UNF Digital Commons

UNF Graduate Theses and Dissertations

Student Scholarship

2015

Sorting by Block Moves

Jici Huang
University of North Florida

SORTING BY BLOCK MOVES

by

Jici Huang

A thesis submitted to the
School of Computing
in partial fulfilment of the requirements for the degree of

Master of Science in Computer and Information Sciences

UNIVERSITY OF NORTH FLORIDA
SCHOOL OF COMPUTING

June, 2015

The thesis “Sorting By Block Moves” submitted by Jici Huang in partial fulfillment of the requirements for the degree of Master of Science in Computer and Information Sciences has been

Approved by the thesis committee: Date

______________________________ ________________

Dr. Swapnoneel Roy
Thesis Advisor and Committee Chairperson

______________________________ ________________

Dr. Ching-Hua Chuan

______________________________ ________________

Dr. William F. Klostermeyer

Accepted for the School of Computing:

______________________________ ________________

Dr. Asai Asaithambi
Director of the School

Accepted for the College of Computing, Engineering, and Construction:

______________________________ ________________

Dr. Mark A. Tumeo
Dean of the College

Accepted for the University:

______________________________ ________________

Dr. John Kantner
Dean of the Graduate School

ACKNOWLEGEMENT

I would like to express my sincerest gratitude to my thesis advisor, Dr. Swapnoneel Roy, who has provided me with the patient guidance, encouragement and advice throughout the duration of the research and development of this thesis. He helped me get started with LaTeX, and provided an experienced ear for my doubts about writing a thesis. Without him, this thesis would not have been completed or written. I would also like to thank my committee members, Dr. Ching-Hua Chuan and Dr. William F. Klostermeyer, as well as Dr. Asai Asaithambi for their remarkable comments and suggestions during my thesis prospectus presentation. Furthermore, I want to express my sincere appreciation to Dr. Roger Eggen for his academic support and advice during my graduate studies. I also want to thank Mr. Jim Littleton, Mr. Larry Snedden and Ms. Shawn Broderick for their help and encouragement.

Last but not the least, I thank my sister, brother, mother, and father for supporting me throughout my studies at the University of North Florida.

CONTENTS

List of Figures . . . . **vii**

Abstract . . . . **viii**

Chapter 1 Introduction **1**

1.1 The Block Sorting Problem 1

1.2 Scope of Present Work 2

1.2.1 Related Previous Work 2

1.2.2 Contributions of the Thesis 3

1.3 Motivations and Applications of Block Sorting 3

1.3.1 Computational Biology 3

1.3.2 Optical Character Recognition (OCR) 8

Chapter 2 Background **10**

2.1 Block Sorting 10

2.1.1 Lower Bounds for Block Sorting 10

2.1.2 A 3-Approximation Algorithm for Block Sorting 11

2.1.3 Block Merging 11

2.1.4 Block Deletion 15

2.2 Sorting by Transpositions 18

Chapter 3 New Approximation Algorithms for Block Sorting **21**

3.1 Preliminaries 21

3.2 Run Merging Algorithm 24

3.3 Ordered Pair Fixing 26

3.4 Significance of Result 29

Chapter 4 Concluding Remarks **30**

4.1 Summary of Contributions 30

4.2 Future Directions 30

Publication **32**

References **33**

Vita **36**

FIGURES

Figure 1.1 Genome Rearrangement 4

Figure 2.1 Block Sorting Moves 13

Figure 2.2 Block Merging Moves 13

Figure 2.3 The Block Sorting Moves 17

Figure 2.4 Relative Block Deletions 17

Figure 2.5 Transpositions in Sorting by Transposition 19

Figure 3.1 Blocks and the Corresponding Identity Permutation 21

Figure 3.2 An Example of a Block Move 22

Figure 3.3 An Example of a Block Sorting Schedule 23

ABSTRACT

The research in this thesis is focused on the problem of Block Sorting, which has applications in Computational Biology and in Optical Character Recognition (OCR). A *block* in a permutation is a maximal sequence of consecutive elements that are also consecutive in the *identity* permutation. BLOCK SORTING is the process of transforming an arbitrary permutation to the identity permutation through a sequence of block moves. Given an arbitrary permutation π and an integer m, the Block Sorting Problem, or the problem of deciding whether the transformation can be accomplished in at most m block moves has been shown to be NP-hard. After being known to be 3-approximable for over a decade, block sorting has been researched extensively and now there are several 2-approximation algorithms for its solution. This work introduces new structures on a permutation, which are called *runs* and *ordered pairs*, and are used to develop two new approximation algorithms. Both the new algorithms are 2-approximation algorithms, yielding the approximation ratio equal to the current best. This work also includes an analysis of both the new algorithms showing they are 2-approximation algorithms.

Chapter 1

INTRODUCTION

1.1 The Block Sorting Problem

An n-permutation is an arrangement of natural numbers from 1 through n. The n-permutation in which the number j occupies position j for $j = 1, 2, \cdots, n$ is called the *identity* permutation. In this thesis, an arbitrary permutation will denoted by a lower case Greek letter, and is referred to the identity permutation as *id*.

A *block* in a permutation π is a maximal sequence of consecutive elements that are also consecutive in the identity or sorted permutation id. Block sorting is the process of transforming an arbitrary permutation to the identity permutation through a sequence of block moves. An arbitrary permutation refers to a randomly-picked permutation from a set of permutations. $bs(\pi)$ is denoted as the least number of block moves needed to transform π to the identity permutation. The Block Sorting problem may be formally stated as:

Given an arbitrary n-permutation π and an integer m, is $bs(\pi) \leq m$?

The above statement is known as the decision version of the Block Sorting problem and it has been shown to be NP-hard (Bein et al., 2002). NP-hard is the abbreviation of Non-deterministic Polynomial-time hard. Being NP-hard means not only that no known efficient algorithms is available for solving the problem, but also that it is quite unlikely that the algorithm exists (Hochbaum, 1997). Thus, approximation algorithms have been designed to approximate the block sorting problem.

Block sorting has been known to be 3-approximable (Bein et al., 2002) for over a decade now, which means there exists a 3-approximation algorithm for block sorting, and currently there are several 2-approximation algorithms (Mahajan et al., 2006; Bein et al., 2005) for solving this problem. The design of algorithms with approximation ratio less than 2 still remains an open question.

1.2 Scope of Present Work

1.2.1 Related Previous Work

Sorting methods like block sorting have gained much attention in the past decade mainly because of their applications in the study of genome rearrangements in computational molecular biology and in optical character recognition. Most of the problems in these application domains have either been proven to be computationally hard, or questions related to their computational complexity still remain open. Hence, a majority of the algorithms designed for these problems are approximation algorithms. Some of the well-studied primitives used for sorting suitable for these problems are Transpositions (Bulteau et al., 2011; Elias and Hartman, 2006; Christie and Irving, 2001; Firoz et al., 2010), Reversals (Caprara, 1997; Bafna and Pevzner, 1996; Hannenhalli and Pevzner, 1999), Prefix Reversals (Bulteau et al., 2012; Gates and Papadimitriou, 1979), and Block Interchanges (Lin et al., 2005; Christie, 1996; Mira and Meidanis, 2007). After having a 3-approximation algorithm, block sorting has been researched intensively as well, leading to several 2-approximation algorithms.

1.2.2 Contributions of the Thesis

This work introduces new structures on a permutation, which are called *runs* and *ordered pairs*, and uses them to develop two new approximation algorithms. While the algorithms are still 2-approximation algorithms, they run in $O(n)$, (linear) time in the number of block moves performed as compared to the $O(n^2)$ (quadratic) time of the algorithm of (Bein et al., 2005), and the $O(n^3)$ (cubic) time of the algorithm of (Mahajan et al., 2006). Moreover, the algorithms are much simpler than Block Merging and Block Deletion approximation algorithms to implement and analyze. This work also includes an analysis of both algorithms showing they are 2-approximation algorithms.

