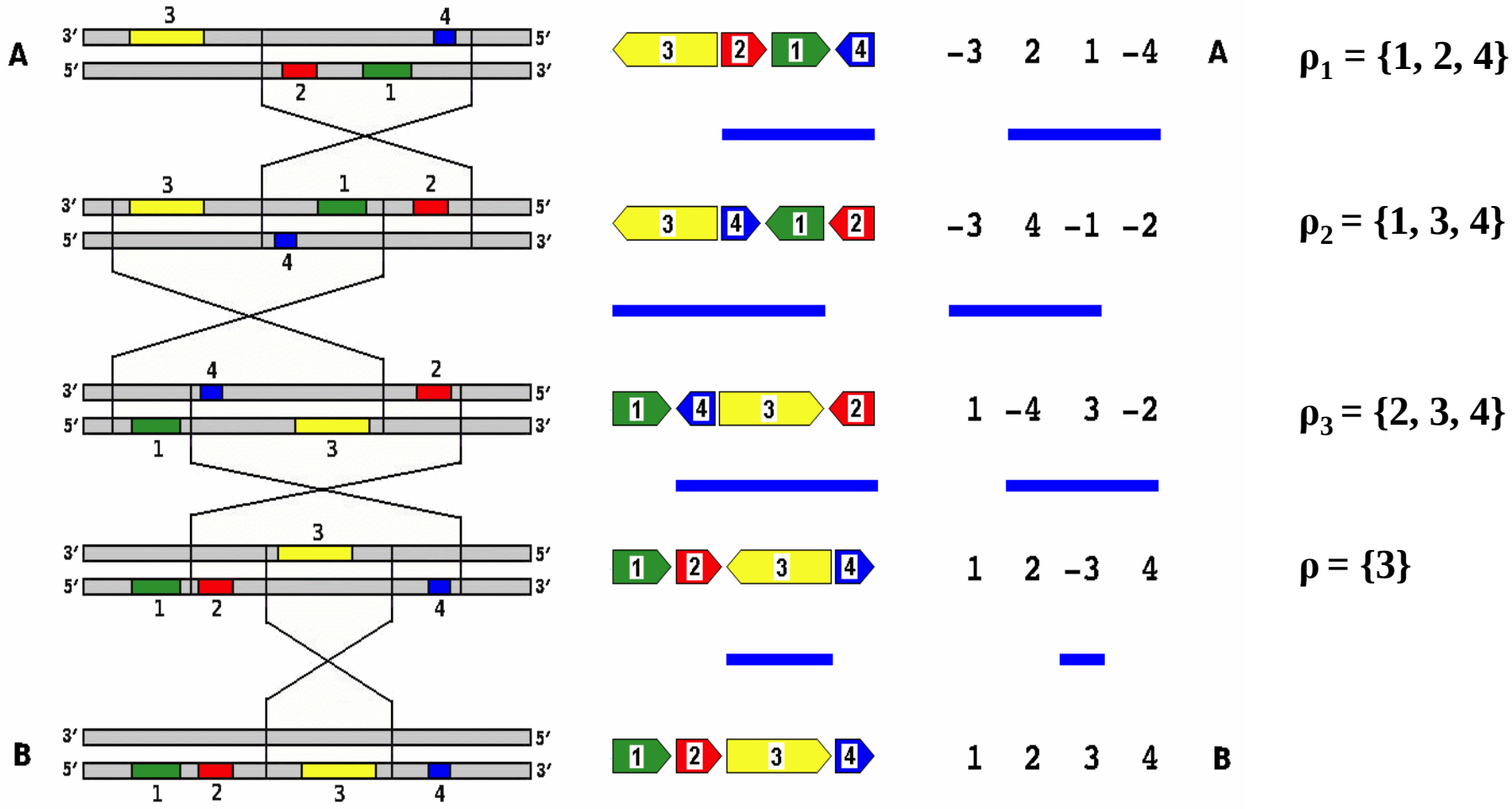
ρ={1,2,4}

$\pi \circ \rho$={-3 4 -1 -2}

Problem Definition

Sorting by Reversals Problem of finding the minimal sequence of reversals, can transform the shared gene order of source genome into that of target genome.

Goal Finding efficient solution to the problem of Genome Sorting by Reversals



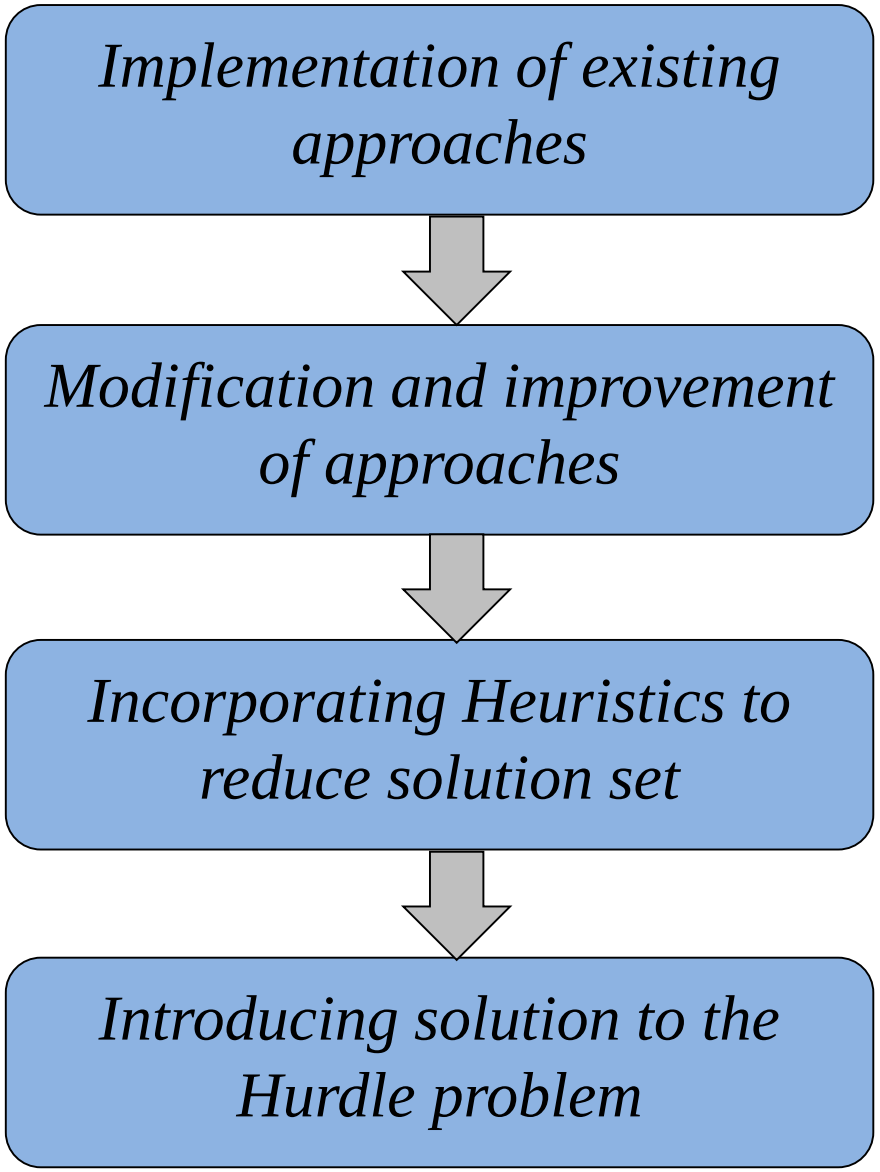
Minimum number of reversals required to transform one genome into the other is **reversal distance**

The sequence **{1, 2, 4} {1, 3, 4} {2, 3, 4} {3}** is an **optimal sorting sequence**

Literature Review

- First polynomial-time algorithm for sorting signed permutation by reversals was given by **Hannenhalli and Pevzner** that runs in polynomial time for permutations of **n** genes
- **Tannier** et al., found one optimal solution in < $O(n^2)$
- **Siepel** proposed an algorithm that listed all sorting reversals (ASR) in $O(n^3)$ time complexity.
- Concept of traces was given by **Bergeron** et al. by grouping the similar sequences into equivalence class.
- **Braga** et al. combined the algorithm of Siepel and with the concept of traces to list all solutions but does not provide solutions in the presence of hurdles and fortress.
- **Baudet** et al. used the approach of exploring the solution space in depth first manner using stack.

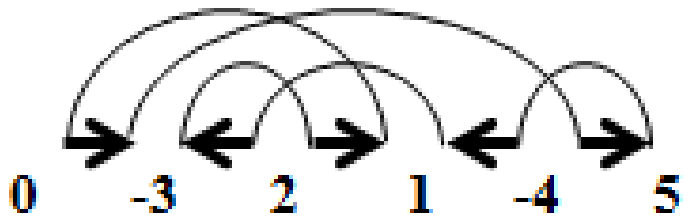
Proposed Methodology



Breakpoint Graph

Breakpoint Graph Graph constructed from the source and target permutations which is used for finding:

- The reversal distance
- Optimal sequence of reversals for sorting the source permutation.

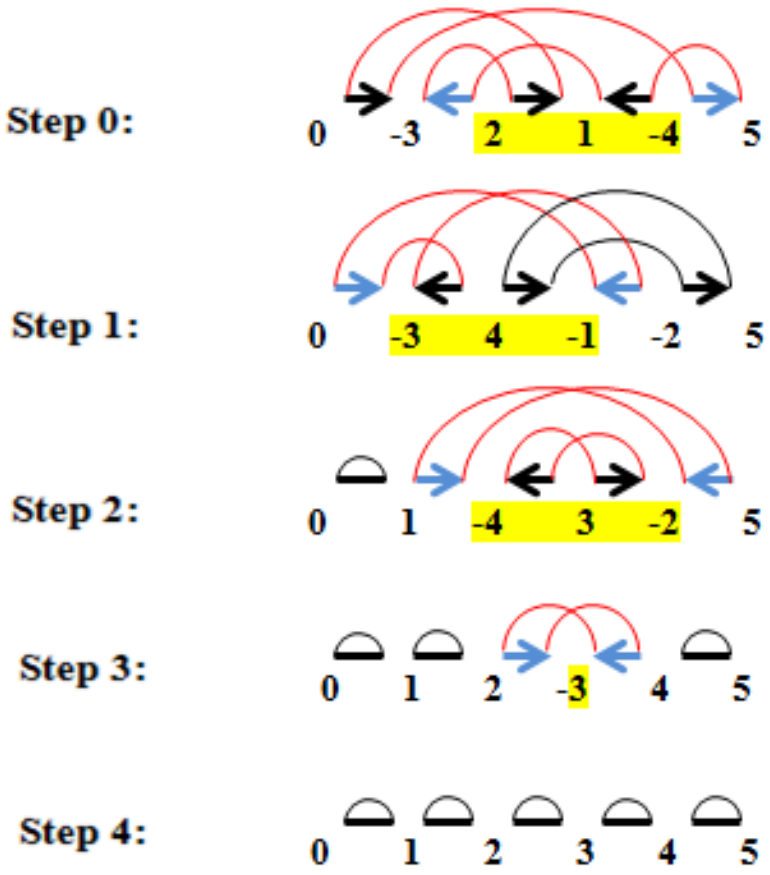


d= (n+1) – c

n- no. of genes

c- no. of cycles

Sorting using Breakpoint Graph



All Sequences of Sorting Reversals

- Siepel gave an algorithm to list all optimal 1-sequences in $O(n^3)$ called All Sorting Reversals.
- Using his algorithm all possible 1-sequences are computed that will bring the given permutation one step closer to the target.
- Each of them are applied to the original permutation to form the new set permutations having reversal distance $d(\pi)-1$.
- New set of reversals are computed for each permutation.
- This 1-sequence when combined with the predecessor ρ in normal form to give optimal 2-traces.
- Algorithm is repeated till all $d(\pi)$-traces are generated reducing the distance to zero and each of them sort permutation π to π_T.

