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

I would like to express my gratitude to **Dr. N S Narayanaswamy** for guiding me throughout this project in both research and writing thesis.

I would also like to thank my parents and sister for constanly supporting me mentally and spiritually throughout the project and my life.

ABSTRACT

KEYWORDS: Block Sorting ; Block Merging; Approximation Algorithm; Heuristic.

Block sorting is a NP-hard problem which has applications in Biology specifically genome sequence matching. We need a P time solution, so we look for approximate solutions for the problem. Any greedy set of moves gives a 3-approximation. There is a 2-approximation for Block sorting using Block merging which is a relaxation to the original problem. Obtaining a better than 2 approximation has been open for a decade. In order to obtain a better approximation a new relaxation K-block merging was introduced whose optimal solution was proved to be $1 + \frac{1}{K}$ approximation. Finding optimal solution for K-block merging was proved to be NP-hard for $K \geq 2$. So in the process of extending Block merging algorithm to find an approximate algorithm for K-Block merging several heuristics were formed which performs better than block merging in many cases but there are tight example which proves it to be a 2 approximation. There is an exact algorithm for Block sorting with order graph, in which finding compatible set is NP-hard. So finding a suboptimal compatible set which performs better than block merging algorithm is the approach followed. Heuristics for finding a good suboptimal compatible set has been discussed in the following chapters.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOTATION

π	Permutation
$bm(\pi)$	Block Merging Distance for a permutation π
$bs(\pi)$	Block Sorting Distance for a permutation π
$ker(\pi)$	Kernel of a Permutation
$C(S)$	Cardinality of maximum non-crossing edge set
c_π	Cardinality of maximum compatible edge set

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 concentrated on genes. Later they shifted to global level gene rearrangements 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. Roy *et al.* (2008) has lower bounds and upper bounds for these primitives. There are approximation algorithms which run in polynomial time for these hard problems and a lot of research is happening in improving these approximation.

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. (Bafna and Pevzner, 1996) has a $\frac{7}{4}$ approximation algorithm for sorting by reversal.

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 APX-hard (Narayanaswamy and Roy, 2015). 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: Let $Blocks(\pi)$ represent number of block in π. 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 a block move can reduce $Blocks(\pi)$ by at most 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 $MAX - E3 - SAT$ (Narayanaswamy and Roy, 2015). 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 $S = \{(9), (6, 7, 8), (3, 4), (1, 2, 5)\}$. Fig 2.1 shows Digraph for the permutation.

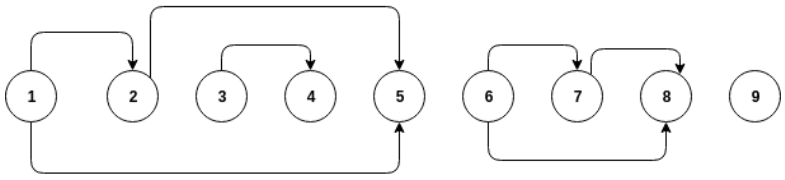

Figure 2.1: Digraph for the above example.

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 in Fig 2.2.

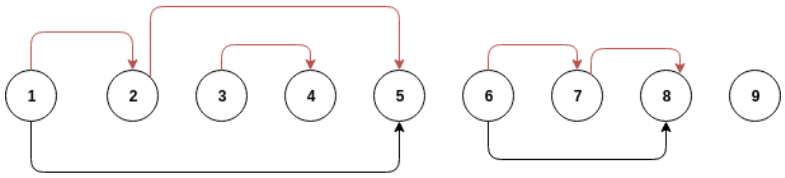

Figure 2.2: The highlighted red edges belong to the maximum non-crosssing edge set

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.

Obtaining optimal moves for 1-Block Merging

Obtaining optimal set of moves starts by finding $bm(\pi)$. Which follows as a result of following lemmas from (Mahajan *et al.*, 2006).

