Approximate Block Sorting*

Meena Mahajan† Raghavan Rama‡ Venkatesh Raman†
S. Vijayakumar†§

April 21, 2005

Abstract

We consider the problem BLOCK-SORTING: Given a permutation, sort it by using a minimum number of *block moves*, where a block is a maximal substring of the permutation which is also a substring of the identity permutation, and a block move repositions the chosen block so that it merges with another block. Although this problem has recently been shown to be NP-hard [3], nothing better than a trivial 3-approximation was known. We present here the first non-trivial approximation algorithm to this problem. For this purpose, we introduce the following optimization problem: Given a set of increasing sequences of distinct elements, merge them into one increasing sequence with a minimum number of block moves. We show that the merging problem has a polynomial time algorithm. Using this, we obtain an $O(n^3)$ time 2-approximation algorithm for BLOCK-SORTING[1].

We also observe that BLOCK-SORTING, as well as sorting by transpositions, are fixed-parameter-tractable in the framework of [6].

1 Introduction

The BLOCK-SORTING problem is the problem of sorting a list of distinct items with as few moves as possible, where in a single move, a maximal sublist of items which is already

*A preliminary version of this paper was presented at the 23rd FSTTCS Conference in 2003 under the title "Merging and Sorting by Strip Moves" [18]

†The Institute of Mathematical Sciences, CIT Campus, Taramani, Chennai 600 113, India. Email: {meena,vraman,vjy}@imsc.res.in

‡Department of Mathematics, Indian Institute of Technology, Madras, Chennai 600 036, India. Email: ramar@iitm.ac.in . Supported in part by a DST grant for the project DST/MS/124/99.

§Part of this work was done while this author was at the Department of Mathematics, Indian Institute of Technology, Madras.

[1]We have learned from one of the referees that a 2-approximation algorithm for BLOCK-SORTING can be obtained in time $O(n^2)$, as presented in the recent manuscript "A Polynomial Time 2-Approximation for Block Sorting" by W. Bein, L. Larmore, L. Morales, and H. Sudborough, available at `http://www.egr.unlv.edu/ bein/pubs/twoblock.ps`

a substring of the sorted list (called a *block*, or a *strip*) is moved so as to combine with another block. This problem arises in connection with optical character recognition, and considerable effort has already been devoted to devising efficient and good heuristics for it; see for instance [9, 15]. In [3], it is proved to be NP-hard, and the authors mention that nothing better than a trivial 3-approximation algorithm is known. In this paper, we present an efficient 2-approximation algorithm for this problem.

A generalisation of the BLOCK-SORTING problem is that of *sorting by transpositions*, which arises in the context of genome rearrangements. A transposition is the operation of picking any sublist, not necessarily a block, and placing it elsewhere. (Clearly, every block move is a transposition, though many transpostions are not block moves.) The sorting by transpositions problem is the problem of finding a shortest sequence of transpostions that sorts a given list. For this problem, 3/2-approximation algorithms are known, but its computational complexity remains unsettled [2, 4, 8, 12][2]. In particular, the NP-hardness of BLOCK-SORTING does not imply NP-hardness for sorting by transpositions. Conversely, the 3/2-approximation algorithm for sorting by transpositions does not directly imply a 3/2 approximation algorithm for BLOCK-SORTING. It is known that an optimal transposition sequence can consist purely of transpositions where the moved substrings consist of one or more adjacent blocks; that is, optimal transpositions need never break existing blocks [4]. This fact pinpoints the close connection between BLOCK-SORTING and sorting by transpositions. For related sorting problems, see [1, 4, 5, 7, 10, 11, 16, 20, 21].

The rest of the paper is organized as follows. In the following section, we fix notation, and establish trivial bounds. In Section 3 we introduce a new but similar problem BLOCK-MERGING, that of optimally merging increasing sequences via block moves across sequences. Sections 4 and 5 then show that BLOCK-MERGING is in P, and that BLOCK-MERGING provides a 2-approximation for BLOCK-SORTING, respectively. In Section 6 we consider relaxations of these problems. We consider variants where (1) at every step a block must be moved, but need not combine with another block, and (2) at every step, a substring of a block may be moved anywhere. Not surprisingly, we show that nothing is gained by these relaxations; the minimum number of such steps to sort is the same as that in block sorting. In the discussion, we observe that both BLOCK-SORTING and sorting by transpositions are fixed-parameter tractable, but pose meaningful problems in this framework that are still open.

2 Preliminaries

We denote by $[n]$ the set $\{1, 2, \ldots, n\}$, by S_n the set of all permutations over $[n]$, and by $\mathtt{id}_n$ the identity permutation of S_n.

Without loss of generality, we assume that the given list to be sorted is a permutation $\pi \in S_n$, written as a string $\pi_1\pi_2 \ldots \pi_n$.

[2]Isaac Elias and Tzvika Hartman have recently obtained a 1.375-approximation; however, there is no manuscript available yet.

A *block* or a *strip* is a maximal substring of π which is also a substring of the identity permutation $\mathtt{id}_n$. A *block move* consists of moving a block to a place where it merges with another block. A transposition consists of moving any substring of π to any other location. (Note: In some earlier literature, e.g. [13, 14], block moves are used to refer to transpositions. We use the term block move in the sense described above, as used in [3]. In the last section, where we consider relaxations of BLOCK-SORTING, we specifically use the term strip to avoid confusion.)