Grouping Solutions into Equivalence Class

Two sequences are said to be equivalent if one can be obtained from another by sequence of commutations of non-overlapping reversals.

Many optimal solutions are **equivalent**:

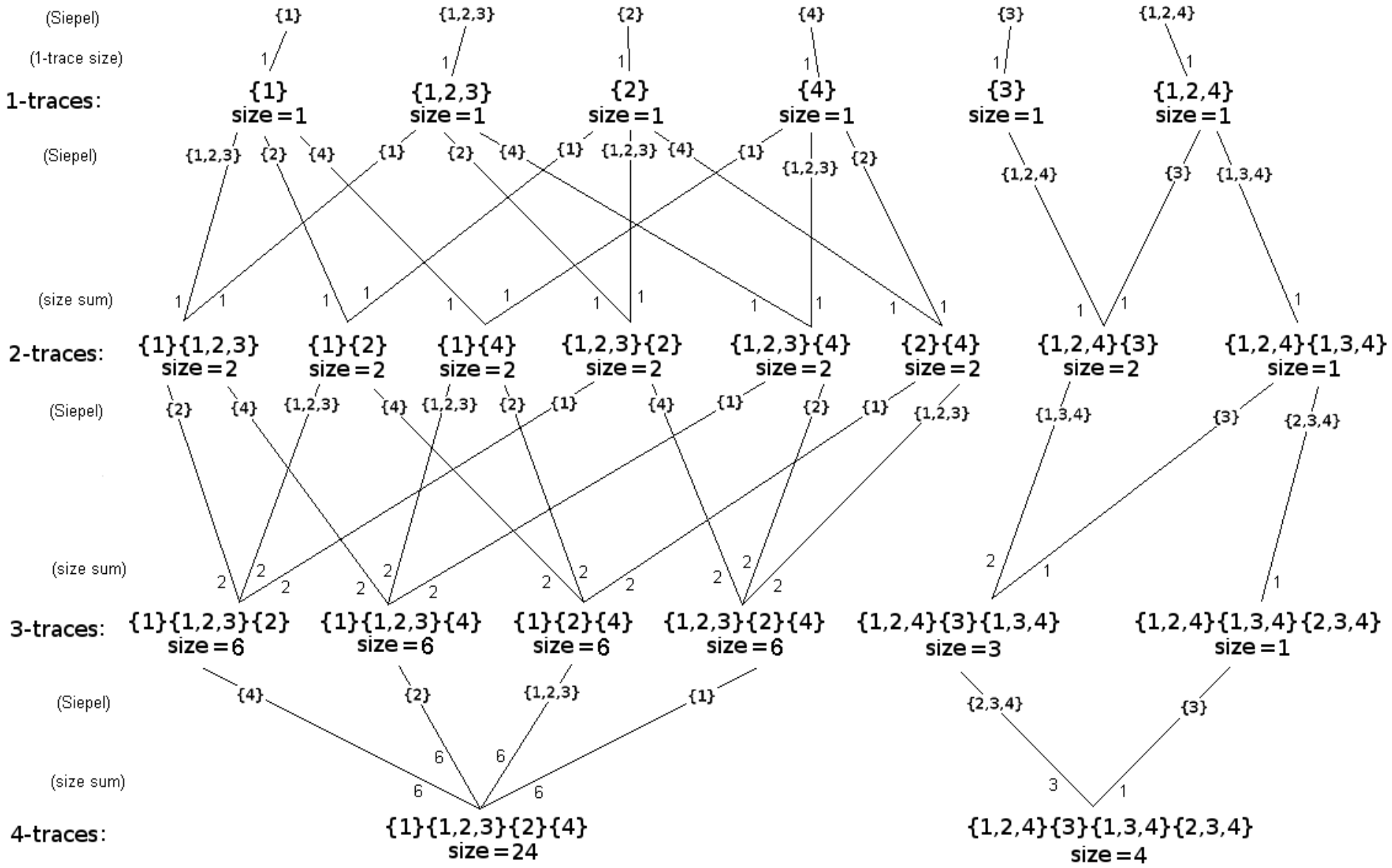
$\pi = (-3, 2, 1, -4)$

{1, 2, 4} {1, 3, 4} {2, 3, 4} {3}
{1, 2, 4} {1, 3, 4} {3} {2, 3, 4}
{1, 2, 4} {3} {1, 3, 4} {2, 3, 4}
{3} {1, 2, 4} {1, 3, 4} {2, 3, 4}

Similar sequences are grouped into equivalence class and represented using normal form trace.

The traces are a compact representation of the **set of all optimal sorting sequences** for a permutation π

All Sorting Traces with count of no. of sorting sequences in each trace



Trace Generation Algorithm

The **TRACE_GENERATION** Procedure is given as follows:

1. Initialize root node using PROCESS_PERMUTATION.
2. Add all the reversals in the node to the trace set and initialize its count as 1.
3. For each i-trace in the trace set
    a. For each reversal in the i-trace
        Apply reversal on the permutation
    b. For the new permutation initialize next level node by calling PROCESS_PERMUTATION.
    c. For each reversal ρ in the node
        i. Check the distance is reduced by 1 after applying ρ.
        ii. Find out the position of the last overlapping sub-word in trace with ρ.
        iii. Append ρ at the next sub-word while maintaining the lexicographic ordering, if it was the last sub-word then append at the end by forming a new sub-word.
        iv. Now check if (i+1)-trace formed is already present in new trace set.
        v. If present then just increase the count of the trace otherwise add it to the new trace set.
4. Trace set is replaced with new trace set.
5. Step 5 and 6 is repeated till the distance reduces to 0.
6. Trace set is the output.

Solution Space

No	Input Permutation	d	Sequence count	Trace count
1.	−3, 2, 1,−4	4	28	2
2.	−4, 1,−3, 6,−7,−5, 2	6	204	5
3.	−6, 5, 7,−1,−4, 3, 2	6	496	8
4.	−4,−3, 12,−11,−8, 10, 9, 7,−6,−5, 2,−1	8	31752	6
5.	−4, 3, 12,−11,−8, 10, 9, 7,−6,−5, 2,−1	9	407232	14
6.	−12, 11,−10, 6, 13,−5, 2, 7, 8,−9, 3, 4, 1	10	8278540	2151
7.	−12,11,−10,−1,16,−4,−3,15,−14,9,−8,−7,−2,−13,5,−6	12	505634256	21902

Theoretical Complexity

- No. of sequences is $O(N. n^{kmax})$
- Finding all 1-sequence takes $O(n^3)$ time
- Processing prefix takes $O(n^4)$ time this is computed as:

  No of 1-sequences generated * Adding them to previous prefix
- Comparison to merge equivalent traces takes $O(n^3 \log(N. n^{kmax})$

  No of sequences * (No of 1-seq * comparing with all traces)
- So, theoretical complexity for generating traces: $O(N. n^{kmax + 4})$

n is number of elements in the sequence.

N is total number of traces.

The width (k) of a trace t is defined as size of the biggest subset of reversals of t such that every pair of reversals in this subset commutes.

Kmax is the max width at the particular i-trace.

Challenges

- Solution space is too large.
- **Time** required for computation increases **exponentially** with size of permutation.
- **Huge memory** requirement to store all the solutions.
- Fails to give results in the presence of **hurdles and fortress.**

Proposed Approaches

Improving efficiency with respect to time and memory by generating traces without counting no. of sequences in each trace

All Sorting Traces without count

- At every step verify if the generated i-trace is a new trace or not by checking the normal form order constraints.
- If the trace is not new then it is discarded.
- **Only those traces in which the reversal appends at the end as this will always leads to the generation of new trace.**
- Computation time can be saved in matching the trace with the existing traces.

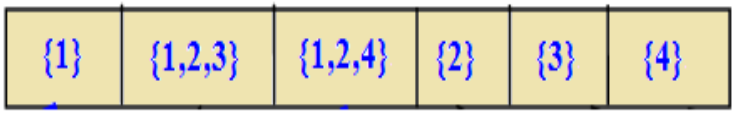
Breadth First Approach

The **TRACE_GENERATION** Procedure of BFA is given as follows:

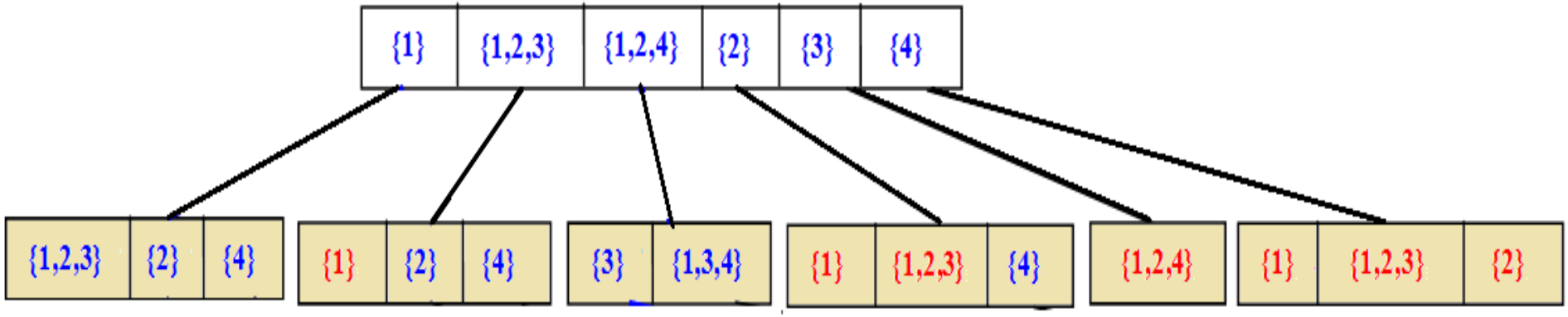
1. Initialize root node using PROCESS_PERMUTATION.
2. Add all the reversals in the node to the trace set.
3. For each i-trace in the trace set
   a. For each reversal in the i-trace