Lemma 2.1:

Any valid block move from S which results in S' follows that $C(S') \leq C(S) + 1$

Proof:

If the block moved is $[i, j]$, then from the source side(Sequence from which the block is moved) there will be a loss of 1 or 0 to $C(S)$ and the destination side(Where the block is moved to) will add at most 1 to $C(S)$. Let $NC(S)$ denote the non-crossing edge set for S.

1. Let the source side for block $[i, j]$ looks like $a, i, ..., j, b$. If $(a, i) \in NC(S)$ and $(j, b) \in NC(S)$, then $(a, b) \in NC(S')$ which make a total loss of 1 in source side. If any one of the edge is absent, then $(a, b) \notin NC(S')$ which also incurs a loss of 1. If both the edges (a, i) and (j, b) are absent, then the loss is 0.
2. In the destination side let the block be inserted between c, d. Then new edges added will be (c, i) and (j, d) is (c, d) is present, which adds 1 to $C(S)$. If one of the c or d is absent, only one edge will be added. So in the destination side there is a profit of maximum 1.

A maximum profit of 1 on destination side and maximum loss of 1 on source side gets us to the result that $C(S') \leq C(S) + 1$.

Lemma 2.2:

Given any sequence set S, there exist a block move which will result in S' such that $C(S') = C(S) + 1$

Proof:

A block is said be free if there is no edge of the form (i, j) such that only one of i or j belong to the block and the other does not. As we can see movement of a free block will induce 0 loss in source side and a profit of 1 in destination side, hence increase in $C(S)$ by 1. So it is left to prove that **we can find a free block in any S if it is not id_n.**

1. If all the edges are unit edges of the form $(i, i + 1)$ and still we have not reached id_n, then all the blocks are free blocks.
2. If there exist a non-unit edge, find a non-unit edge such that it doesn't contain any non-unit edge in it. Let that edge be (i, j). Since $(i, j) \in NC(S)$, $(i, i + 1) \notin NC(S)$ and $(j - 1, j) \notin NC(S)$. As $i + 1, ...j - 1$ contains only unit edge, blocks from $i + 1, ...j - 1$ is a free block.

Hence there exists a move for every S (which is not id_n) to S' such that $C(S')$ $= C(S) + 1$.

Lemma 2.3:

$bm(\pi) = n - 1 - C(S)$.

Proof:

We prove this by showing $bm(\pi) \leq n - 1 - C(S)$ and $bm(\pi) \geq n - 1 - C(S)$.

$bm(\pi) \geq n - 1 - C(S)$:

Because of the Lemma 2.1 every block move can only increase $C(S)$ by 1. In the optimal set of moves we start with $C(S)$ and we complete with n-1. Let $S_1, S_2, ..S_{bm(\pi)}$ be the sets obtained after each move in the optimal set of moves. $C(S_i) \leq C(S_{i-1}) + 1$, Hence $C(S_{bm(\pi)} \leq \mathrm{C}(S) + bm(\pi)$. Since $C(S_{bm(\pi)} = n - 1$, $C(S) \geq n - 1 - C(S)$.

$bm(\pi) \leq n - 1 - C(S)$:

The proof of this is by Induction on $r = n - 1 - C(S)$. For $r = 0$, we take id_n, so $bm(\pi) = 0$. Therefore $bm(\pi) \leq n - 1 - C(S)$. So lets assume that for $r = y$, $bm(\pi) \leq n - 1 - C(S)$. Now lets take a case of $C(S)$ such that $r = y + 1$. We know from Lemma 2.2 that there exist a block move which can result in $C(S)+1$. So we make that move hence obtaining π'. Now π' has $r = y$, so $bm(\pi') \leq n-1-(C(S)+1)$. Which is equivalent to $bm(\pi')+1 \leq n-1-C(S)$. $bm(\pi)$ is the minimum number of moves to sort π. So $bm(\pi) \leq bm(\pi') + 1 \leq n - 1 - C(S)$. So for $r = y+1$, $bm(\pi) \leq n-1-C(S)$. By induction hypothesis $bm(\pi) \leq n-1-C(S)$ for all r.