It is proven that moving all the blocks of π except the blocks of the longest run would yield a 2-approximation algorithm which is presented as the first algorithm. It is further proven that keeping 2 blocks corresponding to any ordered pair in π intact (and moving other blocks) will be sufficient to yield another 2-approximation, which is presented as the second algorithm.

1.3 Motivations and Applications of Block Sorting

1.3.1 Computational Biology

As mentioned previously, the *genome rearrangement* problem, frequently studied in computational molecular biology, lends itself naturally to resemble block sorting, which is the focus of this thesis.

A few decades ago, biologists learned how to analyze DNA, where genes reside. The important techniques of analyzing DNA include copying, cutting, pasting, measuring, and probing DNA. Genomic sequence comparison is a very powerful tool to discover the evolutionary procedures and fundamental relationships between genes (Gu et al., 1999).

The term genome is used to represent the entire DNA of a living organism. A gene is a segment of the DNA that is involved, in producing a protein, for instance, and its orientation depends on the DNA-strand that it lies on.

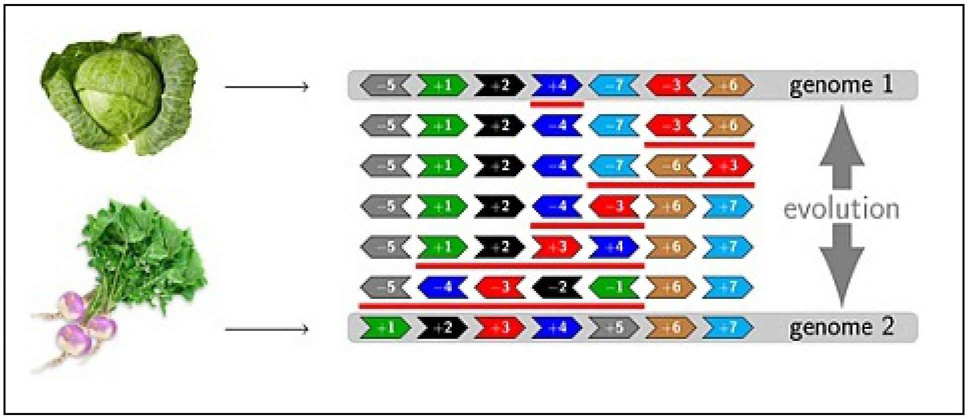

Figure 1.1: Genome Rearrangement

Genome rearrangement or rearrangement of syntenic blocks has become a topic of intensive study by phylogenists, comparative geneticists, and computational biologists. The rearrangements of mitochondrial genomes of cabbages and turnips are studied by Palmer and Herbon (Palmer and Herbon, 1988). Their study indicates that the two species are closely related to each other genetically (See Figure 1.1), as many genes of the two species are 99-99.9 percent identical (Palmer and Herbon, 1988). However, the orders in which the genes appear in the two species are different. The study suggests that the evolutionary path between genomes can be traced by using global rearrangement events, such as reversals and transpositions of genome segments (Hartman and Shamir, 2006). A transposition is an operation to rearrange the permutation by cutting a segment out of the permutation and pasting it at another location (Gu et al., 1999).

Thus, genome rearrangements have been modeled by a variety of sorting primitives, such as reversals, transpositions, block moves and block interchanges. Genome rearrangement is equivalent to sorting a permutation under a single or a combination of such primitives.

The methodology consists of using a single or a combination of well-defined primitives to transform one genome into the other. The number of primitive steps needed to transform one genome into another is a measure for the evolutionary distance between the two species. Genome rearrangement can be formally stated as:

> Given genomes G_1, G_2, and a set S of primitives, what is the shortest sequence of elements of S that will be needed to transform G_1 into G_2?

A related, simpler problem is to compute the evolutionary distance, $d_S(G_1, G_2)$, i.e. just the length of the shortest sequence. There are many variants of the above problems, depending on how genomes are modelled, and what and how variations are taken into account.

Genomes are represented by permutations of the genes in the model that has been employed.

Therefore, in this model genomes:

1. Genomes are considered as ordered sequences of genes or other segments.
2. Two genomes which only differ in the order in which segments occur are considered.
3. It is assumed that there are no duplications or deletion of segments when moving from one genome to another.
4. The individual gene segments in the genomes may be numbered as desired. Assume that the target genome is always the identity or the sorted permutation $id = 12\cdots n$.
5. Hence, genome rearrangement becomes a sorting problem. One starts from one genome, the starting genome; and sorts it to yield the target genome.

> Given a permutation π and a well-defined *sorting primitive S*, what is the smallest number of operations, $d_S(\pi)$, of applying S on π at each step that transforms π into id?

The similarity between two sequences will be measured by the minimum number of special operations such as transpositions, reversals, block moves etc., to transform one sequence into the other. Since the target sequence is always $1, 2, \cdots, n$, the problem is considered a sorting problem. But this is not the usual sorting problem which we are familiar with. It is to sort a sequence in such a way that the number of these special operations is minimized. In other words, finding the algorithms which always sort a sequence with the minimum number of these special operations is an interesting problem.

Some well-known primitives for genome rearrangements are transpositions (Bafna and Pevzner, 1998), reversals, also known as inversions (Bafna and Pevzner, 1996), block moves, also known as strip moves (Bein et al., 2002; Mahajan et al., 2006), block interchanges (Christie, 1998), prefix reversals (Gates and Papadimitriou, 1979), and strip exchanges (Roy and Thakur, 2007). At times, a combination of more than one primitive is used to determine the distance. For instance, transreversals are combinations of transpositions and reversals (Christie and Irving, 2001).

Thus, sorting problems as defined above using special primitives are combinatorial optimization problems, in which the number of steps required to sort an arbitrary permutation needs to be optimized. While SORTING BY REVERSALS (Caprara, 1997), SORTING BY TRANSPOSITIONS (Bafna and Pevzner, 1998), SORTING BY PREFIX REVERSALS (BULTEAU ET AL., 2012), and BLOCK SORTING (Bein et al., 2002) have been proven to be NP-hard, the computational complexity of sorting by strip exchanges still remains an open question. SORTING BY BLOCK INTERCHANGES is one of the very few problems which have been proven to be polynomially solvable. An $O(n^2)$ exact algorithm exists for this problem (Christie, 1998).

A *block move* is a special case of another primitive, a *transposition.* Block sorting is a nontrivial variation of sorting by transposition. Sorting by transposition also arises in the context of genome rearrangements in computational biology. In sorting by transposition, any substring of π may be moved, not necessarily a block, to a different position at each step (Bafna and Pevzner, 1998). To compute the minimum number of such moves to sort an arbitrary permutation has been recently shown to be NP-hard (Bulteau et al., 2011). The current best-known algorithm for sorting by transposition has an approximation ratio of 1.375 (Elias and Hartman, 2006), and is not known whether it admits a polynomial-time approximation scheme (PTAS). It is not known yet whether block sorting takes 3 times number of transpositions to sorting the arbitrary permutation. However, it is known that optimal transpositions never need to break existing blocks (Mahajan et al., 2007a). This shows how the two problems are closely related. The study of the computational complexity of block sorting, therefore, might provide more insight into the complexity of sorting by transposition.