      Apply reversal on the permutation

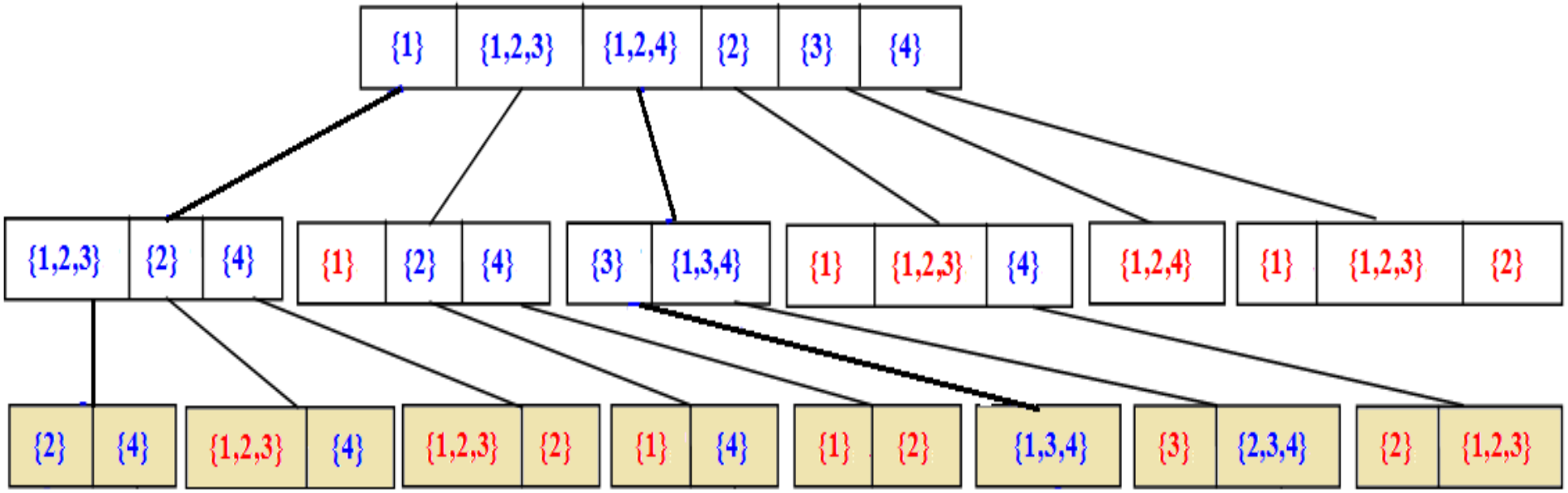
   b. For the new permutation initialize next level node by calling PROCESS_PERMUTATION.
   c. For each reversal ρ in the node
      i. Check the distance is reduced by 1 after applying ρ.
      ii. If ρ has to be appended at the end of trace (as described above) then do so. Otherwise, move to next ρ.
      iii. Add the trace to the new trace set.
4. Trace set is replaced with new trace set.
5. Step 5 and 6 is repeated till the distance reduces to 0.
6. Trace set is the output.



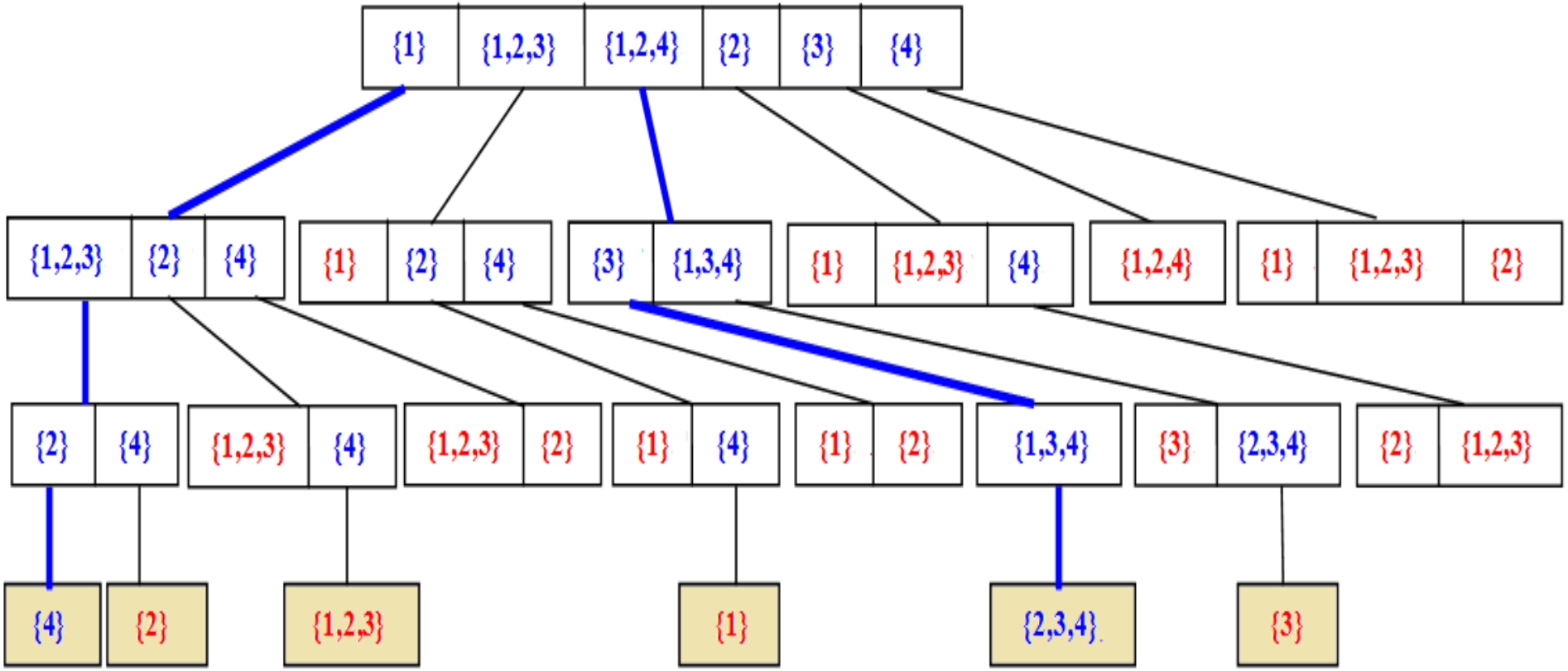
Tree structure of the traces that sort the permutation (−3 2 1 −4)



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



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



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

Improving efficiency with respect to memory

Depth First Approach

- To generate (i+1)-trace it is not necessary to calculate all i-sequences.
- To improve memory usage each branch starting from root node is completely explored before moving to next branch.
- All the 1-sequences are generated and stored in normal form order w.r.t. sub-trace generated so far.
- **Once the branch is explored perform backtracking to explore the new branch**. Repeat this till all the branches have been explored.

TRACE_GENERATION procedure

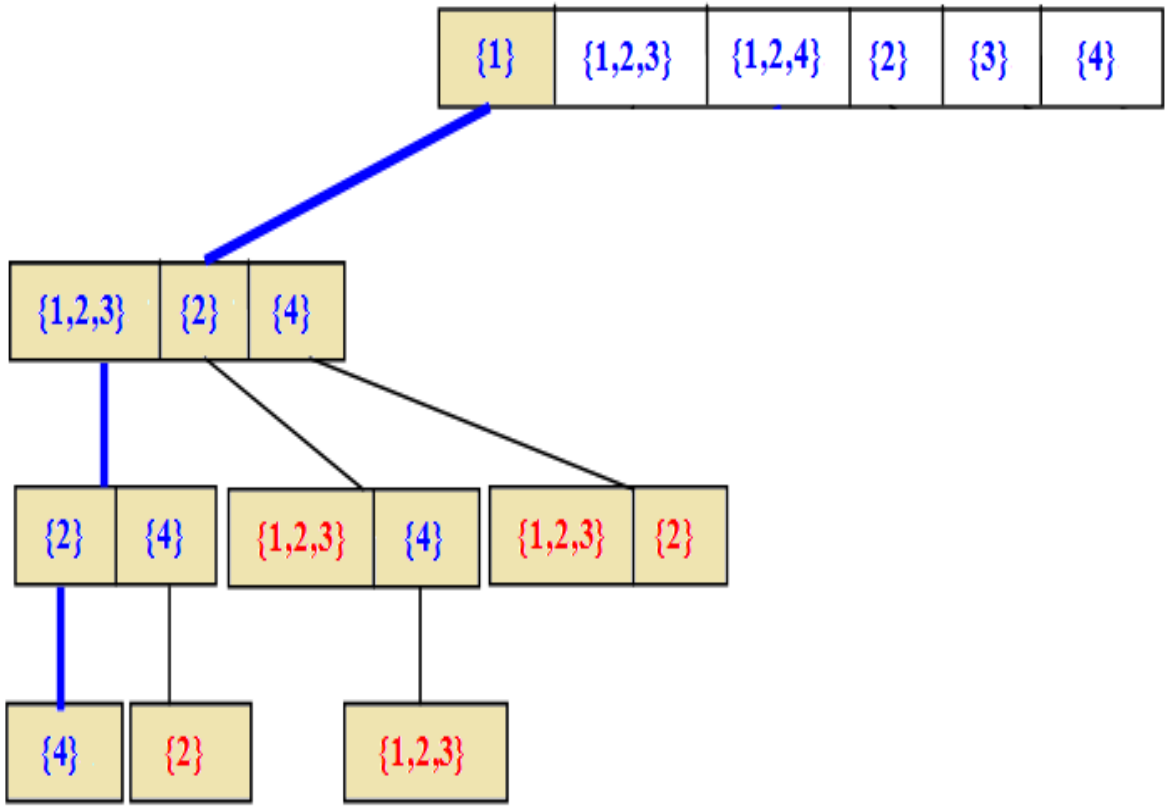
1. Initialize root node using PROCESS_PERMUTATION.
2. For each reversal

   Find the number of solutions generated by the reversal by calling recursive procedure EXPAND.