As the inequalities are in proved in both the directions, $bm(\pi) = n - 1 - C(S)$.

Now as we know that $bm(\pi) = n-1-C(S)$ and how to find a move which results in increase in $C(S)$, we can obtain id_n in $n - 1 - C(S)$ such moves which can be done in $O(n^4)$ time complexity. Now that we have an exact solution for 1-block merging it will serve as 2 approximation to block sorting.

2-Approximation is a tight bound for Block sorting using 1-Block merging:

To prove the above claim it is enough to prove for a generalised example for which the algorithm gives 2-approximation. The example we are considering is described as follows:

$$\pi(i) = \begin{cases} \frac{i+1}{2} & \text{if i is odd} \\ \frac{n+i}{2} & \text{i is even} \end{cases}$$

If the set of elements are from $[1, n]$, then the $\pi = \{1, \frac{n}{2} + 1, 2, \frac{n}{2} + 2, ..., \frac{n}{2}, n\}$.

Algorithm 1 Optimal Block Merging

```
1: procedure BLOCK MERGE(π,C)
2:     Input: π is a permutation and C is a maximum non-crossing edge set for π
3:     while π ≠ id_n do
4:         B ← Free block with respect to C
5:         Let u, v be the first and last element of the block respectively
6:         if B has a predecessor then
7:             Merge B with its predecessor
8:             if (u − 1, l) then
9:                 C = C\(u − 1, l) and C = C ∪ (v, l)
10:            C = C ∪ (u − 1, u)
11:        else
12:            Merge B with its successor
13:            if (l, v + 1) then
14:                C = C\(l, v + 1) and C = C ∪ (l, u)
15:            C = C ∪ (v, v + 1)
```

When we split this permutation into set of sequences, Then $S = \{(1, \frac{n}{2}+1), (2, \frac{n}{2}+2), (3, \frac{n}{2}+3)...(\frac{n}{2}, n)\}$. The above set S forms an input to 1-block merging. The Digraph for the above described sequence is shown in Fig 2.3 As we can see every

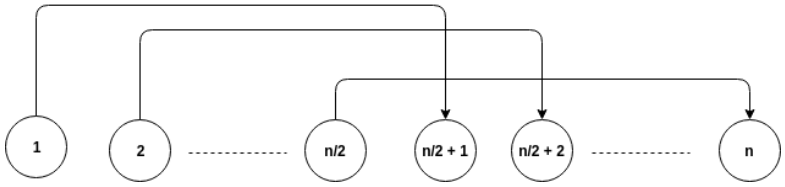

Figure 2.3: Digraph

edge crosses every other edge. So $C(S) = 1$, which means we can take only one edge in the non-crossing set. As we have proved before that the 2 - approximation algorithm takes exactly $n - 1 - C(S)$ $((i.e)\ n - 2)$ steps to complete the sorting procedure.

Now it is left to prove that the optimal procedure sorts the sequence in $\frac{n}{2} - 1$ steps. Without loss of generality assume that the non-crossing edge is $(\frac{n}{2}, n)$. So all the blocks from 1 to $\frac{n}{2} - 1$ are free and similarly all blocks from $\frac{n}{2} - 1$ to $n - 1$ are free. Lets move these blocks with one elements from $\frac{n}{2} + 1$ to $n - 1$, to their predecessors. this will increase $C(S)$ by one with each move finally resulting in $1, 2, ..., n$. Optimal block sorting algorithm sorts it in $\frac{n}{2} - 1$ steps, but the block merging algorithm takes n-2 steps. So this 2-approximation is tight.

The optimal block sorting moves are all valid moves even in 1-block merging,

which will result with $S = \{(1), (2), (3)...(\frac{n}{2} - 1), (\frac{n}{2}, ..., n-1, n)\}$. The remaining $\frac{n}{2} - 1$ steps are to make all the elements to be in a single sequence.

