Block Sorting Heuristics

A Project Report

submitted by

VENKETEP PRASAD M C

in partial fulfilment of the requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

May 2018

THESIS CERTIFICATE

This is to certify that the thesis titled **BLOCK SORTING HEURISTICS**, submitted by **VENKETEP PRASAD M C**, to the Indian Institute of Technology, Madras, for the award of the degree of **B.tech**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. N S Narayanaswamy
Research Guide
Professor
Dept. of CSE
IIT Madras, 600036

Place: Chennai

Date: 30th April 2018

ACKNOWLEDGEMENTS

Thanks to all those who made TEX and LATEX what it is today.

ABSTRACT

KEYWORDS: LaTeX; Thesis; Style files; Format.

A LaTeX class along with a simple template thesis are provided here. These can be used to easily write a thesis suitable for submission at IIT-Madras. The class provides options to format PhD, MS, M.Tech. and B.Tech. thesis. It also allows one to write a synopsis using the same class file. Also provided is a BIBTeX style file that formats all bibliography entries as per the IITM format.

The formatting is as (as far as the author is aware) per the current institute guidelines.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

IITM	Indian Institute of Technology, Madras
RTFM	Read the Fine Manual

NOTATION

r	Radius, m
α	Angle of thesis in degrees
β	Flight path in degrees

CHAPTER 1

INTRODUCTION

Mathmatics, Computer science, Informatics and Machine learning has lot of application in the field of Bio Informatics. Specific field like genome sequence matching has a lot of interesting problems for computer scientists. A Genome is a whole set of DNA. DNAs contain chromosomes which forms blocks of genome. Each chromosome contains a set of genes which are the fundamental reason for heredity. The order in which genes appear in a chromosome is responsible for its functionality, So different order implies different functionality. During molecular evolution there involves rearrangement of genes. Over the years these rearrangements are responsible for different behaviours in different species. There are some set of events which occur during a rearragement. Reversal, Translocation, Block or Strip moves are few such events.

A Chromosome is represented as a sequence of genes. Each gene is given a mapping in number space hence a chromosome with 10 genes might look like

4 10 8 6 5 3 1 2 7 9

Initially in molecular biology experts were interested in local mutations and hence concentrating on genes. Later they shifted to global level gene rearragements in a chromosome. If two species contains different chromosomes C_1 and C_2, but both of them constitute same set of genes, then the similarity between the two chromosomes is defined as minimum number of primitive steps from one chromosome to other. Whenever two chromosomes were compared, one of them will be taken as a base or identity permutation and the other chromosome will be altered using primitive steps to obtain identity permutation. The primitive steps are the possible rearrangements which are Reversals, Transpositions, Block moves. The minimum number of moves to reach identity permutaition is denoted as $D(\pi)$. This distance is a measure of similarity between the two permutations. Lesser distance implies greater similarity. Finding minimum event set resulting in an

identity permutation is NP-hard in case of Reversals and Block moves, whereas for transpositions the complexity remains open. For these hard problems we are in search of approximation algorithms which run in polynomial time.

1.1 Sorting by Reversal

An event in Reversal of a permutation $\pi = \pi_1\pi_2....\pi_n$ is to choose a substring and reverse it. So if we reverse the substring $\pi_i...\pi_j$, then after the reversal event, $\pi = \pi_1...\pi_{i-1}\pi_j....\pi_i\pi_{j+1}\pi_n$. For example, Consider permutation 1 8 2 3 **5 6 4** 7 9 10 $\rightarrow$ 1 8 2 3 4 **6 5** 7 9 10 $\rightarrow$ 1 8 **2 3 4 5 6 7** 9 10 $\rightarrow$ 1 **8 7 6 5 4 3 2** 9 10 $\rightarrow$ 1 2 3 4 5 6 7 8 9 10. In each event the substring highlighted is reversed. Hence the distance is 4.