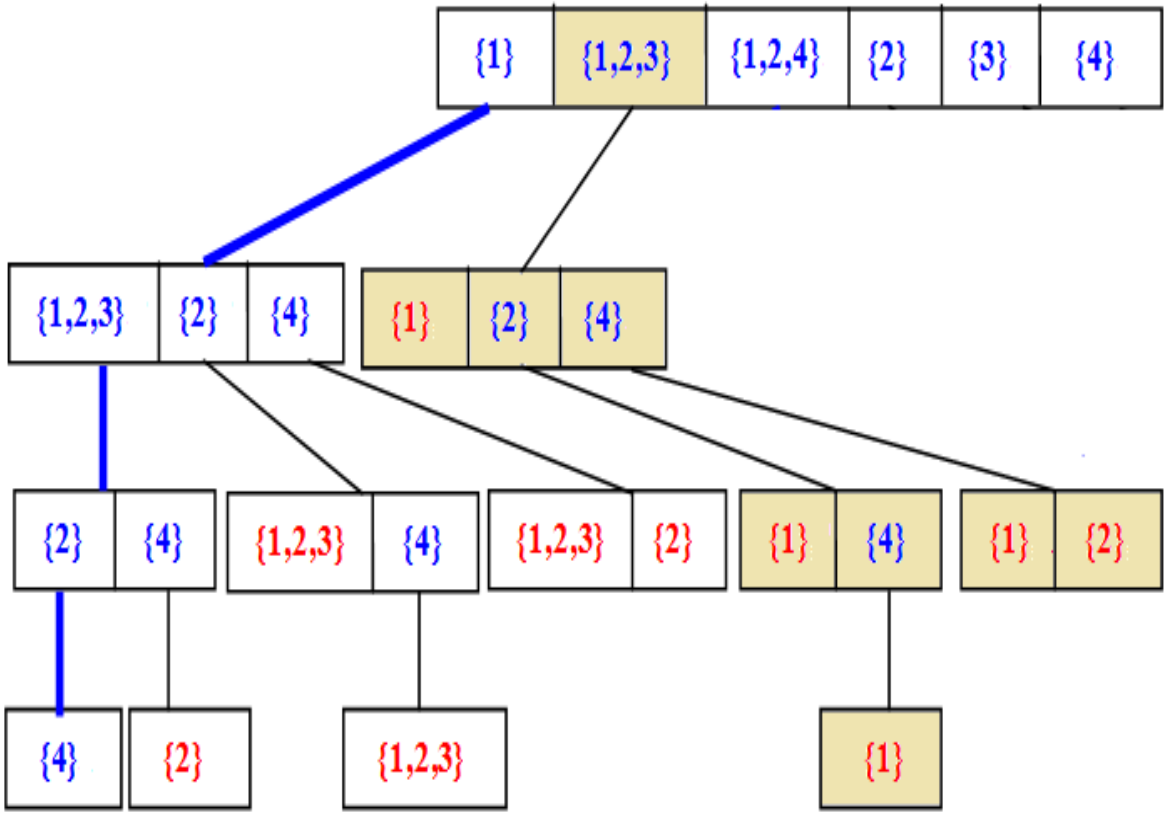
EXPAND procedure

1. Apply reversal on given genome to get new genome sequence.
2. If level is equal to d(π) print the trace.
3. Generate next 1-sequence of reversals and store in new node at next level.
4. For each 1-sequence in node

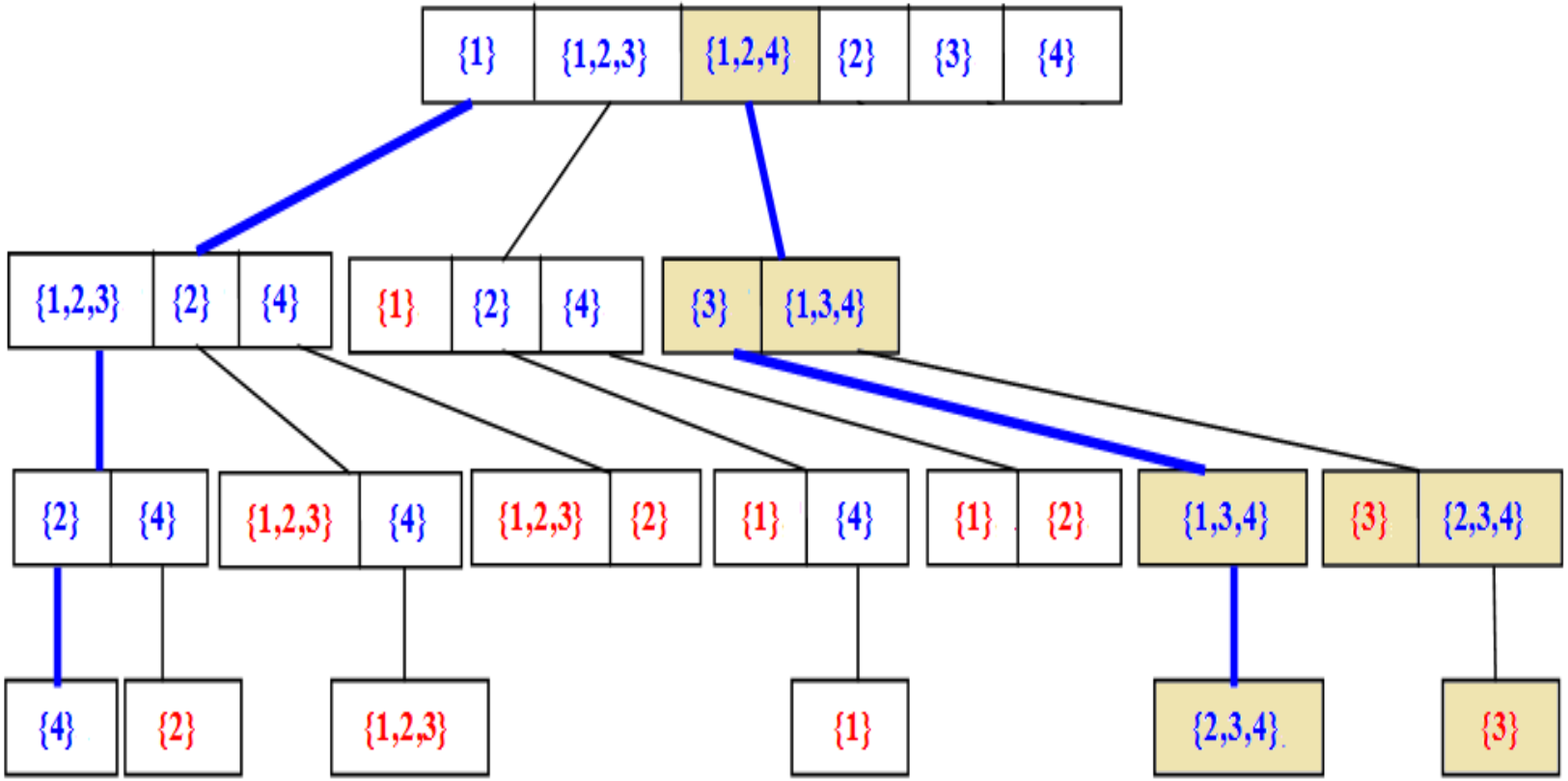
   If appending 1-sequence to predecessor sequence leads to new trace then call EXPAND to further explore the branch otherwise skip the reversal and continue with next reversal.
5. Return



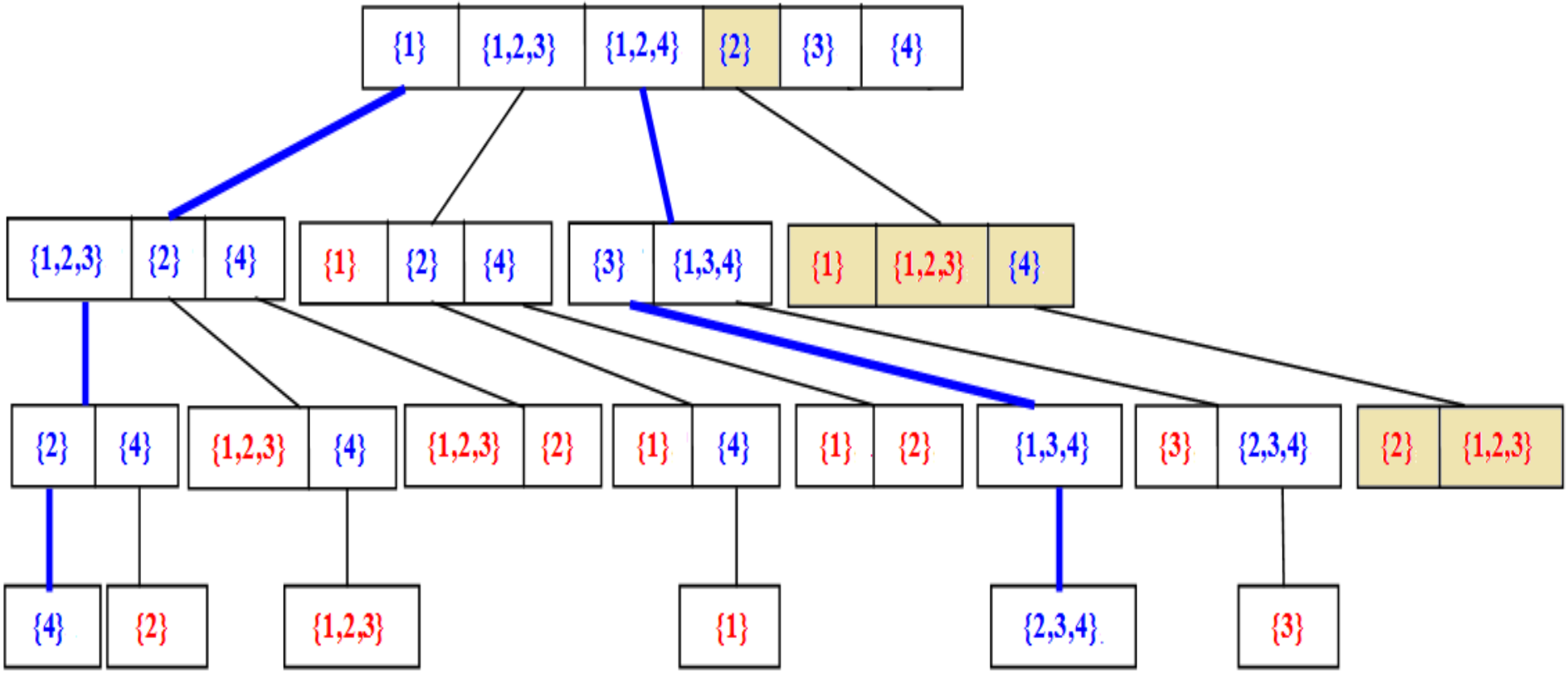
Tree structure of the traces that sort the permutation (−3 2 1 −4)