Lower Bound for *K*-block merging:

Lemma 2.1 proves that $C(S') \leq C(S) + 1$, for any valid block move in 1-block merging. So if we follow the similar analysis for $K-$block merging, the block moves are similar both in 1-block merging and $K-$block merging except that the block can span over K sequences.

Lemma 2.4:

For any valid block move in $K-$block merging which makes a move from S to S', then $C(S') \leq C(S) + k$.

Proof:

The changes in the non-crossing edge due to movement of block from source to destination are same as the one described for $1-$block merging. The additional power in $K-$block merging is that we can take a block spanning K sequences. So the block spanning K sets can be transferred to a single sequence, Due to these moves there can be additional $K - 1$ unit edges added. These unit edges are between the last element of previous sequence and first element of next sequence. There are $K - 1$ such pairs and each pair was not there initially because elements belonged to different sequences. So in addition to $C(S) + 1$ there can be $K - 1$ edges in $C(S')$. So $C(S') \leq C(S) + K$.

By Lemma 2.4, any valid move can only increases $C(S)$ by K. So to reach $n - 1$, we need at least $\frac{n-1-C(S)}{K}$ moves.

$$K - bm(\pi) \geq \frac{n - 1 - C(S)}{K}$$

2.3 Upper bound and Lower bound on block merging

Block merging approximates block sorting by a factor of 2, so $bs(\pi) \leq n - 1 - C(S) \leq 2bs(\pi)$. From the second inequality, $bs(\pi) \geq \frac{n-1-C(S)}{2}$, in addition to that

we know $k - bm(\pi) \geq bs(\pi)$ because all the moves that can be made in $K-$block merging can be made in block sorting. So combining the 2 inequalities,

$$K - bm(\pi) \geq \frac{n - 1 - C(S)}{2}$$

This bound proves to be a tighter lower bound when compared to the previous one.

Upper bound for $K-$Block merging with respect to $C(S)$ is $n - 1 - C(S)$ because for all block related sorting including block sorting take $n-1-C(S)$ moves to sort a reverse permutation.

$$K - bm(\pi) \leq n - 1 - C(S)$$

CHAPTER 3

Optimal Block Sorting

The algorithm construction for optimal block sorting begins with a graph construction similar to the graph for block merging. Since a block can be picked only within a sequence in block merging, we added edges within a sequence. Now with block sorting, we can move any block to any place.

Order Graph:

Order graph of a permutation π is a graph with N vertices where each vertex is an element in permutation. Edge set $E_\pi = \{(u,v)|(u<v) \wedge (\pi_u^{-1} < \pi_v^{-1})\}$.

1. Two edges (u,v) and (w,x) are said to interleave if they either interleave by value $u \leq w < v \leq x$ or they interleave by position $\pi_u^{-1} \leq \pi_w^{-1} < \pi_v^{-1} \leq \pi_x^{-1}$
2. An edge (u,v) is said to contain (w,x), if it contains either by value $u < w < x < v$ or by position $\pi_u^{-1} < \pi_w^{-1} < \pi_x^{-1} < \pi_v^{-1}$.
3. Any $C \subseteq E_\pi$, inclusion graph $G(C,\pi) = \{((u,v),(w,x))|$ if (u,v) contains $(w,x)\}$
4. $C \subseteq E_\pi$ is said to be a compatible set if no two edges in C interleave each other and $G(C,\pi)$ is acyclic.
5. c_π is size of maximum cardinality compatible set.

Example:

Consider π = 2 4 3 5 1 6, Fig 3.1 shows the order graph for π, Fig 3.2 Highlights compatible edge set and shows $G(C,\pi)$ is acyclic.

Lemma 3.1:

Every compatible edge set C contains all unit edges from E_π.(Mahajan *et al.*, 2008)

Lemma 3.2:

Let α be a free block with respect to C. For any valid block move using α from a permutation π to obtain σ, then $c_\sigma \geq c_\pi$.