1.2 Sorting by Transpositions

An event in Transposition of a permutation $\pi = \pi_1\pi_2....\pi_n$ is to choose two adjacent substring and swap them. Let say in an event we transpose substring $[\pi_i, \pi_{j-1}]$ and $[\pi_j, \pi_k]$, after the event permutation $\pi = \pi_1...\pi_{i-1}\pi_j....\pi_k\pi_i....\pi_{j-1}\pi_{k+1}\pi_n$. For example, Consider permutation 7 8 1 2 **4 5 6** *3* 9 $\rightarrow$ **7 8** *1 2 3 4 5 6* 9 $\rightarrow$ 1 2 3 4 5 6 7 8 9. In each event the substring highlighted is swapped with the substring in italic. Hence the distance is 3.

Table 1.1 shows the current status of different sorting primitives.

Table 1.1: Current status of various Sorting primitives

Primitive	**Complexity**	**Best Approximation**
Reversals	NP-hard	1.375
Transposition	Open	1.375
Block sort	NP-hard	2

1.3 Preliminaries and Definitions

Set of elements from $\{1, 2, ..., n-1, n\}$ can form different permutations. Our main aim is to sort the permutation, but with a restriction in the movements allowed. The problem of Block Sorting is to sort the given permutation with the **block moves**.

Block Sorting:
Given a permutation π, it can be written as $\pi_1\pi_2.....\pi_{n-1}\pi_n$ where each $\pi_i \in [1, n]$ $\forall i$. A **Block** is defined as a contiguous sequence in π such that elements are in the same order as in id_n, which is the Identity permutation. Now sorting a given π with only block moves is block sorting. Finding minimum number of moves to obtain id_n is a challenge which is proved to be NP-hard. The minimum number of moves is denoted as $bs(\pi)$.

Example:
Lets assume the permutation $\pi = 9, 6, 7, 8, 3, 4, 1, 2, 5$. There are 5 blocks: (9),(6,7,8),(3,4),(1,2) and (5). The optimal moves which give id_n is by moving block (3,4) between (1,2) and (5), hence obtaining 3 blocks (9),(6,7,8) and (1,2,3,4,5). Now by moving (9) to end of (6,7,8) followed by moving (6,7,8,9) to the end of (1,2,3,4,5) gives us id_n in 3 moves.

A **Kernel** of a permutation is a reduced form of π in which each block is represented as a single element which is the rank of that block in $ker(\pi)$.

Example:
Lets assume the permutation π = 9,6,7,8,3,4,1,2,5. There are 5 blocks: (9),(6,7,8),(3,4),(1,2) and (5). The rank of the blocks are respectively 5,4,2,1,3. So $ker(\pi)$ = 5,4,2,1,3.
It has been proved that $bs(ker(\pi)) = bs(\pi)$. The proof is simple that any move in π can be mimicked(equivalent move) in $ker(\pi)$ and vice-a-versa.

3 - approximation for block sorting: Since Block sorting is proved to be NP-hard, we will have to find approximate algorithm for it which runs in poly-time. Any sequence of block moves where each of the move is moving some block

to its predecessor is a 3 approximation for Block sorting.

Proof: For any block move described above the number of blocks reduce by minimum of 1 and maximum of 3. In $ker(\pi)$, the block moves looks as follows:

1. For 1, $(i),(j)...(y),(i+1),(l)$ to $(i,i+1),(j)...(y),(l)$ where $j \neq i+2$ and $y \neq l-1$
2. For 2, $(i),(i+2)...(y),(i+1),(l)$ to $(i,i+1,i+2)...(y),(l)$ where $y \neq l-1$
3. For 3, $(i),(i+2)...(y),(i+1),(y+1)$ to $(i,i+1,i+2)...(y,y+1)$

In id_n, there is only one block, So we have to reduce $Blocks(\pi)$ to 1. Since any block move can reduce it only 3, $bs(\pi) \geq \frac{Blocks(\pi)-1}{3}$. By following the procedure described each time we reduce $Blocks(\pi)$ by at least 1. Lets denote number of moves as described in the procedure as $bs-app(\pi)$ and $bs-app(\pi) \leq Blocks(\pi)-1 \leq 3bs(\pi)$. Hence the procedure described is in fact a 3 approximation for Block sorting.