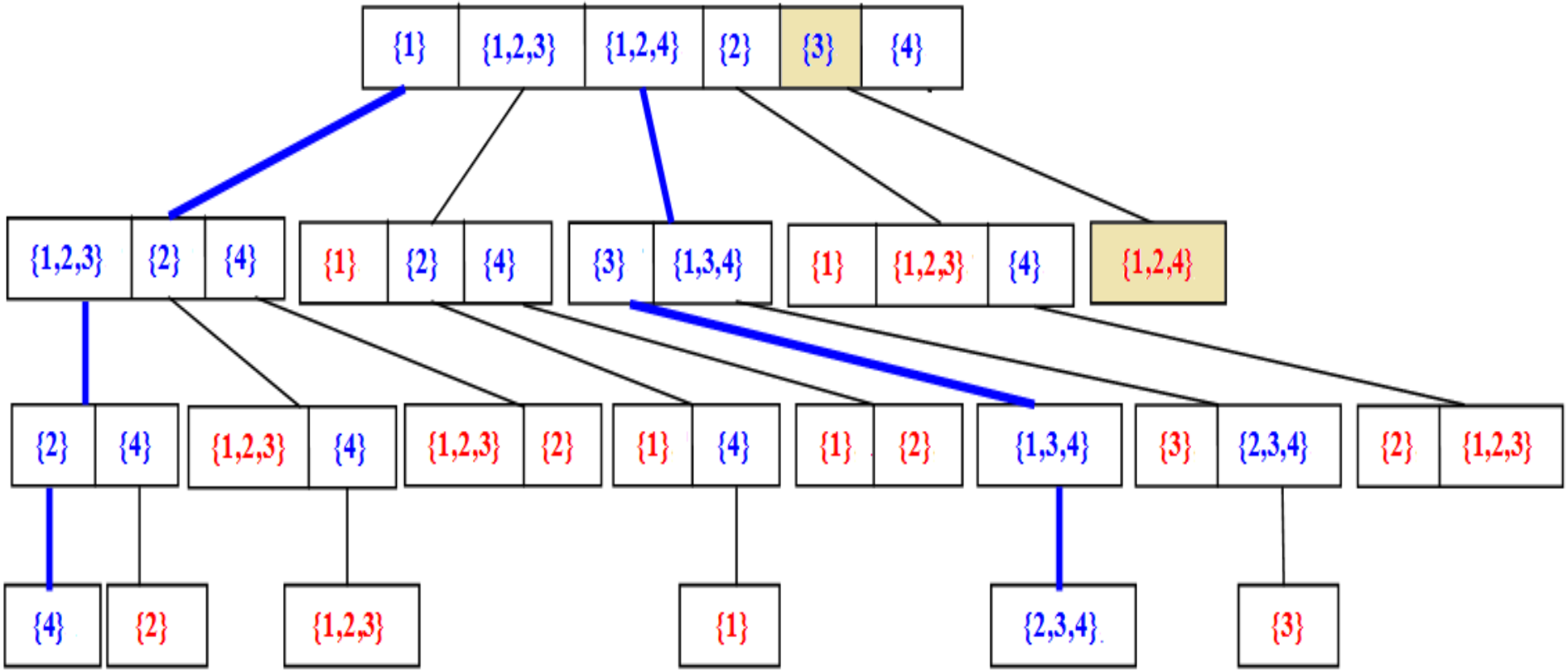
Tree structure of the traces that sort the permutation (−3 2 1 −4)



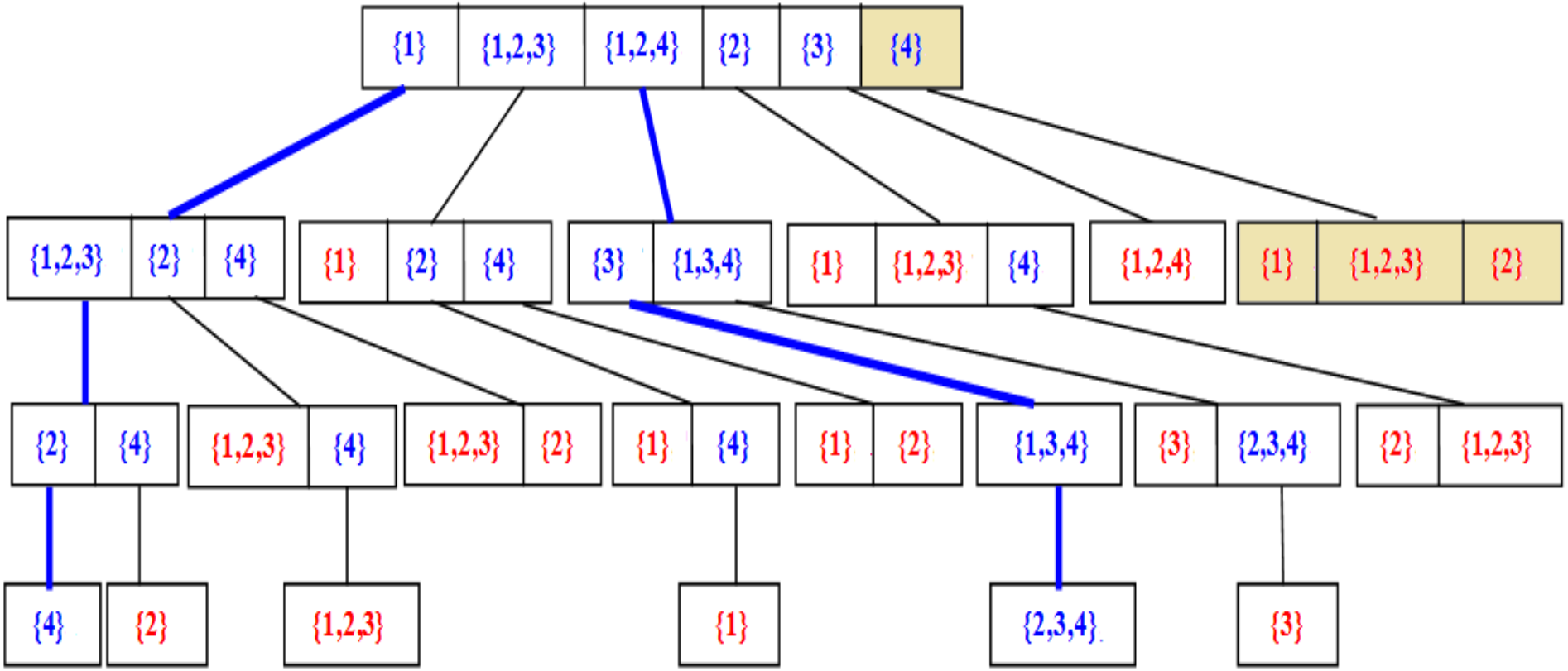
Tree structure of the traces that sort the permutation (−3 2 1 −4)



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



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



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

Efficiency

Memory Efficient

- All the sequences are not stored in memory, as we are not providing count of total number of sequences in each trace.
- Only those branches are expanded in which reversals are added at the end of the partial-trace.

Time Efficient

- As traces are generated without generating all the sequences and not comparing with other traces, makes it time efficient.
- Time saved by not comparing with all traces: $O(N.n^{kmax+3} \log(N.n^{kmax})$

Comparison Table

No	Input Permutation	d	Trace	Memory (bytes)			Time (sec)		
				With Count	Without Count		With Count	Without Count	
					BFA	DFA		BFA	DFA
1.	−3, 2, 1,−4	4	2	395376	315512	296064	0.007	0.001	0.001
2.	−4, 1,−3, 6,−7,−5, 2	6	5	826552	700640	686584	0.022	0.005	0.005
3.	−6, 5, 7,−1,−4, 3, 2	6	8	714560	254680	227440	0.038	0.010	0.010
4.	−4,−3, 12,−11,−8, 10, 9, 7,−6,−5, 2,−1	8	6	1021128	1343240	693288	0.195	0.040	0.030
5.	−4, 3, 12,−11,−8, 10, 9, 7,−6,−5, 2,−1	9	14	1014624	1993592	1132120	1.775	0.120	0.100
6.	−12, 11,−10, 6, 13,−5, 2, 7, 8,−9, 3, 4, 1	10	2151	18795824	29569592	12208712	1488.621	4.650	3.849

Listing solutions more closer to actual evolutionary scenario by applying biological constraints to filter less probable solutions.

(Heuristic Based Approach)

Biological Constraints

- To logically reduce the solution set by limiting the reversals that are practically not possible or less prone to occur.

Common Interval It is the phenomenon of listing the clusters of co-localised genes between the two genomes.

- **Inherited from the common ancestor, chances of occurrence of sequence that breaks this block is very less.**
- Discarding solutions having such reversals reduces solution set.
- 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:

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

- It is a block of elements that are contiguous in both the permutations but their respective positions may be juggled around in the permutations.

Computation

Source	Target
(−5 −2 −7 4 −8 3 6 −1)	(1 2 3 4 5 6 7 8)
(−5 −2 −7 4 −8 3 6 −1)	(1 2 3 4 5 6 7 8)
(−5 −2 −7 4 −8 3 6 −1)	(1 2 3 4 5 6 7 8)
(−5 −2 −7 4 −8 3 6 −1)	(1 2 3 4 5 6 7 8)
(−5 −2 −7 4 −8 3 6 −1)	(1 2 3 4 5 6 7 8)
(−5 −2 −7 4 −8 3 6 −1)	(1 2 3 4 5 6 7 8)
(−5 −2 −7 4 −8 3 6 −1)	(1 2 3 4 5 6 7 8)
(−5 −2 −7 4 −8 3 6 −1)	(1 2 3 4 5 6 7 8)

Common Interval

Filtering

- Filtering is done as the 1-sequences are generated.
- Each reversal is checked whether breaks the common interval or not.
- A common interval **breaks** when:
  - One end of reversal lie inside the interval.
  - Other end outside the interval.
  - Interval is not completely contained within the reversal.
  - After performing the reversal elements of common interval are not contiguous.
- **Reversals that break the interval are not added to the set of 1-sequences, thereby reducing the solution set at each step.**
- Generates solutions that define most probable evolutionary scenario.