Proof:

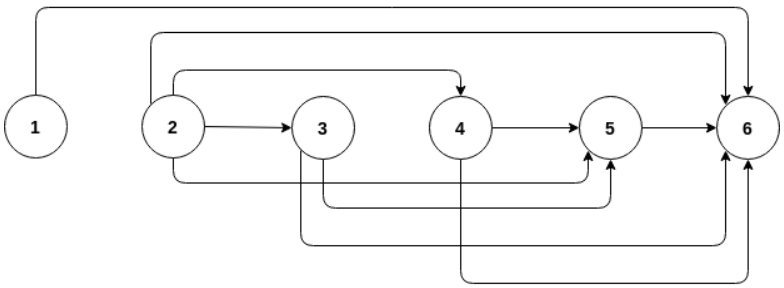

Figure 3.1: Order Graph for the permutation

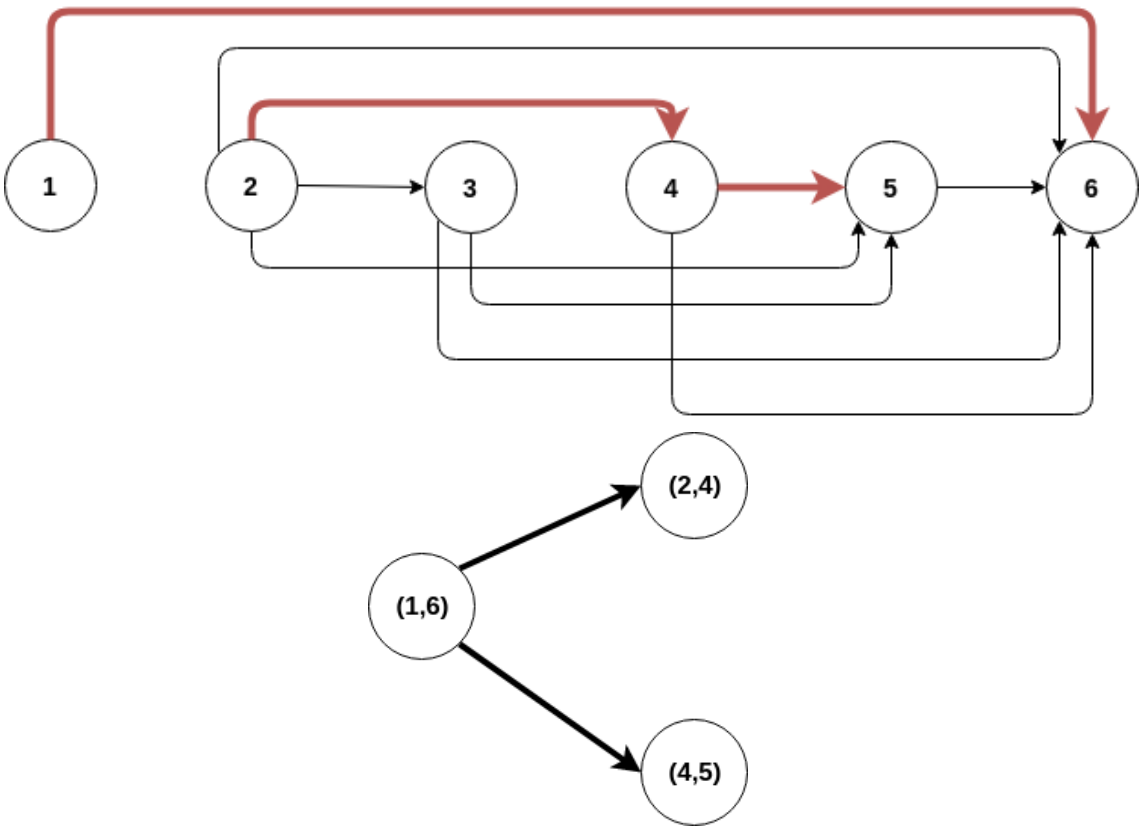

Figure 3.2: The highlighted red edges belong to the maximum compatible edge set and $G(C, \pi)$ is acyclic