1.3.2 Optical Character Recognition (OCR)

Block sorting is also motivated by applications in OCR (Gobi et al., 2000; Latifi, 1996). OCR is a method to locate and recognize text stored in an image, usually a digital image, such as a jpeg or a gif file, and then convert the text into a computer recognized form represented by either ASCII or unicode. Essentially, it converts the pixel representation of a letter into its equivalent character representation in a computer (Palkovic, 2008). Text regions, referred to as *zones*, are selected by drawing rectangles or polygons around the text pieces. The order of zones is significant although the output in practice generated by a zoning procedure will not necessarily be the same as the original or correct text. Zones need to be processed further if they are not in correct order either manually by a human editor or by using a certain device. The OCR community has defined a *zoning metric* (Kanai et al., 1995) as the number of editing operations which may be a variety of deletions, insertions and moves (Bein et al., 2005). It is easy to see how this corresponds to the block sorting problem. In essence, the solution to block sorting plays a significant role in defining the zoning metric.

OCR has a large number of benefits. A large collection of paper forms and documents have been documented in many agencies, companies and libraries, from which useful data can be extracted by implementing OCR techniques. It will cost a large amount of time and money to do this manually.

Chapter 2

BACKGROUND

2.1 Block Sorting

Various experiments seem to have been performed initially to test a number of strategies that have been found to perform well practically in the research history of block sorting. As a measure of performance of approximation algorithms, researchers often define the *approximation ratio* as:

$$\text{Approximation Ratio} = \frac{\text{max. \# of operations the approximation algorithm takes}}{\text{min. \# of operations any other optimal algorithms take}}$$

If the approximation ratio equals r and it takes m number of block moves to sort the permutation, the approximation algorithm will have $r \times m$ moves to get the identity permutation. The first non-trivial approximation algorithm with an approximation ratio of 2 solves the *block merging* problem, a problem related to block sorting (Bein et al., 2005).

2.1.1 Lower Bounds for Block Sorting

Given that there are n blocks in a permutation, since there will be only one block in the identity permutation, $n-1$ blocks need to be removed to get to the identity permutation. One particular block move can at most eliminate three blocks (Mahajan et al., 2006).

Thus, at least $(n-1)/3$ block moves will be needed to sort an arbitrary permutation which has n blocks to begin with. In other words, the lower bound for the number of block moves in block sorting equals $(n-1)/3$.

2.1.2 A 3-Approximation Algorithm for Block Sorting

It is fairly straightforward to show that a 3-approximation algorithm exists for Block Sorting. If the starting permutation is not sorted, an arbitrary maximal block may be simply picked and moved to a position to form a bigger block (Bein et al., 2005). Thus, one block move can eliminate at least one block by using the trivial algorithm. It can be concluded that the upper bound for block sorting is $n-1$ block moves. That is, it will take at most $n-1$ block moves to reach the identity permutation. The lower bound of block sorting is $(n-1)/3$. Therefore, the approximation ratio of block sorting is $\frac{(n-1)}{(n-1)/3}$, which equals 3.

In the following, two of the existing approximation algorithms for block sorting will be discussed.

2.1.3 Block Merging

Block merging is similar to block sorting in which the input permutation is divided into a set of maximal increasing sequences (Narayanaswamy and Roy, 2015). Block merging allows to merge the increasing sequences by moving blocks across the increasing sequences (Narayanaswamy and Roy, 2015). The block merging problem has been shown to be polynomially solvable with a $O(n^3)$ run time (Mahajan et al., 2006). Block merging also guarantees at most 2 times the optimal number of block moves needed to sort any permutation in polynomial time.

An increasing sequence in a permutation is a maximal increasing subsequence of the permutation (Mahajan et al., 2006). For instance, the permutation, 8 2 5 6 3 9 1 4 7, consists of the four increasing sequences (8), (2, 5 6), (3, 9), (1, 4, 7). Thus, a permutation may be considered as a multi-set of disjoint increasing sequences (Mahajan et al., 2006).

A block merging move is to choose a block from any of the increasing sequences and insert the block into one of other sequences so as to merge the block into a new bigger block there (Mahajan et al., 2006). With a block merging move, all increasing sequences of the permutation stay as increasing sequences, but with different blocks (Mahajan et al., 2006).

Figure 2.1 shows an example of how block sorting works. In Figure 2.1,

1. The block $\boxed{2}$ is moved to be positioned following the block $\boxed{1}$ to form the new block $\boxed{1\ 2}$.
2. The block $\boxed{3}$ is moved to be positioned between $\boxed{1\ 2}$ and $\boxed{4}$ to form the new block $\boxed{1\ 2\ 3\ 4}$.
3. The block $\boxed{5\ 6}$ is moved to be positioned between the block $\boxed{1\ 2\ 3\ 4}$ and the block $\boxed{7}$ to form the new block $\boxed{1\ 2\ 3\ 4\ 5\ 6\ 7}$ and the new block $\boxed{8\ 9}$ since the block $\boxed{5\ 6}$ is moved else where.