$K-$Block Merging:

This problem is similar to block sorting but with more restriction in the moves allowed.

Any permutation π is represented as a partition of sets S = $\{S_1, S_2, ..., S_n\}$, where each set S_i is the maximum increasing sequence of the permutation left after S_{i-1} is formed. The block definition is same as the one in block sorting. But we can move a block only if it spans at most K non-empty sets. A block move is removing a block and inserting it in one of the sequences such that the sequence remains as an increasing one. The final aim is to minimum number of moves to get a single block which spans at most K non-empty sets. The minimum number of moves is represented as $bm(\pi)$.

Example of 2-Block Merging:

If we are given a permutation $\pi = \{4,5,12,2,3,6,7,8,1,9,10,11\}$, the set $S = \{(4,5,12),(2,3,6,7,8),(1,9,10,11)\}$.

1. Move the block (1) to (2,3,6,7,8) to obtain $S' = \{(4,5,12),(1,2,3,6,7,8),(9,10,11)\}$.
2. Now (6,7,8,9,10,11) is a block which spans 2 non-empty sets. So moving this to its predecessor gives $S'' = \{(4,5,6,7,8,9,10,11,12),(1,2,3)\}$.
3. Now moving (1,2,3) to its successor gives $S''' = \{(1,2,3,4,5,6,7,8,9,10,11,12)\}$

CHAPTER 2

Approximation algorithm for Block sorting

2.1 Approximation of Block sorting using $K-$ block merging

Theorem: $K-$Block merging approximates Block sorting by a factor of $(1+\frac{1}{K})$.

Proof:

The Idea of this proof is to make all the moves in optimal block sorting(call it P1) in $K-$Block merging(call it P2). If the block move made in P1 is possible in P2, then make the move in P2. Otherwise merge the blocks in P2 so that the block moved in P1 only spans at most K non-empty sequences in P2. Hence P2 has some extra moves because of the merging step. Now it is left to scrutinize the number of moves required.

Initially there are no blocks which span across at least 2 sequences, all block are within a sequence. If a block move is made from set S to obtain S'. Let us denote the number of blocks such that the block(A) is at end of a sequence and has successor block(C) such that C>A to its right in next sequence of set S by $L(S)$. For any Block move, $L(S') \leq L(S)+1$.

Example:

Consider $S = \{(4,5,12),(2,3,6,7,8),(1,9,10,11)\}$, which initially has L$(S) = 0$. By moving block B = (1), we get $S' = \{(4,5,12),(1,2,3,6,7,8),(9,10,11)\}$ with $L(S') = 1$ because for A = (6,7,8), C = (9,10,11) and C>A

$L(S') = L(S)+1$ only in the cases when the block B is moved in such a way that the block on left (from previous/same sequence call it A) and on right(in the same/previous sequence call it C) of the moved block satisfies C>A , hence the block A which was not counted in $L(S)$, will counted for $L(S')$.

So every K block moves can lead to a potential block spanning over $(K+1)$ sequences, which cannot be processed directly but requires a merging step. So irrespective of block moves, we add a buffer move for every K moves to do the merge

operation if required. So the number of moves happened in $K-$block merging is $bs(\pi) + \left\lfloor \frac{bs(\pi)}{K} \right\rfloor$. If we denote the number of moves in this particular method of $K-$block merging by $bma(\pi)$, $bm(\pi) \leq bma(\pi) \leq bs(\pi) + \left\lfloor \frac{bs(\pi)}{K} \right\rfloor \leq bs(\pi)(1+\frac{1}{K})$.

$$bs(\pi) \leq bm(\pi) \leq bs(\pi)(1 + \frac{1}{K})$$

2.2 2 Approximation algorithm for Block sorting

$K-$Block merging is proved to be NP hard for $K \geq 2$ by reducing $MAX - E3 - SAT$. But there is an algorithm which is optimal for $1-$Block Merging problem. So for $K = 1$ an exact algorithm which gives $(1 + \frac{1}{K}) = 2 -$ approximation for block sorting is obtained from a digraph constructed based on the set of sequences.