The definition of free block is a block which has all edges originate and end within the block (same as the definition for block merging). Moving a block will not affect anything with respect to value. Consider all the edges in C. In σ the only change is the block position. A block has full of unit edges in it. Relation between all other edges remain the same. Now consider all the edges from C in σ, the only change is unit edges corresponding to α. None of the edges in C interleave by position because they are unit edges. Consider $G(C, \sigma)$, new edges added on $G(C, \pi)$ are the edges that can contain unit edges of α (certain edges were also deleted). Since no edge can leave from $(u, u+1)$ in $G(C, \sigma)$, the added edges cannot contribute to cycle. Hence showing $G(C, \sigma)$ is acyclic, so C is a compatible set in π, so $c_\sigma \geq c_\pi$. If the block is moved to its predecessor or successor then one more unit edge will be added so $c_\sigma > c_\pi$

Lemma 3.3:

For any valid block move from π to σ, $c_\sigma \geq c_\pi - 1$

Proof:

For the case of free block move, Lemma 3.2 is the proof. Let m denote the number of edges incident or originated from block B in π.

Case $m = 1$:

Let (u, v) be the edge that is incident on the block B. Consider $C' = C \backslash (u, v)$. C' is a compatible set for σ because of the same reasons given in Lemma 3.2.

Case $m = 2$:

Let $B = \{v, v+1, v+2, ..., w\}$ and let the 2 edges touching B be (u, v) and (w, x). Consider $C' = C \backslash ((u, v) \cup (w, x))$, $C'' = C' \cup (u, x)$. The remaining of the proof is to prove that C'' is a compatible set in σ. The newly added edge (u, x) is not interleaved by value. If there were any edge interleaving (u, x), the same edge might have interleaved one of (u, v), unit edges in B and (w, x). So one edge in C' interleaves (u, x) by value. The relative position of all the elements remain the same except for the edges of B, which was moved. So if (u, x) had to be interleaved by position it should be with one of the edges in B. That is not possible because they are all unit edges. So (u, x) is not interleaved by any other edges in C''. It is left to prove that $G(C'', \sigma)$ is acyclic.

There is no cycle in $G(C'', \sigma)$. 2 edges were deleted and an edge (u, x) was added.

Only addition of edge (u, x) could possibly create a cycle. Assume this addition creates a cycle. In the cycle, let $((a, b), (u, x), (c, d)$ be a path. So (a, b) contains (u, x) and (u, x) contains (c, d). If (u, x) contain (c, d) then one among (u, v), (w, x) contain (c, d) lets call it edge q. (a, b) contains (u, x), so (a, b) contains q. So now we have found a cycle without (u, x) where the path in cycle is $((a, b), q, (c, d)$ instead of $((a, b), (u, x), (c, d)$. So this proves there is a cycle in $G(C, \pi)$ if there is a cycle in $G(C'', \sigma)$. Since $G(C, \pi)$ is acyclic, $G(C'', \sigma)$ is also acyclic. By showing $G(C'', \sigma)$ is acyclic, we prove C'' is a compatible edge set. $|C''| = |C| - 1$. Therefore $c_\sigma \geq c_\pi - 1$.

Lemma 3.4:

For any valid block move from σ to obtain π, $c_\pi \leq c_\sigma + 1$

Proof:

This is straightforward proof from Lemma 3.3, just revert the block move from σ to obtain π and the result follows.

Lemma 3.5:

We can always find a free block with respect to C unless $\pi \neq id_n$.