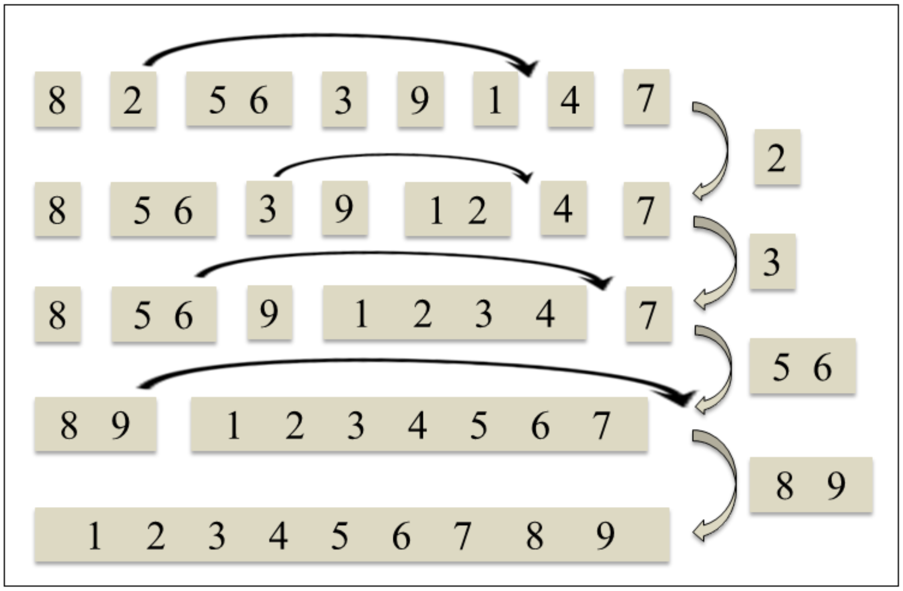

Figure 2.1: Block Sorting Moves

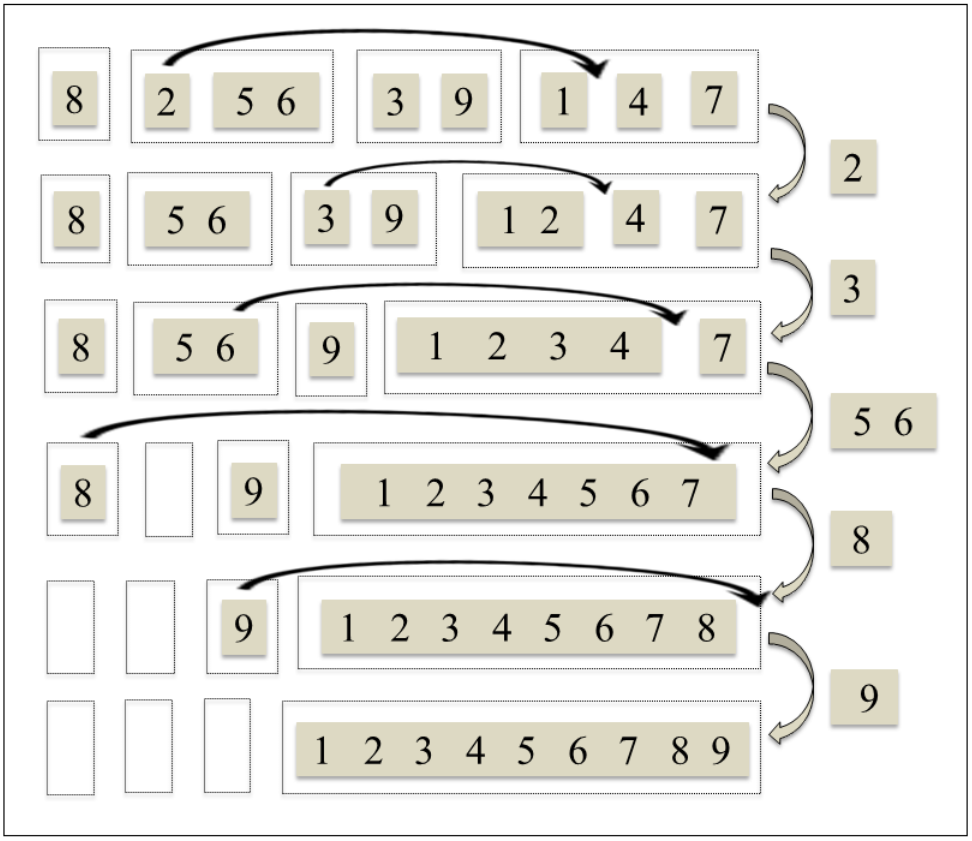

Figure 2.2: Block Merging Moves

4. The block $\boxed{8\ 9}$ is moved to be positioned following the block $\boxed{1\ 2\ 3\ 4\ 5\ 6\ 7}$ to get to the identity permutation, which is a single block at the end.

Block merging divides the permutation into the increasing sequences, which are shown in the boxes in Figure 2.2. In Figure 2.2,

1. The block $\boxed{2}$ is moved to be positioned following the block $\boxed{1}$ from its increasing sequence to another increasing sequence to form the new block $\boxed{1\ 2}$.

2. The block $\boxed{3}$ is moved to be positioned between the block $\boxed{1\ 2}$ and the block $\boxed{4}$ from its increasing sequence to another increasing sequence to form the new block $\boxed{1\ 2\ 3\ 4}$.

3. The block $\boxed{5\ 6}$ is moved to be positioned between the block $\boxed{1\ 2\ 3\ 4}$ and the block $\boxed{7}$ from its increasing sequence to another increasing sequence to form the new block $\boxed{1\ 2\ 3\ 4\ 5\ 6\ 7}$.

4. The block $\boxed{8}$ is moved to be positioned following the block $\boxed{1\ 2\ 3\ 4\ 5\ 6\ 7}$ from its increasing sequence to another increasing sequence to form the new block $\boxed{1\ 2\ 3\ 4\ 5\ 6\ 7\ 8}$.

5. The block $\boxed{9}$ is moved to be positioned following the block $\boxed{1\ 2\ 3\ 4\ 5\ 6\ 7}$ from its increasing sequence to another increasing sequence to get to the identity permutation.

The last step in Figure 2.1 moves 8 9 as one block while 8 and 9 are in two different sequences in Figure 2.2. Therefore, the block 8 and the block 9 have to be moved to the other increasing sequences to get to the identity permutation.

Block merging is the problem of finding the minimum number of block moves k, such that the multi-set can transformed into an identity permutation, or just a single increasing sequence; if n is the number of blocks in the original multi-set, it has been proven that $\frac{n-1}{2} \leq k \leq n-1$ (Mahajan et al., 2006).

On the one hand, any block moved into other sequences can remove the number of blocks by at most 2. The total number of blocks that need to be removed is equal to $n-1$. Thus, the lower bound of the block merging problem is $\frac{n-1}{2}$, which is the best case. On the other hand, it removes at least one block each step as the worst case. For instance, the permutation in the descending order gives rise to repeatedly move the block that contains one block to its successor(Mahajan et al., 2006). Thus, the upper bound is $n-1$, which makes block merging a 2-approximation algorithm for block sorting.

2.1.4 Block Deletion

The second currently known 2-approximation algorithm solves the block deletion problem, which has been shown to be equivalent to the block sorting problem. Staring from a given permutation, block deletion proceeds by deleting one block at a time until an increasing subsequence that is not a block is obtained. (Bein et al., 2005). That is, to delete a block in each step until it becomes a monotonically increasing sequence. If each block is named equal to one of its elements, the block sorting problem on an irreducible list must be equivalent to the block deletion problem on the list of distinct block names (Bein et al., 2002). Block deletion was designed as an approximation algorithm for block sorting after block merging has been designed. Block deletion may be related to block merging but they are not identical problems (Bein et al., 2002). Block deletion also takes two times the number of block moves of block sorting (Bein et al., 2002). While block merging takes $O(n^3)$ run time, block deletion has an improved $O(n^2)$ run time (Bein et al., 2002).

As an example, corresponding to the block sorting moves shown in Figure 2.3, block deletions as shown in Figure 2.4 are carried out.

In Figure 2.3,

1. The block $\boxed{4}$ is moved to be positioned in front of the block $\boxed{5\ 6}$ to form the new block $\boxed{4\ 5\ 6}$.

2. The block $\boxed{5\ 6}$ is moved to be positioned following the block $\boxed{3}$ to form the new block $\boxed{2\ 3\ 4\ 5\ 6}$.

3. The block $\boxed{2\ 3\ 4\ 5\ 6}$ is moved to be positioned between the block $\boxed{1}$ and $\boxed{7}$ to form the new block $\boxed{1\ 2\ 3\ 4\ 5\ 6\ 7}$.