For example, in the permutation 6 7 4 5 2 3 1 on 7 elements, there are four blocks, namely $\boxed{6\ 7}$, $\boxed{4\ 5}$, $\boxed{2\ 3}$ and $\boxed{1}$. Moving the block $\boxed{4\ 5}$ to the immediate right of the block $\boxed{2\ 3}$ is a block move, giving rise to the permutation 6 7 2 3 4 5 1. Moving 6 7 4 5 to the immediate right of 1 is not a block move, but it is a transposition, giving rise to the permutation 2 3 1 6 7 4 5.

By $bs(\pi)$ and $t(\pi)$ we denote the lengths of the shortest sequences of block moves and transpositions, respectively, sorting π. We call $bs(\pi)$ and $t(\pi)$ the *block move distance* and *transposition distance*, respectively, of π.

For a permutation π and an element $a \in [n]$, we denote by $\mathtt{block}(a, \pi)$ the block in π containing a, by $\mathtt{last}(a, \pi)$ the largest element of $\mathtt{block}(a, \pi)$, by $\mathtt{next}(a, \pi)$ the element immediately following a in π if such an element exists (undefined if $a = \pi_n$), and by $\mathtt{prev}(a, \pi)$ the element immediately preceding a in π if such an element exists (undefined if $a = \pi_1$). For a block x, we denote by $\mathtt{first}(x, \pi)$ and $\mathtt{last}(x, \pi)$ the smallest and largest elements respectively of x. By $\mathtt{pred}(x, \pi)$ we mean $\mathtt{block}(\mathtt{first}(x, \pi) - 1, \pi)$ (when $\mathtt{first}(x, \pi) > 1$), and $\mathtt{succ}(x, \pi)$ denotes $\mathtt{block}(\mathtt{last}(x, \pi) + 1, \pi)$ (when $\mathtt{last}(x, \pi) < n$). By "move a to predecessor" we mean "move $\mathtt{block}(a, \pi)$ to the immediate right of $\mathtt{pred}(\mathtt{block}(a, \pi), \pi)$". Similarly, "move a to successor" means "move $\mathtt{block}(a, \pi)$ to the immediate left of $\mathtt{succ}(\mathtt{block}(a, \pi), \pi)$". "Move 1 to predecessor" means "move $\mathtt{block}(1, \pi)$ to the extreme left"; "move n to successor" similarly means "move $\mathtt{block}(n, \pi)$ to the extreme right".

Example 2.1 *For the permutation* $\pi = 7\ 6\ 5\ 2\ 3\ 4\ 8\ 1$, $\mathtt{block}(7, \pi) = 7$ *while* $\mathtt{block}(3, \pi) = 2\ 3\ 4$. *Also,* $\mathtt{last}(3, \pi) = 4$ *and* $\mathtt{next}(3, \pi) = 4$*;* $\mathtt{prev}(3, \pi) = 2$*,* $\mathtt{prev}(2, \pi) = 5$ *and* $\mathtt{prev}(7, \pi)$ *is undefined. Further,* $\mathtt{first}(7, \pi) = 7$*,* $\mathtt{last}(2\ 3\ 4, \pi) = 4$*,* $\mathtt{pred}(7, \pi) = 6$*,* $\mathtt{succ}(1, \pi) = 2\ 3\ 4$*, "move* 1 *to predecessor" gives* 1 7 6 5 2 3 4 8*, "move* 1 *to successor" gives* 7 6 5 1 2 3 4 8*, and "move* 8 *to successor" gives* 7 6 5 2 3 4 1 8.

Both BLOCK-SORTING and sorting by transpositions have a 3-approximation algorithm which is fairly straightforward. Let $\#\mathtt{block}(\pi)$ denote the number of blocks in π. It is easy to see that a block move, and even a transposition, can reduce the number of blocks at most by 3. And $\#\mathtt{block}(\mathtt{id}_n) = 1$. Thus reducing the number of blocks to 1 needs at least $(\#\mathtt{block}(\pi) - 1)/3$ transpositions. On the other hand, the naive algorithm of repeatedly moving the block containing 1 to its successor reduces the number of blocks by at least 1 in each move; hence $bs(\pi)$ is no more than $\#\mathtt{block}(\pi) - 1$. Thus

Proposition 2.2

$$\left\lceil \frac{\#\mathtt{block}(\pi) - 1}{3} \right\rceil \le t(\pi) \le bs(\pi) \le \#\mathtt{block}(\pi) - 1$$

It follows that the naive algorithm is a 3-approximation for BLOCK-SORTING and for sorting by transpositions.