Proof:

If C has full of unit edges then all the blocks are free. So we can take any block. But if C has at least one non-unit edge, then consider the graph $G(C', \pi)$, where C' is C without unit edges. Find an edge in $G(C', \pi)$ which does not have an out edge, lets call it (u, v). (u, v) does not contain non-unit edge of C. If $i = \pi_u^{-1}$, $j = \pi_v^{-1}$. So all the edges between the vertices of set $\{u + 1, ..., v - 1\}$, because all those edges are contained by (u, v). Similarly for all the edges between the vertices of set $\{\pi_{i+1}, \pi_{i+2}, ...\pi_{j-2}, \pi_{j-1}\}$. Since all the edges in these 2 sets are unit edges, any block in this set is a free block. Moreover $\{u + 1, ..., v - 1\}$ is a non-empty set as (u, v) is not an unit edge. So we can definitely find a free block with respect to C.

Theorem:

For all π, $bs(\pi) = n - 1 - c_\pi$.

Proof:

Using Lemma 3.4 we know that $c_\pi \leq c_\sigma + 1$ for any block move from π to σ. From

Lemma 3.5 we know that we can always find one such move to increase c_π by 1. If $m_1, ..., m_x$ be the x moves done in total to obtain id_n where i^{th} move increases the c_π^i by 1 ($c_\pi^0 = c_\pi$).

$$c_\pi + x = n - 1$$

$$x = n - 1 - c_\pi$$

.

Algorithm 2 Optimal Block Sorting

```
1:  procedure BLOCK SORT(π,C)
2:      Input: π is a permutation and C is a Maximum Compatible edge set for π
3:      while π ≠ id_n do
4:          B ← Free block with respect to C
5:          Let u, v be the first and last element of the block respectively
6:          if B has a successor then
7:              Merge B with its successor
8:              ∀(l, v+1) ∈ C, C = C ∪ (l, v) and C = C\(l, v+1)
9:              C = C ∪ (v, v+1).
10:         else
11:             Merge B with its predecessor
12:             C = C ∪ (u−1, u)
```

Though we have an algorithm which executes in polynomial time, we cannot find maximum compatible set in polynomial time.

CHAPTER 4

Block sorting heuristics

First we list two heuristics from (Mahajan *et al.*, 2008). For each of the heuristics a tight example for 2-approximation is also given in it.

In the proof for tightness of 2 approximation block sorting algorithm, we have said that the additional moves that we require for block merging are redundant moves.

Heuristic 1: Run Block merging algorithm and remove redundant moves.

Heuristic 2: The heuristic function for a permutation π is $bm(\pi)$. A block sorting move for a permutation π is made with the help of this heuristic function. For a block move b on π which results in a permutation π^b, we calculate $bm(\pi^b)$ which is its value. We choose a block b which has its value to be minimum.

This heuristic performs so well on many cases that it gives optimal solution for them, but there exist an example for which it is a 2-approximation. The identified problem with this heuristic is the tie breaking method. During a tie few locally bad moves are actually globally optimal. The problem with many such heuristic is the tie breaking mechanism. **A possible solution is to increase of depth of analysis from 1, So that local optimals can be avoided.**

4.1 Approximation algorithms using compatible set

Finding optimal compatible set for a given edge set in an order graph is hard, but when the edge set contains edges with a property that interleaving by position implies interleaving by value, then we can find a optimal substructure for the problem, hence we can find optimal solution using Dynamic Programming. So we can construct approximation algorithms if we could find a good compatible set with an edge set that satisfies the above property. Block merging is one such approximation for finding a compatible set. The edges of a digraph is only within a sequence which is an increasing sequence and edges between two sequences do not cross each other. If edge (a, b) and (c, d) interleaves by position

then $\pi_a^{-1} \leq \pi_c^{-1} < \pi_b^{-1} \leq \pi_d^{-1}$. Because (a, b) and (c, d) are part of increasing sequence, $a \leq c < b \leq d$.

So our aim is to find a better compatible edge set when compared to maximum non-crossing edge set by adding few more edges which also satisfies the property that if two edges interleave by position then they interleave by value. By doing this we can possibly increase c_π and hence we can get a better approximation than 2. The approach is to find an increasing sequence that do not violate the property with the edges in digraph.