4. The block $\boxed{8\ 9}$ is moved to be positioned following the other block to get to the identity permutation.

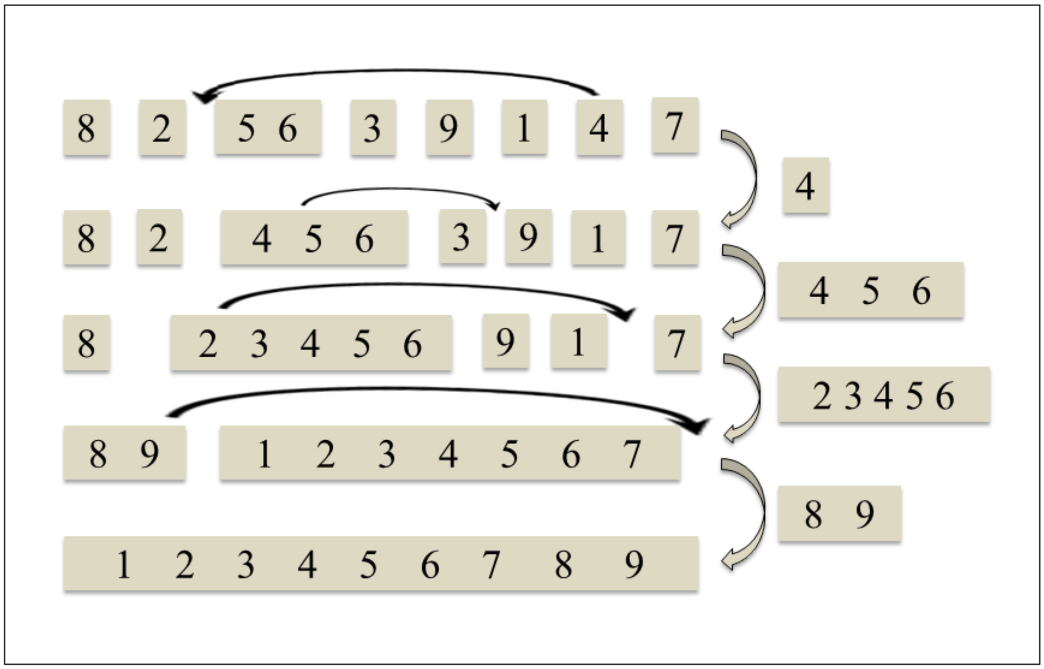

Figure 2.3: The Block Sorting Moves

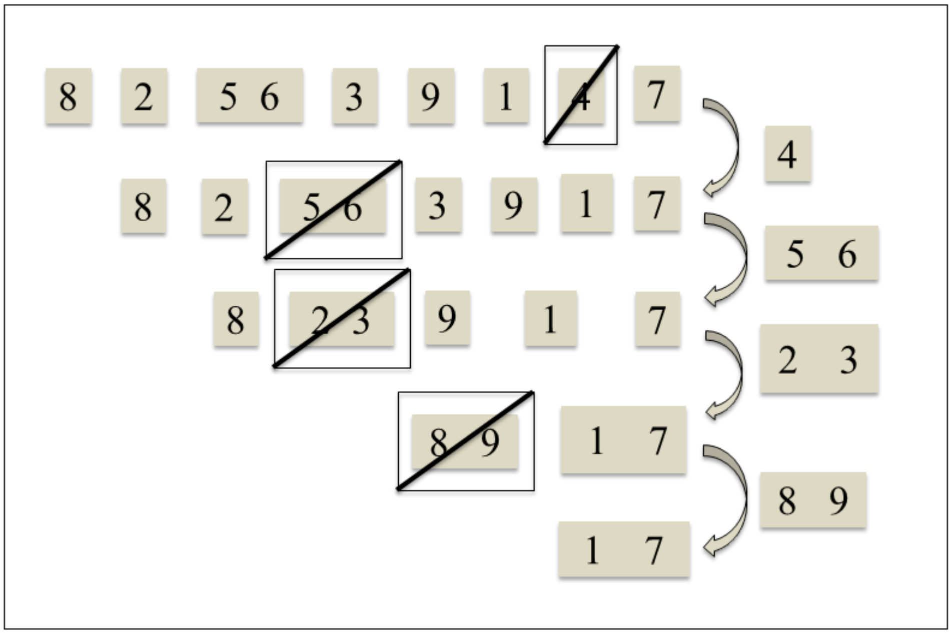

Figure 2.4: Relative Block Deletions

In Figure 2.4,

1. The block $\boxed{4}$ is deleted corresponding to the block move $\boxed{4}$ in Figure 2.3.
2. The block $\boxed{5\ 6}$ is deleted corresponding to the block move $\boxed{4\ 5\ 6}$.
3. The block $\boxed{2\ 3}$ is deleted corresponding to the block move $\boxed{2\ 3\ 4\ 5\ 6}$.
4. The block $\boxed{8\ 9}$ is deleted corresponding to the block move $\boxed{8\ 9}$ in Figure 2.3.

When once moves a block in block sorting, the block is deleted in block deletion so as to have the monotonically increasing sequence at the end, which corresponds to the identity permutation in block sorting. The example above gets 1 7 as the monotonically increasing sequence at the end. The end result of 1 7 in block deletion then indicates that the original permutation has been sorted with the idea that the blocks deleted will occupy their correct positions in the identity permutation.

2.2 Sorting by Transpositions

The sorting by transpositions problem occurs in the study of genome rearrangement in the field of computational biology (Bafna and Pevzner, 1998). In the genome rearrangement problem, the genes on a genome of a species can be represented by a list of integers (Bafna and Pevzner, 1998), such as $1, 2, 3, \cdots, n$. Thus, a genome can be represented by a permutation of integers. A Transposition can be defined as a primitive to transform a given genome. In other words, a transposition transforms a given permutation to another permutation.

A transposition exchanges two adjacent subsequences of any length without changing the order of the two subsequences. For instance, suppose the starting genome is X: 3 1 5 2 4, and the target genome is Y: 1 2 3 4 5, the genome A: 3 2 4 1 5 can be obtained by exchanging 1 5 and 2 4. Similarly, genome A may lead to the genome B: 2 3 4 1 5 by exchanging 3 and 2. Finally, the target genome Y: 1 2 3 4 5 from the genome B can be obtained by exchanging 1 and 2 3 4. Thus, the genome X is transformed into the genome Y by three transpositions.

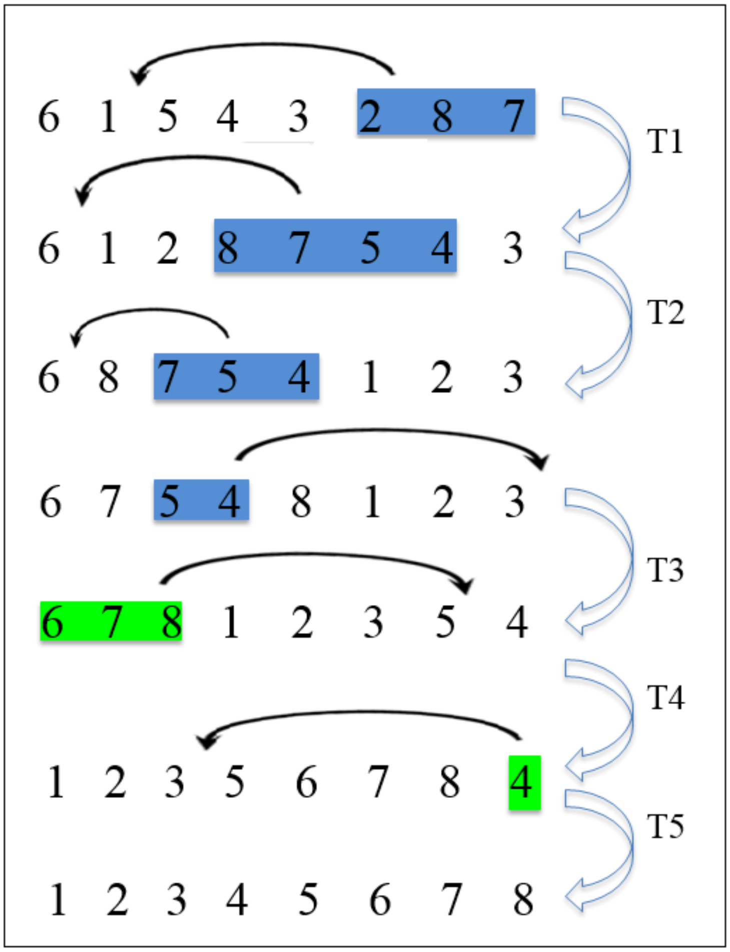

Figure 2.5: Transpositions in Sorting by Transposition

Block sorting is a nontrivial variation of sorting by transposition (Mahajan et al., 2007b). A sequence of block moves defines a sequence of transpositions while a sequence of transpositions is not necessarily a sequence of block moves (Bein et al., 2002). Block sorting is a special case of sorting by transpositions with a block move during each step as a restriction. Figure 2.5 presents an example illustrating how a transposition differs from a block move.

There are five transpositions.

1. T1 exchanges 2 8 7 and 5 4 3.
2. T2 exchanges 8 7 5 4 and 1 2.
3. T3 exchanges 5 4 and 8 1 2 3.
4. T4 exchanges 6 7 8 and 1 2 3 5. Since 6 7 8 is also a block, T4 is also a block move.
5. T5 exchanges 4 and 5 6 7 8. T5 is also a block move since 4 itself can be a block.

Both block sorting and sorting by transpositions have been proven to be NP-hard. A block cannot be divided once two or more blocks are combined into a new block. Block sorting sorts a given permutation based on block moves as a primitive while sorting by transpositions does not have the restriction of blocks, which results in the much better approximation ratio for sorting by transpositions than for block sorting. In general, the best approximation algorithm of sorting by transpositions is not applicable to block sorting.