It also follows that BLOCK-SORTING is fixed-parameter-tractable FPT, in the framework introduced in [6]. Here, the input instance π comes along with a parameter k, and we want to decide if $bs(\pi) \leq k$. There is an obvious $n^k n^{O(1)}$ algorithm which tries out all n^k sequences of length k. However, since typically n is huge, this is infeasible even for fairly small k. On the other hand, in the FPT framework, we look for an algorithm with run time $f(k)n^c$, where f can be any function, but c is a constant that must be independent of k. In the genome rearrangement setting, for instance, typically k will be less than 20 while n runs into thousands, so even if f is a fast-growing function, such an algorithm would outperform the brute-force $n^k n^{O(1)}$ algorithm. But Proposition 2.2 gives an FPT algorithm for BLOCK-SORTING (and sorting by transpositions): if $k < \#\texttt{block}(\pi)/3$, then the answer is trivially no. Otherwise, at each stage, explore all possible block moves (transpositions that do not break blocks); there are $O(\#\texttt{block}(\pi))$ ($O(\#\texttt{block}(\pi)^3)$, respectively) such moves, and the exploration goes down for k stages, but $O(\#\texttt{block}(\pi)^3) \in O(k^3)$. Thus

Corollary 2.3 *The problems* BLOCK-SORTING *and* Sorting by Transpositions *are both fixed-parameter tractable.*

The bounds of Proposition 2.2 are all tight, and there are permutations for which the inequalities are strict. For instance, in the example below, (a) follows from straightforward induction, while (b) follows from the fact that for $n \geq 3$, $t(\texttt{rev}_n) = \lceil (n+1)/2 \rceil$ [4, 8], where $\texttt{rev}_n$ denotes the reversal of the identity permutation $\texttt{id}_n$. More examples illustrating the other bounds can similarly be constructed.

Example 2.4 *(a) For* $n \geq 1$, $bs(\texttt{rev}_n) = n - 1 = \#\texttt{block}(\texttt{rev}_n) - 1$.
(b) For $n \geq 5$, $\lceil \#\texttt{block}(\texttt{rev}_n) - 1 \rceil / 3 < t(\texttt{rev}_n) < bs(\texttt{rev}_n)$.

In [19], it is shown that for every $\pi \in S_n \setminus \{\texttt{rev}_n\}$, $bs(\pi) < n - 1$. So $\texttt{rev}_n$ is the only worst-case input of length n for BLOCK-SORTING. This is not the case for sorting by transpositions; in [8] the authors show permutations in S_{13} and S_{15} with transposition distance greater than $\texttt{rev}_{13}$ and $\texttt{rev}_{15}$, respectively.

3 Block Merging

We introduce a new problem BLOCK-MERGING which will be used to obtain the 2-approximation for BLOCK-SORTING. The input to this problem is a multiset $\mathbb{S} = \{S_1, S_2, \ldots, S_k\}$ of disjoint increasing sequences[3] whose union is $[n]$. That is, each S_i is an increasing sequence, and $S_1 S_2 \ldots S_k \in S_n$. (Note that $\mathbb{S}$ is a multiset only because more than one sequence in it could be empty. The non-empty sequences in $\mathbb{S}$ form not just a multiset but a set. While empty sequences can be removed from $\mathbb{S}$ without changing the complexity of the instance, we prefer

[3]If S and T are two sequences, we denote by ST the sequence obtained by concatenating S and T. We denote the empty sequence by ϵ.

to explicitly allow them to simplify notation later on.) The goal is to transform $\mathbb{S}$ to the multiset $\mathbb{M}_n = \{\mathtt{id}_n, \epsilon, \ldots, \epsilon\}$. The moves allowed are of the form: Pick a block from any sequence S_i, and insert it into some other sequence S_j so that it merges with a block there. (Thus the new sequence S'_j is again an increasing sequence.) Here, a block is defined as a maximal substring of some S_i which is also a substring of $\mathtt{id}_n$.

The BLOCK-MERGING problem is the problem of finding the smallest number k such that $\mathbb{S}$ can be transformed into $\mathbb{M}_n$ with k moves. This smallest number of required moves is denoted $bm(\mathbb{S})$.

Let $\mathtt{\#block}(\mathbb{S})$ denote the number of blocks in $\mathbb{S}$. It is easy to see that any block move in a block-move-merging sequence can reduce the number of blocks by at most 2. But $\mathtt{\#block}(\mathbb{S}) - 1$ block moves suffice: repeatedly move the block containing 1 to its successor. Thus, similar to Proposition 2.2, we have

Proposition 3.1

$$\left\lceil \frac{\mathtt{\#block}(\mathbb{S}) - 1}{2} \right\rceil \leq bm(\mathbb{S}) \leq \mathtt{\#block}(\mathbb{S}) - 1$$

The examples below show that these inequalities can be strict and that the bounds are tight.

Example 3.2

1. *For $n = 2k$, let $\mathbb{S}_k$ be the set $\{S_1, S_2, \ldots, S_k\}$ where $S_i = (i, 2k+1-i)$. For instance, with $n = 6$, $\mathbb{S}_3 = \{(1,6), (2,5), (3,4)\}$. This has $2k-1$ blocks and can be merge-sorted with $k-1$ moves: keep moving the last non-empty sequence (which will be a single block) into the previous sequence. It cannot be merged with any fewer moves since there are k non-empty sequences to begin with. Hence $bm(\mathbb{S}_k) = k-1$, showing that the lower bound is tight.*

2. *For every n, let $\mathbb{S}_n$ be the set of n singleton sequences. e.g. $\mathbb{S}_4 = \{(1), (2), (3), (4)\}$. Merging this needs $n-1$ moves, showing that the upper bound is tight.*

3. *For $n = 2k$, let $\mathbb{S}_k$ consist of k sequences, where the ith sequence contains exactly $i, k+i$. For instance, with $n = 6$, $\mathbb{S}_3 = \{(1,4), (2,5), (3,6)\}$. Using techniques from Section 4, we can show that $bm(\mathbb{S}_k) = 2k-2$, showing that both inequalities can be strict.*

A transposition on an instance $\mathbb{S}$ of BLOCK-MERGING is a move that picks up any subsequence and places it in another sequence in such a way that the resulting set of sequences is a legal instance of BLOCK-MERGING (all sequences are increasing). We will use such moves in the analysis in Section 4.