Heuristic 3: Consider the digraph constructed for Block merging, we are going to find a maximum increasing subsequence consisting of the first and last elements of each sequence. In this sequence for all $i < j$ add edge (i, j) to the edge set.

Example:

Consider $\pi = \{1, 5, 2, 6, 3, 7, 9, 4, 8\}$. Corresponding set $S = \{(1,5),(2,6),(3,7,9),(4,8)\}$. The edges in the digraph are $E = \{(1, 5), (2, 6), (3, 7), (3, 9), (7, 9), (4, 8)\}$. First and last elements sequence are $A = \{1, 5, 2, 6, 3, 9, 4, 8\}$. So we find maximum increasing subsequence of A which is $\{1, 2, 3, 4, 8\}$. New edge set $E' = E \cup \{(1,2),(1,3),(1,4),(1,8),(2,3),(2,4),(2,8),(3,4),(3,8),(4,8)\}$.

Lemma 4.1:

Adding these edges will not violate the property that interleaving by position implies interleaving by value.

Proof:

Let initial edges in the digraph be denoted as E and the new edges added be denoted as N. Edges within N which interleave each other satify the property because they are part of increasing sequence, same for edges within E. Edges in N consists only of boundary points in the sequence. If an edge from E cross an edge from N by position, then the edge from E must have either a starting point or end point to be a part of the edge from N, Because an edge from E cannot cross a boundary. When interleaving by position implies an intersection of vertices, this in fact implies interleaving by value.

Heuristic 4: This is an improvement to Heuristic 3. Let the set be $A = \{e_1, e_2, ..., e_m\}$ in which the maximum increasing sequence is to be found. Let the maximum increasing sequence found be $M = \{e_{i1}, e_{i2}, ..., e_{in}\}$. After adding edges, the elements of A between two consecutive elements of M do not have any

additional edges. So lets recurse problem with $n-1$ problems A_{i1} to $A_{i(n-1)}$ where $A_{ij} = [e_{ij}, e_{i(j+1)}]$ $(i.e)$ elements of A in the range $[e_{ij}, e_{i(j+1)}]$. Recurse until the subproblem contains one or two points.

Lemma 4.2:

Edges added during recursion will satisfy interleaving by position implies interleaving by value

Proof:

Lets build a parent children relation based on the subproblems created. If a subproblem F is created with the maximum increasing sequence of another subproblem G, then G is parent to F. The claim is that edges added by F do not cross G, This is because the subproblem F is between 2 consecutive elements of the increasing sequence in G. So all the edges are contained by G. Since none of the child edges interleave parent edges, it is left to prove that none of the added edges interleave the edges of digraph. This can be proved by the same argument used for Lemma 4.1. Since all the edges added have endpoints to be start and end of sequences, interleaving by position will imply having a common point which implies interleaving by value.

FUTURE WORK: Empirical analysis of the heuristics

REFERENCES

1. **Bafna, V.** and **P. A. Pevzner** (1996). Genome rearrangements and sorting by reversals. *SIAM Journal on Computing*, **25**(2), 272–289.

2. **Mahajan, M.**, **R. Rama**, **V. Raman**, and **S. Vijaykumar** (2006). Approximate block sorting. *International Journal of Foundations of Computer Science*, **17**(02), 337–355.

3. **Mahajan, M.**, **R. Rama**, and **S. Vijayakumar**, Block sorting: A characterization and some heuristics. *In Nordic Journal of Computing*. 2008.

4. **Narayanaswamy, N.** and **S. Roy**, Block sorting is apx-hard. *In International Conference on Algorithms and Complexity*. Springer, 2015.

5. **Roy, S.**, **M. Rahman**, and **A. K. Thakur** (2008). Sorting primitives and genome rearrangement in bioinformatics: A unified perspective. *World Academy of Science, Engineering and Technology*, **38**, 363–368.