Construction of Digraph G:

Let the sequences $S = \{S_1, S_2..S_{n-1}, S_n\}$ correspond to π. V, the vertex set of G consists of all the elements in the permutation. The edge set E = $\{(u, v)|u < v$ and $\exists i \in [1, n], u \in S_i$ and $v \in S_i\}$.

Example:

Lets assume the permutation π = 9,6,7,8,3,4,1,2,5. The set

$S = \{(9), (6, 7, 8), (3, 4), (1, 2, 5)\}$

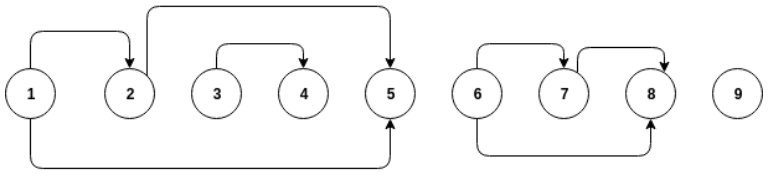

Figure 2.1: Two IITM logos in a row. This is also an illustration of a very long figure caption that wraps around two two lines. Notice that the caption is single-spaced.

Largest Non-crossing set: C(S)

2 edges (u, v) and (w, z) are said to cross each other if $u \leq w < v \leq z$ or $w \leq u < z \leq v$. An edge set E' is said to be non-crossing if no 2 edges in the set cross each other. The size of the maximal non-crossing edge set is denoted as

C(S). C(S) for the example is 5, the red edges shown below.

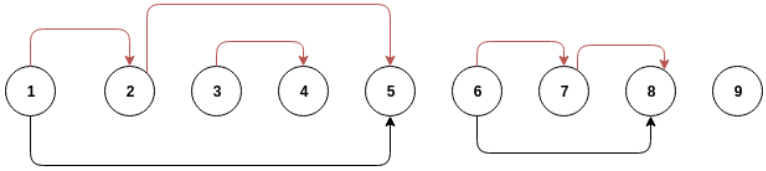

Figure 2.2: Two IITM logos in a row. This is also an illustration of a very long figure caption that wraps around two two lines. Notice that the caption is single-spaced.

There is a **Polynomial time algorithm using Dynamic Programming to find the C(S)**. Let $DP(i, j)$ represent the maximum non-crossing set for the sub-graph with the vertices $[i, j]$.

Base case:

1. $DP(i, i) = 0 \ \forall i$.
2. $DP(i, i+1) = 1$ if $(i, i+1) \in$ E
3. $DP(i, i+1) = 0$ if $(i, i+1) \notin$ E

$$DP(i,j) = \begin{cases} max(DP(i,j), DP(i,l) + DP(l,j)) & \forall l \in [i+1, j-1] \\ max(DP(i,j), DP(i+1,j-1)+1) & (i,j) \in E \\ max(DP(i,j), DP(i+1,j-1)) & (i,j) \notin E \end{cases}$$

The explanation for the construction of $DP(i, j)$ using the above procedure is as follows:

1. $max(DP(i, j), DP(i, l) + DP(l, j)) \ \forall \ l \in [i + 1, j - 1]$ deals with all the cases where the non-crossing edge sets are possible without using edge (i, j).
2. $max(DP(i, j), DP(i + 1, j - 1) + x)(i, j) \in E$ deals with the cases where edge (i, j) is present. x is 0 or 1 depending on whether $(i, j) \in$ E.

APPENDIX A

A SAMPLE APPENDIX

Just put in text as you would into any chapter with sections and whatnot. Thats the end of it.

REFERENCES

LIST OF PAPERS BASED ON THESIS

1. Authors.... Title... *Journal*, Volume, Page, (year).