The following theorems are the main results of this paper. The next two sections are devoted to proving Theorems 3.3 and 3.4 respectively.

Theorem 3.3 BLOCK-MERGING *is in P.*

Theorem 3.4 *Given an instance π of* BLOCK-SORTING*, in $O(n)$ time we can construct an instance $\mathbb{S}$ of* BLOCK-MERGING *such that $bs(\pi) \leq bm(\mathbb{S}) \leq 2bs(\pi)$.*

From these two theorems, we conclude:

Theorem 3.5 BLOCK-SORTING *has a polynomial time 2-approximation algorithm.*

Since all the proofs in this paper are (efficiently) constructive, we also have a constructive version of Theorem 3.5: in polynomial time, we can find a block move sorting sequence whose length is at most twice the optimal.

4 Optimal Block Merging is in P

We present a dynamic programming algorithm to compute $bm(\mathbb{S})$ exactly. The strategy is to construct a related graph, and identify a substructure in that graph corresponding to an optimal merging sequence. The substructure defined below is shown to be related to $bm(\mathbb{S})$ in Lemma 4.4, and is shown to be efficiently computable in Lemma 4.7. Theorem 3.3 follows from these lemmas.

Given an instance $\mathbb{S}$ of BLOCK-MERGING, we build an associated directed graph G as follows: G has n vertices numbered 1 through n. The multiset $\mathbb{S}$ having k sequences partitions the vertices into k parts, and the edges of G form a total order on each part. Formally,

Definition 4.1 *Let $\mathbb{S} = \{S_1, S_2, \ldots, S_k\}$. Then*

1. *$G = (V, E)$ where $V = [n]$, and $E = \{(u, v) \mid u < v, \quad \exists p \in [k] : \; u, v \in S_p\}$.*
2. *Edges in E of the form $(i, i+1)$ are called unit edges.*
3. *For $1 \leq i \leq j \leq n$, $G([i, j])$ denotes the subgraph of G induced by the vertex set $\{i, i+1, \ldots, j\}$, and its edge set is denoted $E([i, j])$.*
4. *Edges (i, j) and (k, l) are said to* cross *if $i \leq k < j \leq l$ or $k \leq i < l \leq j$. A set $E' \subseteq E$ is said to be a non-crossing set if no two edges in E' cross.*
   (Note that (i, j) and (i, k) cross, but not (i, j) and (k, l) where $i < k < l < j$.)
5. *By $c(i, j)$ we denote the size of the largest non-crossing set in $G([i, j])$. We denote $c(1, n)$ by $c(\mathbb{S})$.*

For instance, Figure 1 shows the directed graph associated with the instance
$\mathbb{S} = \{(8), (2\ 5\ 6), (3\ 9), (1\ 4\ 7)\}$.

In this example, edges $(2, 5)$ and $(1, 4)$ cross, as also edges $(2, 5)$ and $(2, 6)$. But edges $(5, 6)$ and $(3, 9)$ do not, neither do $(1, 4)$ and $(4, 7)$. The set $\{(3, 9), (4, 7), (5, 6)\}$ is a largest non-crossing set here, as also the sets $\{(2, 5), (5, 6), (1, 7)\}$ and $\{(1, 4), (4, 7), (5, 6)\}$.

Proposition 4.2 *For every $1 \leq i < j \leq n$, any largest non-crossing set in $G([i, j])$ contains all the unit edges of $G([i, j])$.*

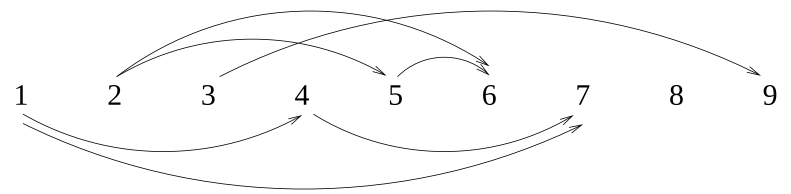

Figure 1: The graph of $\mathbb{S} = \{(8), (2\ 5\ 6), (3\ 9), (1\ 4\ 7)\}$

Proof: Let C be a largest non-crossing set in $G([i,j])$. Assume that some unit edge $(p, p+1) \in E([i,j])$ is not in C; then it must cross some edge in C. Assume that it crosses an edge $(p,q) \in C$, where $q > p+1$. Then C cannot have any edge of the form (p,r) where $r > p+1, r \neq q$, since such an edge crosses (p,q). Also, C cannot have any edge of the form $(r, p+1)$ where $r < p$, since such an edge also crosses (p,q). Note that $(p,q) \in C$ and $(p.p+1) \in E$ means that $p, p+1, q$ are in the same sequence, so $(p+1, q) \in E$ as well. Now $C \setminus \{(p,q)\} \cup \{(p,p+1),(p+1,q)\}$ is a larger non-crossing set in $G([i,j])$, a contradiction. Similarly, we can show that $(p,p+1)$ does not cross any edge $(r,p+1) \in C$, where $r < p$. By definition, $(p,p+1)$ does not cross any other type of edge. So C must contain $(p,p+1)$. □

It follows that $c(\mathbb{M}_n) = n-1$ and that $c(\mathbb{S}) < n-1$ for all $\mathbb{S} \neq \mathbb{M}_n$.