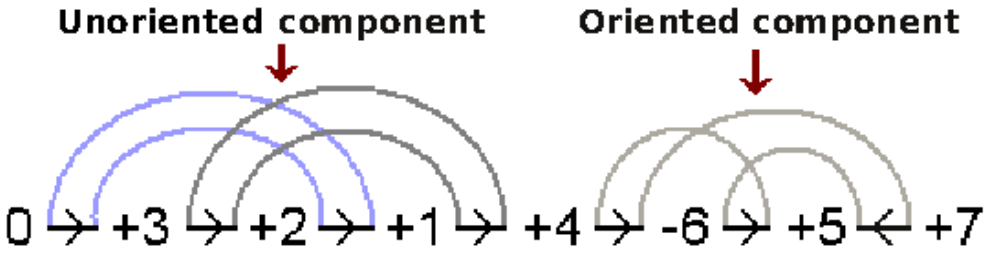
Comparing Solution Space

No	Input Permutation	d	Without CI constraints		With CI constraints	
			Sequence count	Trace count	Sequence count	Trace count
1.	−3, 2, 1,−4	4	28	2	4	1
2.	−4, 1,−3, 6,−7,−5, 2	6	204	5	108	1
3.	−6, 5, 7,−1,−4, 3, 2	6	496	8	24	1
4.	−4,−3, 12,−11,−8, 10, 9, 7,−6,−5, 2,−1	8	31752	6	24	1
5.	−4, 3, 12,−11,−8, 10, 9, 7,−6,−5, 2,−1	9	407232	14	240	2
6.	−12, 11,−10, 6, 13,−5, 2, 7, 8,−9, 3, 4, 1	10	8278540	2151	1698480	12
7.	−12, 11,−10,−1, 16,−4,−3, 15,−14, 9,−8,−7,−2, −13, 5,−6	12	505634256	21902	122862960	171

Enumerating all solutions in the presence of hurdles and fortress

(The problem of Hurdles & Fortress not solved till date)

Cycles and Components



Oriented Cycle A non-trivial cycle with at least 1 black edge in different directions i.e. divergent edge.

Component Those cycles whose grey edge overlap forms a component. **Oriented component** are those non-trivial component that have at least one oriented cycle. Rest all are **un-oriented components**.

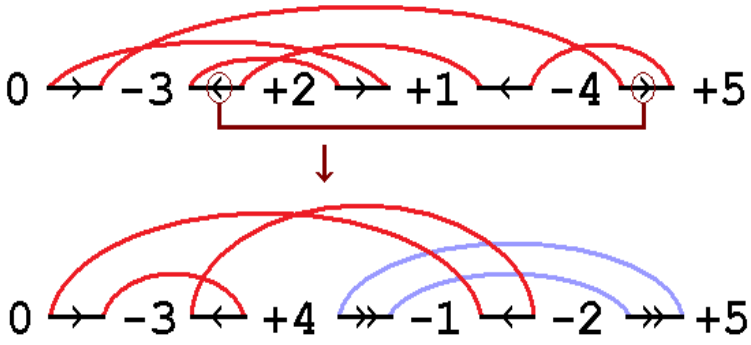
Hurdle It is an un-oriented component that does not separate two other un-oriented components.

Super & Simple Hurdle A super hurdle is a hurdle when it is eliminated a protected non hurdle emerge as a hurdle, otherwise it is a simple hurdle.

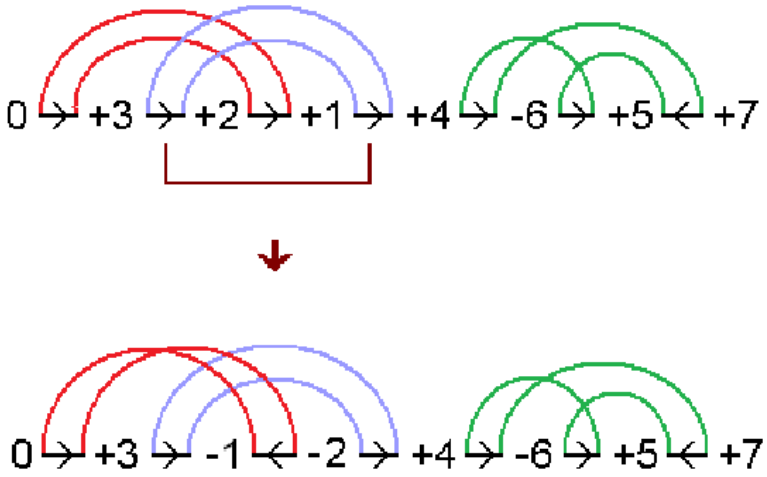
Fortress A fortress exists iff there is an odd number of hurdles and all are super hurdles.

Reversal distance = (n+1) – c + h + f ,where n is elements, c is number of cycles, h is the hurdle and f represents fortress.

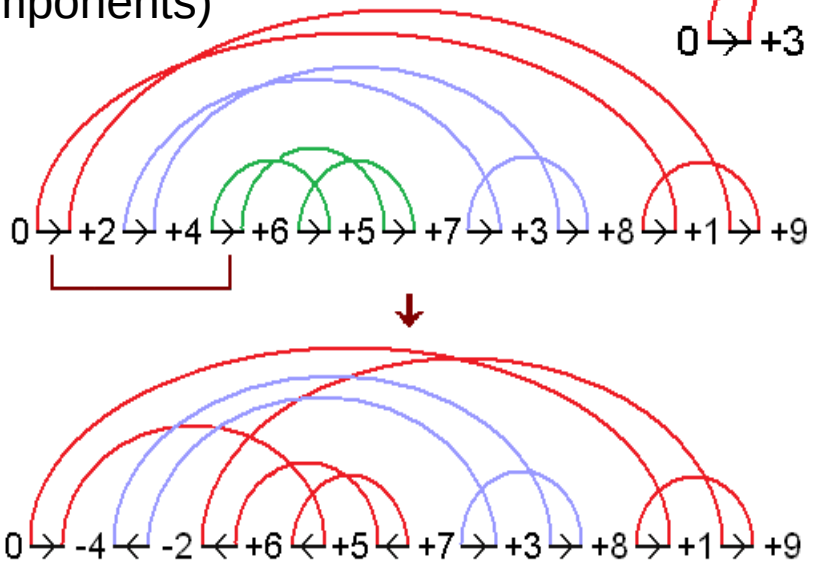
Reversals



Split Reversal
(oriented components)

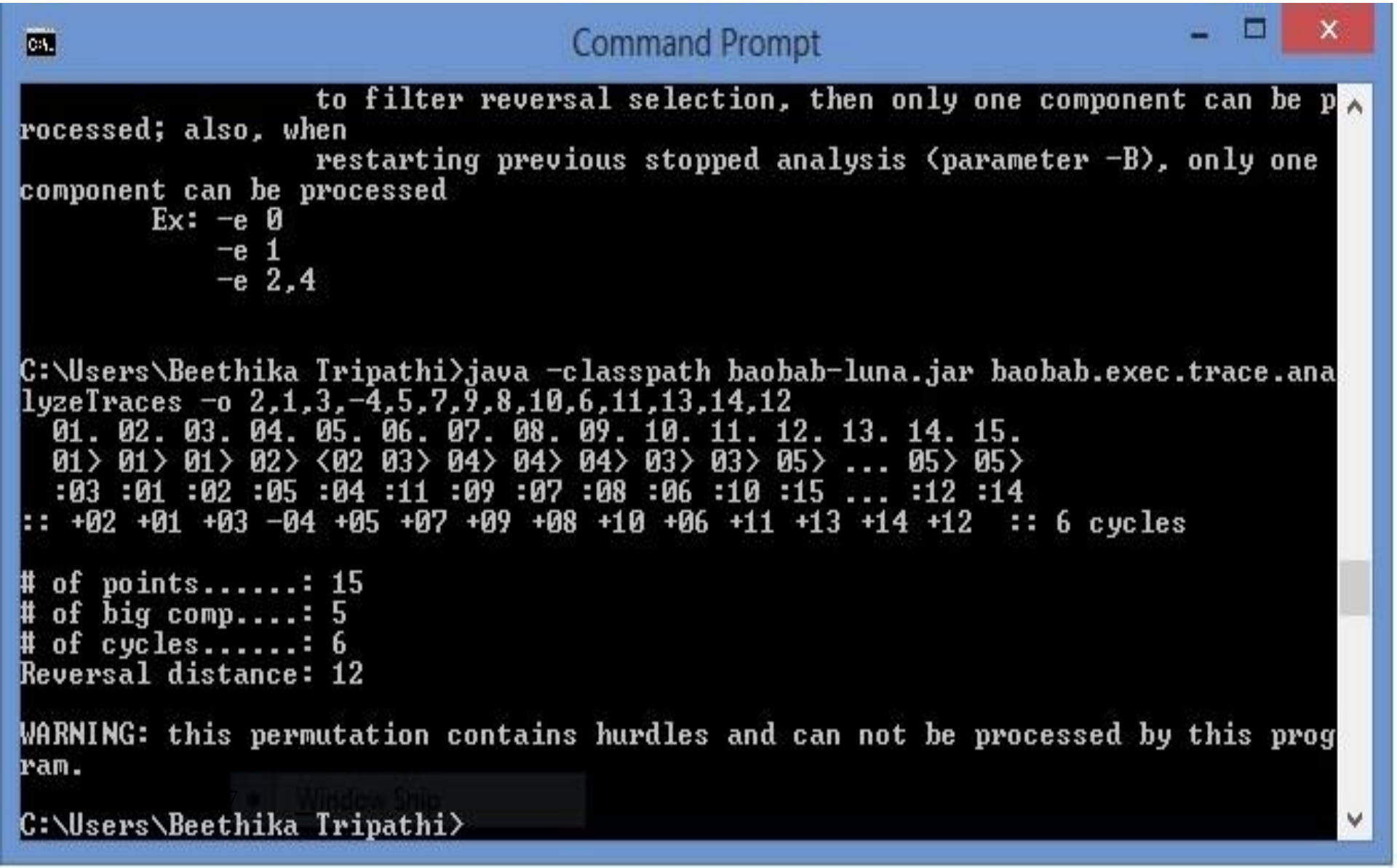


Neutral Reversal
(unoriented components)



Merge Reversal
(unoriented components)