Chapter 3

NEW APPROXIMATION ALGORITHMS FOR BLOCK SORTING

This chapter represents our contributions to the research on block sorting and presents two new approximation algorithms for block sorting, beginning with a few definitions and concepts.

3.1 Preliminaries

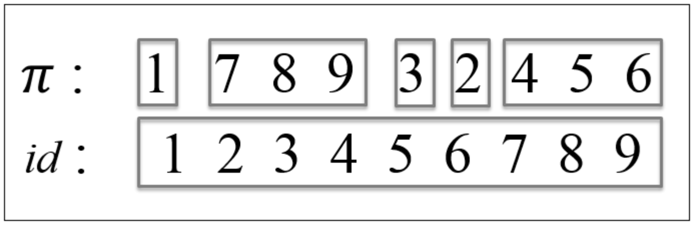

Figure 3.1: Blocks and the Corresponding Identity Permutation

Let the set $\{1, 2, \cdots, n\}$ is denoted by $[n]$, and let S_n denote the set of all permutations over $[n]$, and id the sorted or identity permutation of length n. Figure 3.1 shows an example of blocks in the permutation and the corresponding identity permutation. It is clear that the identity permutation is a single block.

Definition 1 (Block Move)**.** *A* block move *picks up a block and places it elsewhere in the permutation.*

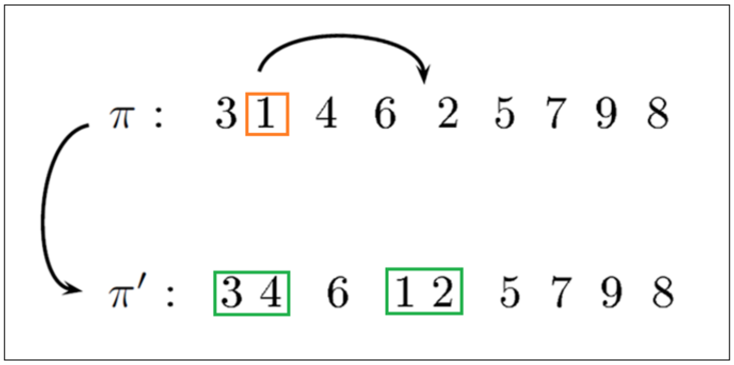

Figure 3.2: An Example of a Block Move

As an example, Figure 3.2 shows a block move on a permutation π to obtain the permutation π'. The permutation 3 1 4 6 2 5 7 9 8 contains 9 blocks. A block $\boxed{1}$ is moved in front of the block $\boxed{2}$ and the block move reduces 2 blocks in the permutation.

Definition 2 (Block Sorting Schedule)**.** *A block sorting schedule $\mathscr{S}$ is a sequence of block moves such that performing the sequence of block moves on permutation π results in the identity permutation id.*

The length of a block sorting schedule is the number of block moves in the schedule. Block sorting distance $bs(\pi)$ is the number of block moves in a minimum length block sorting schedule for π. A block sorting schedule is shown on the permutation 8 2 5 6 3 9 1 4 7 in Figure 3.3. The block moves are indicated at each step. The block sorting distance for this example equals 3 since the number of block moves in Figure 3.3 is 3.

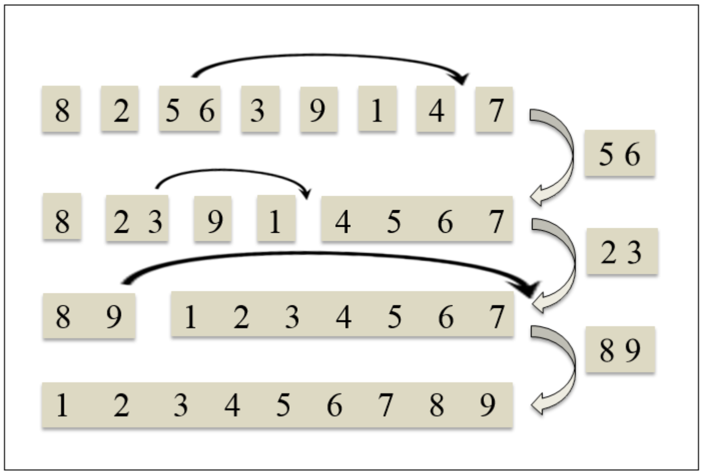

Figure 3.3: An Example of a Block Sorting Schedule

For the following definitions, it is denoted that a block $a \in \pi$, its position in π by $\pi(a)$.

Definition 3 (Reversal)**.** *In a permutation π, a pair of consecutive elements or blocks a and b form a reversal if $a > b$, and $\pi(b) = \pi(a) + 1$.*

Denote the number of reversals in π by $rev(\pi)$. It has been shown that a block sorting sequence of length $rev(\pi)$ is optimal (Mahajan et al., 2006), since the block sorting distance $bs(\pi) \geq rev(\pi)$. As an example, the reversals in the permutation 8 2 5 6 3 9 1 4 7 are $\{(8,2),(\boxed{5\ 6},3),(9,1)\}$. There are $n-1$ reversals in the reversed permutation with n blocks, which is the worst case.

Definition 4 (Inversion)**.** *In a permutation π, a pair of elements or blocks a and b form an inversion if $b = a+1$, and $\pi(a) > \pi(b)$.*

Inversions are also called *dual reversals* (Mahajan et al., 2006). Denote the number of inversions in π by $inv(\pi)$. It has been shown that a block sorting sequence of length $inv(\pi)$ is optimal (Mahajan et al., 2006), since the block sorting distance $bs(\pi) \geq inv(\pi)$. For instance, the inversions in the permutation 8 2 5 6 3 9 1 4 7 are $\{(1,2),(4,\boxed{5\ 6}),(7,8))$. To emphasize, $(8,1)$ and $(9,1)$ are not inversions in the research. There are $n-1$ inversions in the reversed permutation with n blocks, which is the worst case.

Definition 5 (Run)**.** *The blocks* $\{a, a+1, a+2, \ldots, a+r-1\}$ *form a* run *of length* r *if,* r *is the largest values such that* $\pi(a) < \pi(a+1) < \pi(a+2) < \ldots < \pi(a+r-1)$ *(Huang and Roy, 2014).*

Then number of runs in π is denoted by $runs(\pi)$. As an example, the runs in the permutation 8 2 5 6 3 9 1 4 7 are $\{(1),(2,3,4),(\boxed{5\ 6},7),(8,9))$. The longest run in the permutation is $(2,3,4)$, which has 3 blocks. The number of the blocks in the longest run will help generate a lower bound of block sorting.

3.2 Run Merging Algorithm

To get to the lower bounds for block sorting, it is denoted that

1. n is the number of blocks in π,
2. i is the number of inversions in π, i.e. $inv(\pi) = i$,
3. b is the number of reversals in π, i.e. $rev(\pi) = b$,

4. r is the number of blocks in the *longest* run of π.

Then $bs(\pi) \geq n-1-(i+b)$ (Bein et al., 2002). Algorithm 1 describes the first new approximation algorithm for block sorting. This algorithm is called *run merging*. In simple words, the longest run in π need to be found, then keeping the r elements of that run *intact*, every other block is moved to obtain the sorted permutation id.

Algorithm 1 Run Merging Algorithm.

```
$\pi \leftarrow$ the input permutation
$id \leftarrow$ the sorted permutation
$R \leftarrow$ longest run in $\pi$
while $\pi \neq id$ do
    if Block $B \notin R$ then
        if $B$ is not in proper position then
            Move $B$ either before $succ(B)$ or after $pred(B)$
        end if
    end if
end while
```

Theorem 3.2.1. *Algorithm 1 (Run Merging) is a 2 approximation algorithm for block sorting.*

Proof. The above algorithm takes at most $n-r$ moves where n is the number of blocks in π. It was proceed to be proven that the approximation ratio $\frac{n-r}{n-1-(i+b)}$ is at most 2 by considering different ranges of value for $n-1-(i+b)$.