Lemma 4.3 *Let $\mathbb{S} \neq \mathbb{M}_n$. Then there exists a block move which, when applied to $\mathbb{S}$, results in $\mathbb{S}'$ with $c(\mathbb{S}') > c(\mathbb{S})$.*

Proof: Let C be a largest non-crossing set in G.

If C contains only unit edges, then moving any block to its successor or predecessor will create one more unit edge and thus increase the non-crossing set size. (Since $\mathbb{S} \neq \mathbb{M}_n$, there are at least two blocks in $\mathbb{S}$.) Essentially, all blocks in $\mathbb{S}$ are *free* in C: we say that a block is free in C if no edge of C has one endpoint in this block and the other endpoint in a different block.

So now assume that C contains some non-unit edges as well. Find a non-unit edge $(i,j) \in C$ which does not *contain* any other non-unit edge. (Edge (i,j) contains edge (k,l) if $i < k < l < j$.) Clearly, such an edge must exist. By definition of non-crossing set, $(l, i+1) \notin C$ for any $l \leq i$. Then from Proposition 4.2, $(i, i+1) \notin E$. Also, if k is the last element of the block containing $i+1$, then C has no edge of the form (k,l), since any such edge would have to have $l < j$ to avoid crossing, and it would be a non-unit edge contained in (i,j), contradicting our choice of non-unit edge (i,j). In fact, the block containing $i+1$ is *free* in C. See Figure 2.

Perform the block move "move $i+1$ to predecessor", giving $\mathbb{S}'$ with digraph G'. Now $C \setminus \{(i,j)\} \cup \{(i,i+1),(k,j)\}$ is a non-crossing set in $\mathbb{S}'$, of size $|C|+1$, and so $c(\mathbb{S}') \geq c(\mathbb{S})+1$. □

Lemma 4.4 $bm(\mathbb{S}) = n - 1 - c(\mathbb{S})$.

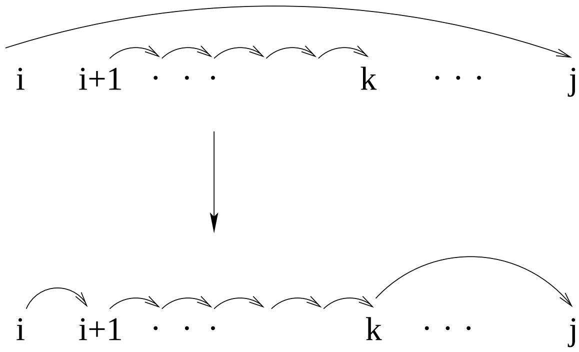

Figure 2: Moving a free block increases non-crossing set size

Proof: We first prove the following claims:

Claim 4.5 *Let $\mathbb{S}_1$ be an instance of* BLOCK-MERGING*, and let ρ be a transposition that moves a block or a substring of a block to any other location to give a legal instance $\mathbb{S}_2$ of* BLOCK-MERGING*. Then $c(\mathbb{S}_1) \leq c(\mathbb{S}_2) + 1$.*

Proof of Claim: Let $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ respectively denote the digraphs associated with $\mathbb{S}_1$ and $\mathbb{S}_2$. Let ρ move a (sub)block $a, \ldots, b$.

Let C_1 be a largest non-crossing set in G_1. Partition C_1 as $A_1 = C_1 \setminus E_2$, $A_2 = C_1 \cap E_2$. Since C_1 has all unit edges of E_1, A_1 can have at most two edges, one of the form (i, a) and one of the form (b, j). If $|A_1| \leq 1$, then A_2 is a non-crossing set in G_2 of size at least $|C_1| - 1$. Otherwise, $C_2 = A_2 \cup \{(i, j)\}$ is a non-crossing set in G_2 of size $|C_1| - 1$. Thus in either case $c(\mathbb{S}_2) \geq c(\mathbb{S}_1) - 1$. $\Box$(*Claim*)

Claim 4.6 *Let $\mathbb{S}$ be an instance of* BLOCK-MERGING*, and let ρ be a block move giving the* BLOCK-MERGING *instance $\mathbb{S}'$. Then $c(\mathbb{S}') \leq c(\mathbb{S}) + 1$.*

Proof of Claim: Suppose ρ moves the block x. Then moving x back to its original place in $\mathbb{S}$ is a transposition from $\mathbb{S}'$ to $\mathbb{S}$. By the preceding claim, $c(\mathbb{S}') \leq c(\mathbb{S}) + 1$. $\Box$(*Claim*)

To prove $bm(\mathbb{S}) \geq n-1-c(\mathbb{S})$**:** Let $bm(\mathbb{S}) = m$ and let $\rho_1, \ldots, \rho_m$ be an optimal block-move merging sequence for $\mathbb{S}$. Applying these moves sequentially to $\mathbb{S}$ gives $\mathbb{S}_1, \ldots, \mathbb{S}_m = \mathbb{M}_n$. By the above claim, $c(\mathbb{S}_m) \leq c(\mathbb{S}_{m-1}) + 1 \leq \ldots \leq c(\mathbb{S}) + m$. But $c(\mathbb{S}_m) = n - 1$. Hence $bm(\mathbb{S}) = m \geq n - 1 - c(\mathbb{S})$.