BaobabLuna fails in presence of Hurdles



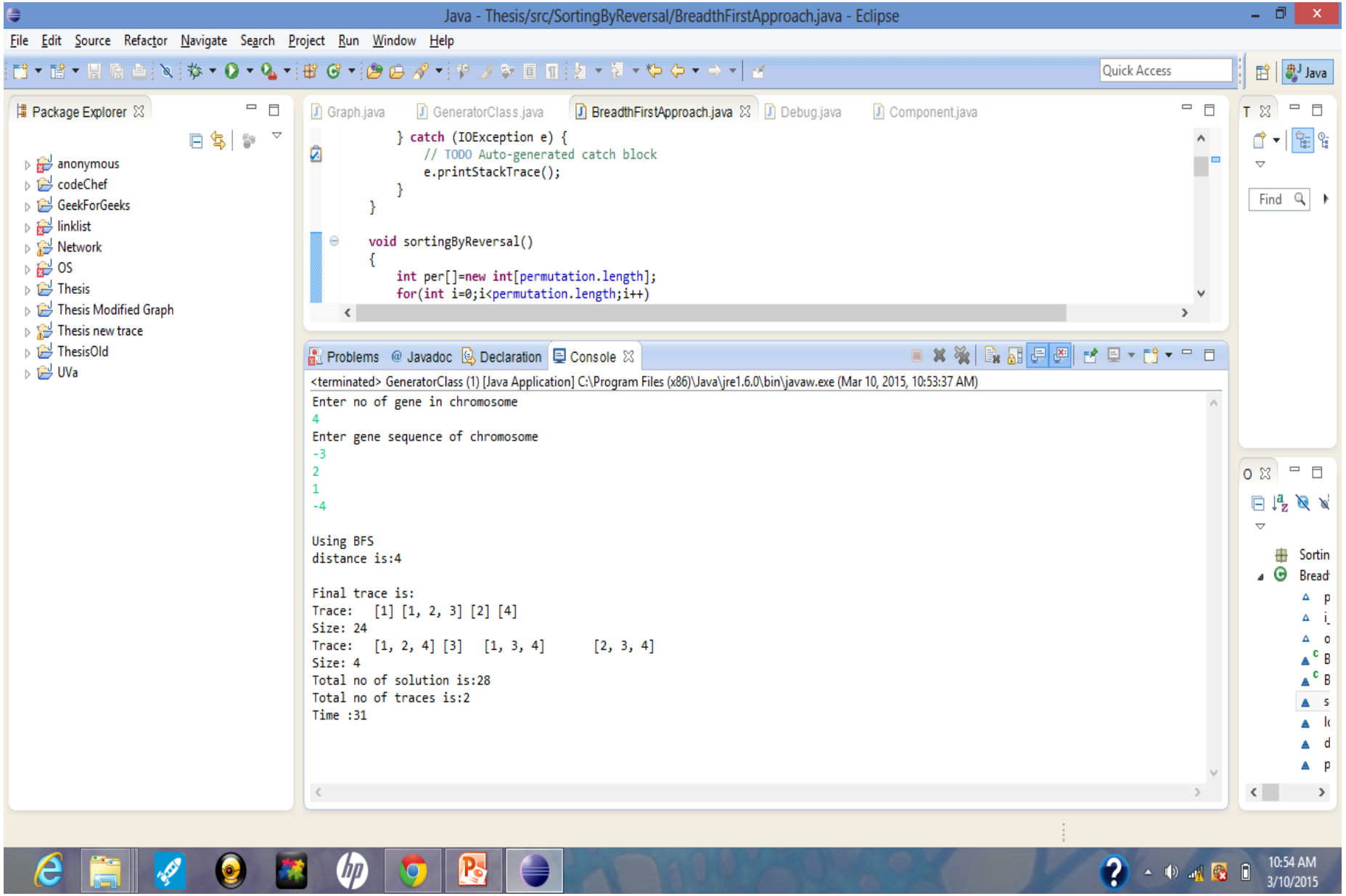
Examples with Hurdles and Fortress
(using DFA)

No.	Input permutation	d	Hurdle	Fortress	Traces	Time (sec)	Memory (Bytes)
1	2,1,3,-4,5,7,9,8,10,6, 11,13,14,12	12	Simple, protected nonhurdle	0	172	2.421	903704
2	2,7,3,5,4,6,8,13,9,11,10,12,14,1	13	Simple, Super, protected nonhurdle	0	1048	8.790	2893504
3	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

Testing

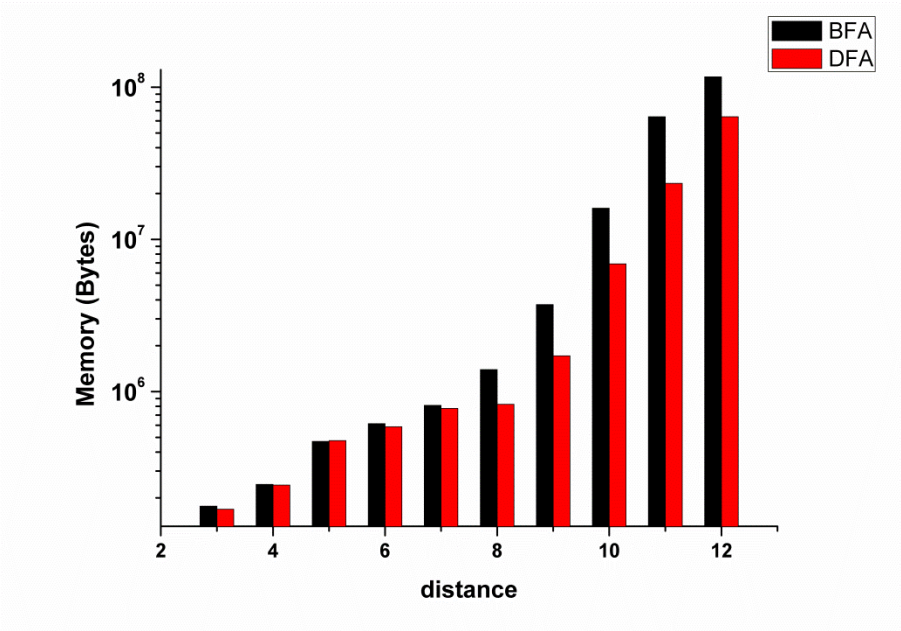
- Two types of data set of 20 permutations was generated, with n=2d and n=d.
- The permutations were generated by randomly selecting two cut points on an identity permutation of size n and performing reversal between them, then checking the distance increased by 1. In this way permutation of desired distance is generated.
- The results were obtained on these 20 permutations and later on their average was taken to obtain a generalized result.
- 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).