First, let us consider the case $n-1-(i+b) < 0$. In this case, since the ratio is always negative, it cannot exceed 2.

Next, consider the case $n-1-(i+b) \geq 0$, and then the following sub cases.

Case A: $n-1-(i+b) < i+b$. This means $n-1 < 2(i+b)$. In order for $\dfrac{n-r}{n-1-(i+b)} \leq 2$ to be true, it must be true that:

$n-r \leq 2(n-1) - 2(i+b)$

$n-r < 2(n-1) - (n-1)$

$n-r < n-1$

For all $r > 1$, $n-r < n-1$ is always true. Thus, the bound has been established.

Case B: $n-1-(i+b) \geq i+b$. This means $n-1 \geq 2(i+b)$, or $\dfrac{n-1}{2} \geq (i+b)$. Thus, $\dfrac{n-r}{n-1-(i+b)} \leq \dfrac{n-1}{n-1-(i+b)} < \dfrac{n-1}{n-1-\dfrac{n-1}{2}} = 2.$

□

Theorem 3.2.2. *Algorithm 1 (Run Merging) runs in linear time $O(n)$, where n is the number of blocks in π.*

Proof. Finding the longest run in π takes $O(n)$ time. The remaining part of the algorithm takes at most $n-r$ block moves which is again $O(n)$. Therefore the total runtime is $O(n)$. □

3.3 Ordered Pair Fixing

Definition 6 (Ordered pair)**.** *The blocks (a,b) form an ordered pair if $a < b$ and $\pi(a) < \pi(b)$.*

As an example, some of the ordered pairs in the permutation, 8 2 5 6 3 9 1 4 7, are $\{(8,9),(2,\boxed{5\ 6}),(2,3)(2,9),(2,4),(2,7)\}$. In other words, $(a,b) \in \pi$ form an ordered pair if they appear in the same order in π as they appear in the identity permutation id.

In this section, it is observed that keeping just one ordered pair fixed instead of the longest run as in run merging, and sorting the rest of the elements is sufficient to yield a 2-approximation algorithm for block sorting.

Algorithm 2 (Ordered Pair Fixing) An ordered pair (a, b) in π needs to be found first. Then, keeping the elements a and b of that ordered pair *intact*, every other block is moved to obtain the sorted permutation id. However, if π does not contain an ordered pair, it is the reversed permutation.

Algorithm 2 Ordered Pair Fixing Algorithm.

```
$\pi \leftarrow$ the input permutation
$id \leftarrow$ the sorted permutation
$(a, b) \leftarrow$ an ordered pair in $\pi$
while $\pi \neq id$ do
    if $B$ is a block such that $a \neq B$ and $b \neq B$ then
        if $B$ is not in proper position then
            Move $B$ either before $succ(B)$ or after $pred(B)$
        end if
    end if
end while
```

Theorem 3.3.1. *Algorithm 2 (Ordered Pair Fixing) is a 2 approximation algorithm for block sorting.*

Proof. Algorithm 2 (Ordered Pair Fixing) takes at most $n-2$ block moves where n is the number of blocks in π. The approximation ratio is thus $\dfrac{n-2}{n-1-(i+b)}$. This ratio is at most 2 by exploring different ranges of value for $n-1-(i+b)$.

First, let us consider the case $n-1-(i+b)<0$. In this case, since the ratio is always negative, it cannot exceed 2.

Next, consider the case $n-1-(i+b)\geq 0$, and then the following sub cases.

Case A: $n-1-(i+b)<i+b$. This means $n-1<2(i+b)$. In order for $\frac{n-2}{n-1-(i+b)}\leq 2$ to be true, it must be true that:

$n-2\leq 2(n-1)-2(i+b)$

$n-2<2(n-1)-(n-1)$

$n-2<n-1$, which is always true.

Case B: $n-1-(i+b)\geq i+b$. This means $n-1\geq 2(i+b)$, or $\frac{n-1}{2}\geq(i+b)$. Thus, $\frac{n-2}{n-1-(i+b)}\leq\frac{n-1}{n-1-(i+b)}<\frac{n-1}{n-1-\frac{n-1}{2}}=2.$

Theorem 3.3.2. *Algorithm 2 runs in linear time $O(n)$, where n is the number of blocks in π.*

Proof. Finding an ordered pair $(a,b)\in\pi$ takes $O(n)$ time. The remaining part of the algorithm takes at most $n-2$ block moves which is again $O(n)$. Therefore the total runtime is $O(n)$. ■

3.4 Significance of Result

The result is significant because it considerably simplifies the design of a 2-approximation algorithm for block sorting. It eliminates any preprocessing that the previous algorithms (Bein et al., 2005; Mahajan et al., 2007a, 2006) rigorously perform to achieve a 2-approximation. Though this two algorithms are equivalent in terms of the approximation ratio to the best known algorithms, the implementations offered use considerably less resources than their predecessors.

Chapter 4

CONCLUDING REMARKS

This chapter restates the results on block sorting that have been discovered in the study. It also includes the discussion of directions for further research on the problem.

4.1 Summary of Contributions

The study developed two approximation algorithms for block sorting, known as Run Merging and Ordered Pair Fixing. The development is based on the new concepts introduced, namely, *runs* and *ordered pairs*. The two algorithms have an approximation ratio of 2 equal to the best known. Additionally, they incur linear time in terms of the number of block moves as opposed to the cubic and quadratic times needed by the preceding algorithms for block sorting. The two algorithms are the simplest algorithms for this problem in terms of analysis and implementation.

4.2 Future Directions

There are still a few open questions on block sorting and sorting by transposition. A few of them are listed below.

1. Is there an approximation algorithm for block sorting with an approximation ratio less than 2?
2. Can block sorting be performed using an algorithm that takes a number of block moves less than 3 times the number of transpositions used by sorting by transposition?
3. Is sorting by transposition hard to approximate?
4. Can a fully polynomial-time approximation scheme(FPTAS) for sorting by transposition be designed?

LIST OF PUBLICATIONS FROM THE THESIS

1. J. Huang and S. Roy. *On Sorting Under Special Transpositions.* In Proceedings of IEEE 14th International Conference on BioInformatics and BioEngineering(BIBE), 2014.

2. J. Huang, S. Roy, and A. Asaithambi. *New Approximations for Block Sorting.* Submitted to the NetMAHIB journal.

REFERENCES

Bafna, Vineet and Pevzner, Pavel A. 1996. Genome Rearrangements and Sorting by Reversals. *SIAM J. Comput.* 25, 2 (Feb. 1996), 272–289. DOI:http://dx.doi.org/10.1137/S0097539793250627

Bafna, Vineet and Pevzner, Pavel A. 1998. Sorting by Transpositions. *SIAM Journal on Discrete Mathematics* 11, 2 (May 1998), 224–240. DOI:http://dx.doi.org/10.1137/S089548019528280X

Bein, Wolfgang W., Larmore, Lawrence L., Morales, Linda, and Sudborough, I. Hal. 2005. A Faster and Simpler 2-approximation Algorithm for Block Sorting. In *Proceedings of the 15th International Conference on Fundamentals of Computation Theory (FCT'05).* Springer-Verlag, Berlin, Heidelberg, 115–124. DOI:http://dx.doi.org/10.1007/11537311_11

Bein, W.W., Larmore, L.L., Latifi, S., and Sudborough, IH. 2002. Block sorting is hard. In *Parallel Architectures, Algorithms and Networks, 2002. I-SPAN '02. Proceedings. International Symposium on.* 307–312. DOI:http://dx.doi.org/10.1109/ISPAN.2002.1004305