To prove $bm(\mathbb{S}) \leq n - 1 - c(\mathbb{S})$**:** The proof proceeds by induction on $r = n - 1 - c(\mathbb{S})$. The basis case is clear, since $r = 0 \iff \mathbb{S} = \mathbb{M}_n$. For the inductive step, assume the statement is true upto $r - 1$, and let C be a largest non-crossing set in G, of size $n - 1 - r$. Since $r > 0$, $|C| < n - 1$ and so $\mathbb{S} \neq \mathbb{M}_n$. By Lemma 4.3, we can find a block move leading to $\mathbb{S}'$ with $c(\mathbb{S}') \geq c(\mathbb{S}) + 1$. But by the preceding claim, $c(\mathbb{S}') \leq c(\mathbb{S}) + 1$. Hence $c(\mathbb{S}') = c(\mathbb{S}) + 1 = n - 1 - (r - 1)$. Using the induction hypothesis, $bm(\mathbb{S}) \leq 1 + bm(\mathbb{S}') \leq 1 + (n - 1 - c(\mathbb{S}')) = n - c(\mathbb{S}) - 1$. $\Box$

Note that via the above proof, we don't just compute $bm(\mathbb{S})$ from $c(\mathbb{S})$; we actually obtain an optimal merging sequence from a largest noncrossing set C. It now remains to compute $c(\mathbb{S})$ and to construct a C of this size.

Lemma 4.7 *$c(\mathbb{S})$ can be computed in $O(n^3)$ time.*

Proof: Let $t(i,j)$ denote the size of the largest non-crossing set in $G([i,j])$ that uses the edge (i,j), and let $q(i,j)$ denote the size of the largest non-crossing set in $G([i,j])$ that does not use the edge (i,j). We present recurrence relations for the $c(i,j)$'s, $t(i,j)$'s and $q(i,j)$'s, which directly lead to an $O(n^3)$ time dynamic programming algorithm for computing $c(1,n)$. The recurrences are as follows:

$$\begin{array}{rrl} \text{For } i \in [n], & c(i,i) = & 0 \\ \text{For } i \in [n-1], & c(i,i+1) = & \begin{cases} 1 & \text{if } (i,i+1) \in E \\ 0 & \text{otherwise} \end{cases} \\ \text{For } 1 \le i \le j-2 \le n-2 & c(i,j) = & \max\{t(i,j), q(i,j)\} \end{array}$$

$$\begin{array}{rrl} \text{For } 1 \le i \le j-2 \le n-2 & t(i,j) = & \begin{cases} 1 + c(i+1,j-1) & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases} \\ & q(i,j) = & \max\limits_{i<k<j} c(i,k) + c(k,j) \end{array}$$

Let us see why these recurrences hold.
The recurrence for $c(i,j)$: Follows directly from the definition.
The recurrence for $t(i,j)$: If $(i,j) \notin E$, then $t(i,j) = 0$. Otherwise, if (i,j) is in a non-crossing set C, then C cannot have any other edge incident on i or j. For C to be a largest such set, $C \setminus \{(i,j)\}$ must be a largest non-crossing set in $G([i+1,j-1])$, giving the claimed recurrence.
The recurrence for $q(i,j)$: Clearly, $q(i,j) \ge \max_{i<k<j} c(i,k) + c(k,j)$. To see the other direction, we show how to find a k such that $q(i,j) = c(i,k) + c(k,j)$. Let $E'([i,j]) = E([i,j]) \setminus \{(i,j)\}$, and let C be a largest non-crossing subset of $E'([i,j])$, so that $|C| = q(i,j)$. By definition of crossing edges, C has at most one edge incident on i.
Case 1: Some edge (i,k) incident on i is present in C. Let $A = C \cap E([i,k])$ and $B = C \cap E([k,j])$. Then A and B are disjoint, and C must be equal to $A \cup B$. Now A is a non-crossing set in $G([i,k])$, and it must be a largest such set. Similarly, B must be a largest non-crossing set in $G([k,j])$. Thus $q(i,j) = |C| = |A| + |B| = c(i,k) + c(k,j)$.
Case 2: In C, i is isolated. Then $(i,i+1) \notin E'([i,j])$ and $C \subseteq E([i+1,j])$. Hence, $c(i,i+1) = 0$ and $c(i,j) = c(i,i+1) + c(i+1,j)$. □

We note that a noncrossing set of size $c(\mathbb{S})$ (and thus a merging sequence of length $bm(\mathbb{S}) = n - 1 - c(\mathbb{S})$) itself can be computed by storing some additional information in the dynamic programming algorithm.

For completeness, the complete algorithm for block merging is presented below.