Implementation Snapshot

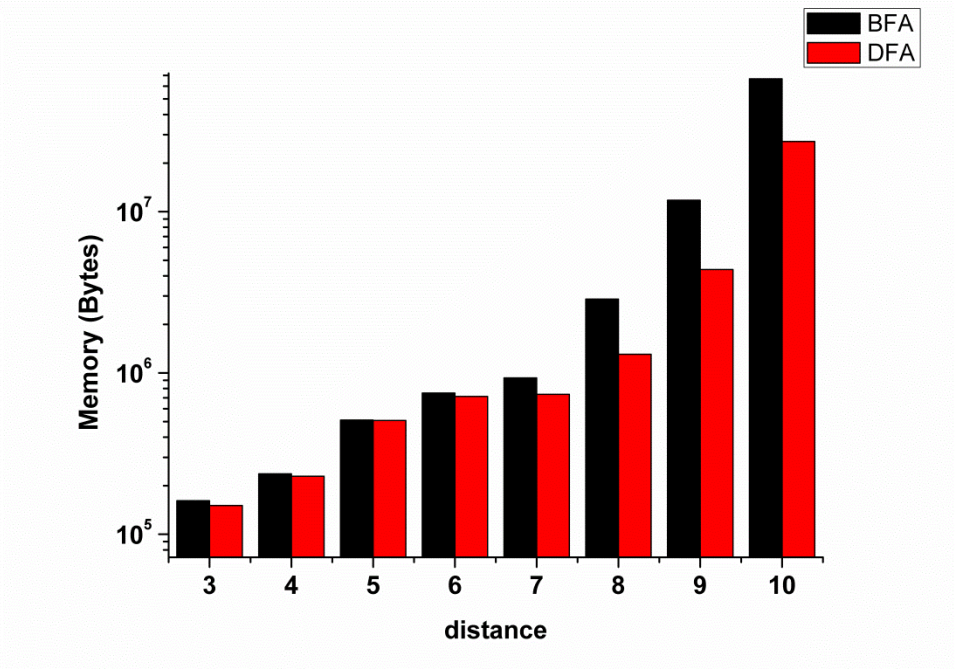


Results

Memory Analysis

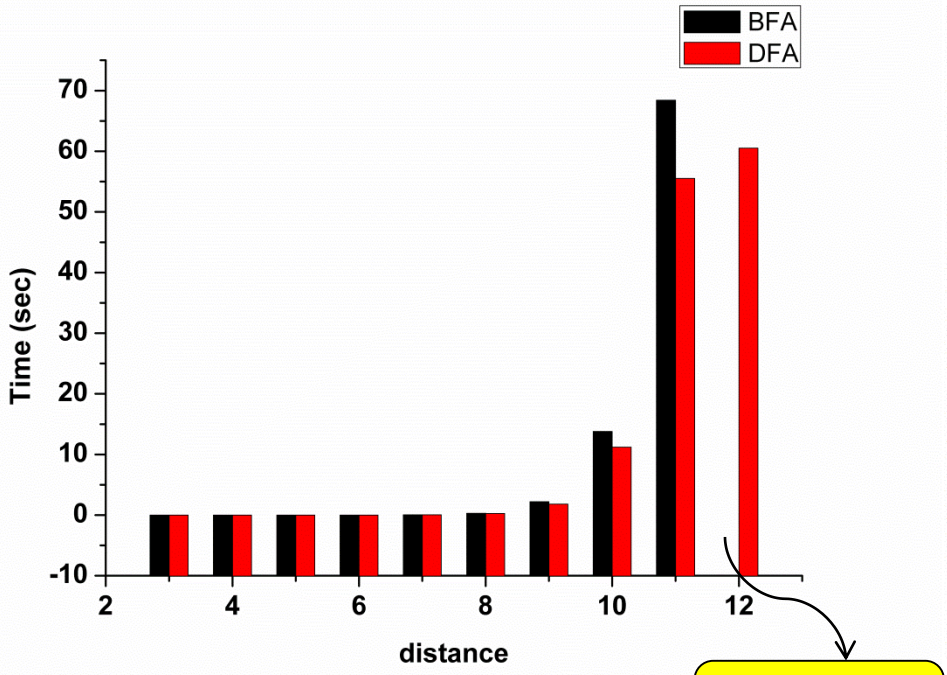


Memory used when n= 2d



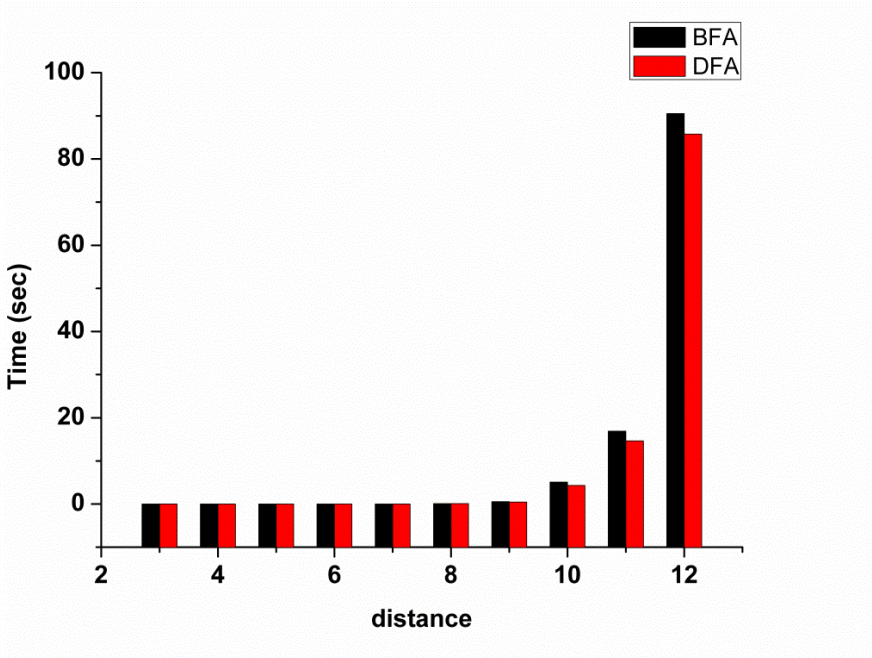
Memory used when n= d

Time Analysis



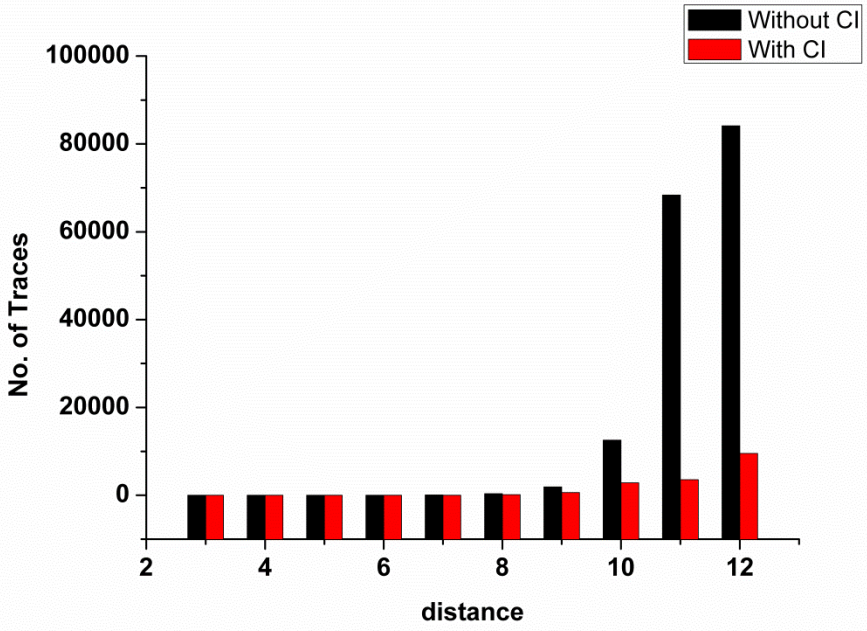
BFA fails

Time elapsed when n= d

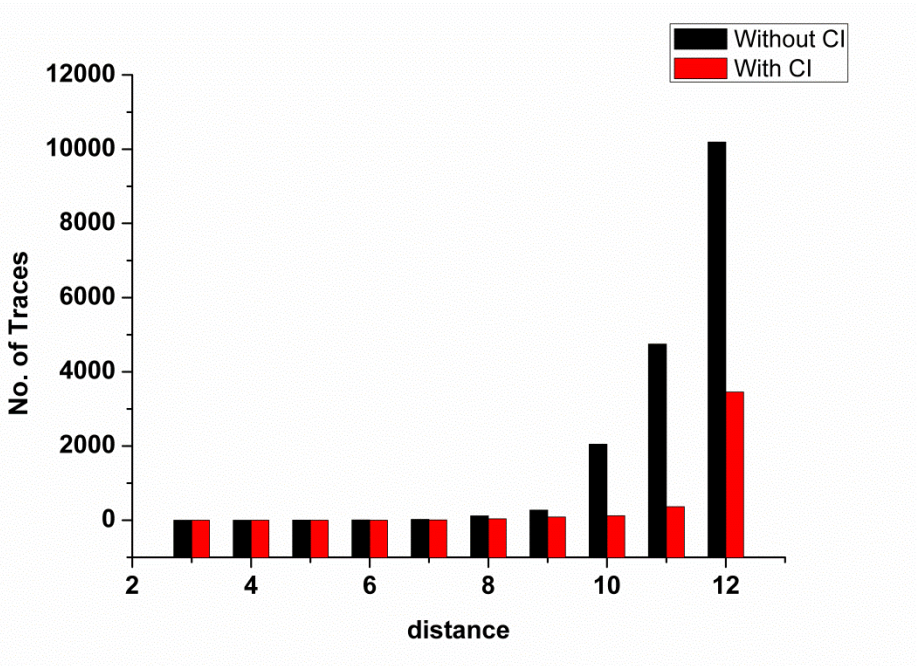


Time elapsed when n= 2d

Variation in number of Traces (using DFA)



Average Traces when n= 2d



Average Traces when n= d

Some Real Examples

N0.	Input permutation	Species	d	Hurdle	Sequences	Traces	Time	Memory
1	7,1,2,4,5,3,6,8	conserved blocks in cpDNA of tobacco and Lobelia fervens	6	Simple	1540	23	94	460672
2	4,6,1,7,2,3,5,8	conserved linkage groups in mouse and human X chromosomes	7	Simple	13088	108	143	628672
3	-4,-3,-2,1,-13,-15,14,-16 ,8,9,10,-11,12,5,6,7	Mouse and rat	9	No	776907	323	1021	1548328
4	-6,-5,4,13,14,-15,16,1,-3, 9,-10,11,12,-7,8,-2	Human and mouse	10	No	3362310	218	1254	873296
5	-13,-4,5,-6,-12,-8,-7,2, 1, -3,9,10, 11, 14,-15,16	Human and rat	10	No	2419750	418	1406	1719968

Conclusion

- The modified sorting by reversals algorithm produced more efficient solutions by discarding those branches that lead to the generation of redundant traces.
- DFA turns out to be much more memory efficient than BFA as after exploring one branch it frees the memory before proceeding to another.
- All possible solutions for any given sequence are listed irrespective of the presence of hurdle or fortress.
- The universe of sequences and class (solution space) have been drastically reduced.
- By logically filtering out less probable solutions, scenario more closer to reality is constructed (Heuristic based approach)

List of Publications

Papers Accepted:

Beethika Tripathi and Amritanjali, “An improved algorithm for reducing the solution space of sorting by reversals”, International Journal of Basic and Applied Biology, vol. 2(2), pp. 23-27, 2014.

Papers under review:

Beethika Tripathi and Amritanjali, “Enumerating All Plausible Traces of Reversals to Sort a Signed Permutation”, Conference on Computational Intelligence in Bioinformatics and Computational Biology, IEEE, 2015.

Future Directions

- The solution could be extended to other genomic rearrangement operations.
- If duplicate genes are present in the genome then it has to be removed listing all the possible sequences then working on it.
- Other different types of constraints could be applied depending upon the applications.
- Parallel version of the solution could be formed that could increase the speed of generation of solutions.
- The model can be made interactive so that flexibility can be given to the user to give preference to rearrangements in specific regions of chromosomes.
- The solution could be modified to deal with permutations of larger size by producing partial solutions at various stages and later on combining those solutions.

Application

- Study of various diseases occurring due to mutation in genomic sequence like progeria.
  - The stage of the disease can be predicted.
  - Study the mutations that lead to origin of a disease.
- Construction of phylogenetic tree helping in determination of evolution history.
- Reasons for various types of mutations can be predicted.

Relevant Bibliography

1. 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.
2. Berard S., Bergeron A., Chauve C. and Paul C. “Perfect sorting by reversals is not always difficult”, to appear in IEEE transactions on bioinformatics and computational biology, 2006.
3. 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.
4. Bergeron A., “A very elementary presentation of the Hannenhalli-Pevzner theory”, Discrete Applied Mathematics, vol. 146, 2005, pp. 134–145.
5. Tannier, E., Bergeron, A., Sagot, M.-F.: "Advances on sorting by reversals.", Disc. Appl. Math. (2007), 155(6-7), pp. 881–888.
6. 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

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. Braga, M.D.V.: Baobabluna: "The solution space of sorting by reversals.", Bioinformatics 25(14) (2009).
9. Braga, M.D.V., Sagot, M., Scornavacca, C., Tannier, E.: "The solution space of sorting by reversals", in: Mˇandoiu, I.I., Zelikovsky, A. (eds.) ISBRA 2007. LNCS (LNBI), vol. 4463, Springer, Heidelberg (2007), pp. 293–304.
10. 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.
11. 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).
12. T. Uno, M. Yagiura,: "Fast algorithms to enumerate All Common Intervals of two Permutations", Algorithmica 26(2), 2000, pp. 290-309.
13. "Fast algorithms to enumerate All Common Intervals of two Permutations", Algorithmica 26(2), 2000, pp. 290-309

Thank You

Theoretical Complexity

- Theoretical complexity for generating traces: $O(N. n^{kmax + 4})$
- n is number of elements in the sequence.
- N is total number of traces.
- The width (k) of a trace t is defined as the biggest subset of reversals of t such that every pair of reversals in this subset commutes.

{1, 2, 4}{1, 3, 4}{2, 3, 4}{3}

Subsets:

{ {1, 2, 4},{3} }, size = 2
{ {1, 3, 4},{3} }, size = 2
{ {2, 3, 4},{3} }, size = 2
k = 2

{1}{1, 2, 3}{2}{4}

Subsets:

{ {1},{1, 2, 3},{2},{4} }, size = 4
k = 4

kmax = 4

For π = (-3, 2, 1, -4), we have n = 4, N = 2 and kmax = 4

Representing Equivalence Class using Normal Form of Trace

Two reversals ρ and θ **commute** if they are **disjoint sets** or if **one is a subset of the other**.

{1,3,4} and {2,5} commute

{1,3,4} and {3} commute

{1,2,4} and {1,3,4} do not commute

ρ and θ **commute : ... ρθ ... is equivalent to ...** θρ **...**

{1, 2, 4} **{1, 3, 4} {3}** {2, 3, 4} is equivalent to

{1, 2, 4} **{3} {1, 3, 4}** {2, 3, 4}

›Commuting reversals when arranged in lexicographic order forms 1 sub-word of the normal form of trace.

›The other non-commuting 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**