Bulteau, Laurent, Fertin, Guillaume, and Rusu, Irena. 2011. Sorting by Transpositions is Difficult. In *Proceedings of the 38th International Colloquim Conference on Automata, Languages and Programming - Volume Part I (ICALP'11).* Springer-Verlag, Berlin, Heidelberg, 654–665. http://dl.acm.org/citation.cfm?id=2027127.2027197

Bulteau, Laurent, Fertin, Guillaume, and Rusu, Irena. 2012. Pancake Flipping is Hard. In *Proceedings of the 37th International Conference on Mathematical Foundations of Computer Science (MFCS'12).* Springer-Verlag, Berlin, Heidelberg, 247–258. DOI:http://dx.doi.org/10.1007/978-3-642-32589-2_24

Caprara, Alberto. 1997. Sorting by Reversals is Difficult. In *Proceedings of the First Annual International Conference on Computational Molecular Biology (RECOMB '97).* ACM, New York, NY, USA, 75–83. DOI:http://dx.doi.org/10.1145/267521.267531

Christie, D. and Irving, R. 2001. Sorting Strings by Reversals and by Transpositions. *SIAM Journal on Discrete Mathematics* 14, 2 (2001), 193–206. DOI:http://dx.doi.org/10.1137/S0895480197331995

Christie, David A. 1996. Sorting permutations by block-interchanges. *Inform. Process. Lett.* 60, 4 (1996), 165–169.

Christie, David Alan. 1998. *Genome rearrangement problems.* Ph.D. Dissertation. University of Glasgow.

Elias, Isaac and Hartman, Tzvika. 2006. A 1.375-Approximation Algorithm for Sorting by Transpositions. *IEEE/ACM Trans. Comput. Biol. Bioinformatics* 3, 4 (Oct. 2006), 369–379. `DOI:http://dx.doi.org/10.1109/TCBB.2006.44`

Firoz, JesunS., Hasan, Masud, Khan, AshikZ., and Rahman, M.Sohel. 2010. The 1.375 Approximation Algorithm for Sorting by Transpositions Can Run in O(nlogn) Time. In *WALCOM: Algorithms and Computation*, Md.Saidur Rahman and Satoshi Fujita (Eds.). Lecture Notes in Computer Science, Vol. 5942. Springer Berlin Heidelberg, 161–166. `DOI:http://dx.doi.org/10.1007/978-3-642-11440-3_15`

Gates, William H and Papadimitriou, Christos H. 1979. Bounds for sorting by prefix reversal. *Discrete Mathematics* 27, 1 (1979), 47–57.

Gobi, R, Latifi, S, and Bein, WW. 2000. Adaptive sorting algorithms for evaluation of automatic zoning employed in OCR devices. In *Proceedings of the 2000 International Conference on Imaging Science, Systems, and Technology*. CSREA Press, 253–259.

Gu, Qian-Ping, Peng, Shietung, and Sudborough, Hal. 1999. Contribution: A 2-approximation algorithm for genome rearrangements by reversals and transpositions. *Theoretical Computer Science* 210 (1999), 327–339. `https://login.dax.lib.unf.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=edselp&AN=S0304397598000929&site=eds-live`

Hannenhalli, Sridhar and Pevzner, Pavel A. 1999. Transforming Cabbage into Turnip: Polynomial Algorithm for Sorting Signed Permutations by Reversals. *J. ACM* 46, 1 (Jan. 1999), 1–27. `DOI:http://dx.doi.org/10.1145/300515.300516`

Hartman, Tzvika and Shamir, Ron. 2006. A simpler and faster 1.5-approximation algorithm for sorting by transpositions. *Information and Computation* 204, 2 (2006), 275–290. `DOI:http://dx.doi.org/10.1016/j.ic.2005.09.002`

Hochbaum, D.S. 1997. *Approximation Algorithms for NP-hard Problems.* PWS Publishing Company. `https://books.google.com/books?id=E2VRAAAAMAAJ`

Huang, Jici and Roy, Swapnoneel. 2014. On Sorting under Special Transpositions. In *Bioinformatics and Bioengineering (BIBE), 2014 IEEE International Conference on.* IEEE, 325–328.

Kanai, J., Rice, S.V., Nartker, T.A., and Nagy, G. 1995. Automated evaluation of OCR zoning. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 17, 1 (Jan 1995), 86–90. `DOI:http://dx.doi.org/10.1109/34.368146`

Latifi, Shahram. 1996. How can permutations be used in the evaluation of zoning algorithms? *International journal of pattern recognition and artificial intelligence* 10, 03 (1996), 223–237.

Lin, Ying Chih, Lu, Chin Lung, Chang, Hwan-You, and Tang, Chuan Yi. 2005. An efficient algorithm for sorting by block-interchanges and its application to the evolution of vibrio species. *Journal of Computational Biology* 12, 1 (2005), 102–112.

Mahajan, Meena, Rama, Raghavan, Raman, Venkatesh, and Vijaykumar, S. 2006. Approximate Block Sorting. *International Journal of Foundations of Computer Science* 17, 02 (2006), 337–355. DOI:http://dx.doi.org/10.1142/S0129054106003863

Mahajan, Meena, Rama, Raghavan, and Vijayakumar, S. 2007a. Block Sorting: A Characterization and Some Heuristics. *Nordic J. of Computing* 14, 1 (Jan. 2007), 126–150. http://dl.acm.org/citation.cfm?id=1515784.1515790

Mahajan, Meena, Rama, Raghavan, and Vijayakumar, S. 2007b. Block Sorting: A Characterization and Some Heuristics. *Nordic J. of Computing* 14, 1 (Jan. 2007), 126–150. http://dl.acm.org/citation.cfm?id=1515784.1515790

Mira, Cleber and Meidanis, Joao. 2007. Sorting by block-interchanges and signed reversals. In *Information Technology, 2007. ITNG'07. Fourth International Conference on.* IEEE, 670–676.

Narayanaswamy, N.S. and Roy, Swapnoneel. 2015. Block Sorting Is APX-Hard. In *Algorithms and Complexity*, Vangelis Th. Paschos and Peter Widmayer (Eds.). Lecture Notes in Computer Science, Vol. 9079. Springer International Publishing, 377–389. DOI:http://dx.doi.org/10.1007/978-3-319-18173-8_28

Palkovic, AJ. 2008. Improving optical character recognition. In *Proceedings of the 2nd Villanova University Undergraduate Computer Science Research Symposium (CSRS 2008). December*, Vol. 5.

Palmer, JeffreyD. and Herbon, LauraA. 1988. Plant mitochondrial DNA evolved rapidly in structure, but slowly in sequence. *Journal of Molecular Evolution* 28, 1-2 (1988), 87–97. DOI:http://dx.doi.org/10.1007/BF02143500

Roy, S. and Thakur, AK. 2007. Towards Construction of Optimal Strip-Exchanging Moves. In *Bioinformatics and Bioengineering, 2007. BIBE 2007. Proceedings of the 7th IEEE International Conference on.* 821–827. DOI:http://dx.doi.org/10.1109/BIBE.2007.4375655

VITA

Jici Huang was born She earned a Bachelor of Science in Information Systems from Donghua University in 2011. She expects to receive a Master of Computer and Information Sciences from the University of North Florida, August 2015. Dr. Swapnoneel Roy of the University of North Florida is serving as Jici's thesis advisor.

Jici has on-going interests in the computational biology and the software development fields. She has two years of experience in software developing and software quality assurance. She is able to write software test cases and to perform tests on various Operating Systems and web browsers.

In the pursuit of her master's degree, she has advanced her knowledge in artificial intelligence, programming language design, and new technologies for both mobile apps and web development. Jici is fluent in Mandarin and English. She has been accepted into Computer Science Doctoral program at the University of Nevada, Las Vegas, beginning Fall 2015.