Input: An instance $\mathbb{S} = \{S_1, S_2, \ldots, S_k\}$.

1. Construct the graph G corresponding to $\mathbb{S}$, as described in Definition 4.1.
2. Compute a largest non-crossing set C in G.
   (In the dynamic programming algorithm of Lemma 4.7, carry forward not just the values of $c(i,j)$, $t(i,j)$ and $q(i,j)$ but also a set each witnessing these values.)
3. **While** $\mathbb{S} \neq \mathbb{M}_n$ **Do**

   Find a block α in $\mathbb{S}$ free with respect to C.
   (The proof of Lemma 4.3 shows that such a block always exists. One can be found by direct inspection of each block with respect to C.)
   Let $i = \texttt{first}(\alpha, \mathbb{S})$ and $j = \texttt{last}(\alpha, \mathbb{S})$.
   If $j < n$ **Then**
   Move α to successor, to get the set of sequences $\mathbb{S}'$.
   If C has an edge $(l, j+1)$ for some l
   Then $C' = C \cup \{(l,i)\} \setminus \{(l, j+1)\}$
   EndIf
   $C' = C \cup \{(j, j+1)\}$.
   Else
   Move α to predecessor, to get the set of sequences $\mathbb{S}'$.
   $C' = C \cup \{(i-1, i)\}$.
   EndIf
   $\mathbb{S} \longleftarrow \mathbb{S}'$, $C \longleftarrow C'$.

   EndWhile

Figure 3: The algorithm for BLOCK-MERGING

5 Block Merging can approximate Block Sorting

Any permutation π can be uniquely decomposed into maximal increasing substrings. Considering each of these substrings as a sequence gives an instance $\mathbb{S}_\pi$ of BLOCK-MERGING. We show that $bm(\mathbb{S}_\pi)$ instance provides an approximation for $bs(\pi)$.

Definition 5.1 *Let S be any non-empty sequence $a_1, a_2, \ldots, a_t$ of distinct elements. The set $\mathbb{S}_S$ of increasing sequences which are substrings of S is constructed as follows. Let l be the maximum number such that $a_1 \leq a_2 \leq \ldots \leq a_{l-1} \leq a_l$. If $l = t$, then $\mathbb{S}_S = \{S\}$. Otherwise, $\mathbb{S}_S = \{S_1\} \cup \mathbb{S}_{S'}$, where S_1 is the sequence $\langle a_1, a_2, \ldots, a_l \rangle$, and $S = S_1 S'$.*

In particular, when S is a permutation, then $\mathbb{S}_S$ is an instance of BLOCK-MERGING. For instance, when $\pi = 8\ 2\ 5\ 6\ 3\ 9\ 1\ 4\ 7$, $\mathbb{S}_\pi$ is the instance $\{(8), (2\ 5\ 6), (3\ 9), (1\ 4\ 7)\}$ shown in Figure 1.

The examples below show that $bm(\mathbb{S}_\pi)$ and $bs(\pi)$ can behave quite differently. Nonetheless, we show in Lemmas 5.3 and 5.4 that $bm(\mathbb{S}_\pi)$ is in fact a 2-approximation for $bs(\pi)$. Theorem 3.4 follows from these lemmas.

Example 5.2 *(a) Consider the permutation π on $2n$ elements where $\pi_{2k-1} = k$ and $\pi_{2k} = n+k$. For instance, when $n = 4$, then $\pi = 1\ 5\ 2\ 6\ 3\ 7\ 4\ 8$, and $\mathbb{S}_\pi = \{(1,5),(2,6),(3,7),(4,8)\}$. Then it can be shown that $bs(\pi) = n-1$, whereas $bm(\mathbb{S}_\pi) = 2n-2$.*
(b) For $\pi = \texttt{rev}_n$, $bs(\pi) = bm(\mathbb{S}_\pi) = n-1$.

Lemma 5.3 *$bm(\mathbb{S}_\pi) \le 2bs(\pi)$.*

Proof: Given any block move sorting sequence for π of length l, we will show that there is a block move merging sequence for $\mathbb{S}_\pi$ with at most $2l$ moves. Let $\rho_1, \rho_2, \ldots, \rho_l$ be a sequence of block moves sorting π. Let $\pi^0 = \pi$, and let π^{i+1} result from applying block move ρ_{i+1} to π^i. Consider a direct simulation of these moves on $\mathbb{S}_\pi$. The first move can always be performed. At later stages, a block being moved in the permutation may not be a block in the corresponding set of sequences but may consist of several fragments. At the end too, when $\pi^l = \texttt{id}_n$, we may have a set of single-block sequences instead of $\mathbb{M}_n$, and the blocks have to be collected to give $\mathbb{M}_n$.

To handle the first problem described above, we collect the fragments through a series of block moves, and then move the built block. For the second problem, at the end, we move the non-empty single-block sequences into one sequence. Thus we make l moves corresponding to the ρ_is, and some extra moves to collect fragments, and some extra moves at the end to merge blocks. Every time we collect more than 2 fragments to form a bigger block, we empty out some sequence, which saves a move in the final merging step. A careful analysis shows that the number of extra moves required does not exceed l.

See the example in Figure 4. On the left, a block move sorting sequence for π with 4 moves is given; the block to be moved is underlined. On the right, the simulation on $\mathbb{S}_\pi$ is illustrated, with | used to denote the boundaries between sequences. Simulating move 2 directly is not possible, since the moved block 3 4 has two pieces on the right. So first the block is built up, and then it is moved. Similarly, after simulating all 5 moves, the instance on the right is not yet merged and needs one extra move of block 10.

Formally, let $\mathbb{S}_\pi = \langle S_1, S_2, \ldots, S_k \rangle$ be denoted by $\mathbb{S}_0$. In the first l stages, we describe how to construct, for each $i < l$, an instance $\mathbb{S}_{i+1}$ of BLOCK-MERGING from the instance $\mathbb{S}_i$. ($\mathbb{S}_i$ is *not* necessarily $\mathbb{S}_{\pi^i}$.) Let $\mathbb{S}_i = \{S_1^i, S_2^i, \ldots, S_k^i\}$, where we list the sequences in the same order (that is, for each $1 \le j \le k$, S_j^i is the sequence obtained from S_j^{i-1} after performing move ρ_i).

We denote by $\texttt{Inc}(\pi)$ the set $\{a \in [n-1] \mid \texttt{next}(a,\pi) > a\}$. Clearly, $\texttt{Inc}(\texttt{id}_n) = [n-1]$. For $a \in [n]$ and BLOCK-MERGING instance $\mathbb{S}$, we denote by $\texttt{Inc}(\mathbb{S})$ the set $\{a \in [n-1] \mid a \text{ is not the last element of any sequence of } \mathbb{S}\}$. We have $\texttt{Inc}(\mathbb{M}_n) = [n-1]$. For instance, for the permutation $\pi = 7\ 6\ 5\ 2\ 3\ 4\ 8\ 1$, the set $\texttt{Inc}(\pi)$ is $\{2,3,4\}$ while the set $\texttt{Inc}(\mathbb{S}_\pi)$ is $\{2,3\}$.

$$
\begin{array}{cccccccccc}
3 & \underline{1} & 4 & 6 & 2 & 5 & 7 & 9 & 8 & 10 \\
\underline{3} & \underline{4} & 6 & 1 & 2 & 5 & 7 & 9 & 8 & 10 \\
 & & & & & & & & & \\
\underline{6} & 1 & 2 & 3 & 4 & 5 & 7 & 9 & 8 & 10 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 9 & \underline{8} & 10 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10
\end{array}
$$

(a) A block move sorting of π

$$
\begin{array}{rrrrrrrrrrr}
 & 3| & 1 & 4 & 6| & 2 & 5 & 7 & 9| & 8 & 10 \\
 & 3| & 4 & 6| & 1 & 2 & 5 & 7 & 9| & 8 & 10 \\
 & 3 & 4| & 6| & 1 & 2 & 5 & 7 & 9| & 8 & 10 \\
| & 6| & 1 & 2 & 3 & 4 & 5 & 7 & 9| & 8 & 10 \\
|| & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 9| & 8 & 10 \\
|| & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9| & 10 \\
|| & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10|
\end{array}
$$

(b) The simulated block move merging of $\mathbb{S}_\pi$

Figure 4: Simulating a block move sorting sequence by a merging sequence

We will maintain the following invariants, which are clearly true for $i = 0$:

$$\forall i \qquad \pi^i \quad = S_1^i S_2^i \dots S_k^i \tag{1}$$

$$\forall i \qquad \texttt{Inc}(\mathbb{S}^i) \quad \subseteq \texttt{Inc}(\pi^i) \tag{2}$$

Let ρ_{i+1} move a block x of π^i to obtain π^{i+1}. We simulate ρ_{i+1} by performing a defragmentation phase followed by a block move. If x is already a block in $\mathbb{S}_i$, then the defragmentation phase is empty. Otherwise, due to the invariants we maintain, there are indices $u < v$ such that $x = \langle\text{last block of } S_u^i\rangle S_{u+1}^i ... S_{v-1}^i \langle\text{first block of } S_v^i\rangle$. The defragmentation phase consists of at most $v - u$ moves which collect these fragments, in the same relative order, into the sequence S_u^i. This gives the $\mathsf{BLOCK\text{-}MERGING}$ instance $\mathbb{S}_i'$ in which x is a block. Clearly, composing the sequences of $\mathbb{S}_i'$ still gives π^i.

If ρ_{i+1} moves x to its predecessor, do the same in $\mathbb{S}_i'$ to obtain $\mathbb{S}_{i+1}$; otherwise, if ρ_{i+1} moves x to its successor, do the same to obtain $\mathbb{S}_{i+1}$. (Thus, if ρ_{i+1} moves x to between its predecessor and successor, and these are in different sequences of $\mathbb{S}_i'$, then we move x to the sequence containing its predecessor.) It is clear that this move maintains the invariants of Equations 1 and 2.

Since $S_1^l S_2^l \dots S_k^l = \pi^l = \texttt{id}_n$, each non-empty sequence in $\mathbb{S}_l$ must be a block. In the last $(l+1)$st stage, we repeatedly move the block containing 1 to its successor block until $\mathbb{M}_n$ is obtained.

It remains to analyze the length of this merging sequence. We bound this by $2l$ by choosing a suitable potential function and performing an amortized analysis of each stage.

Let c_i denote the cost of the ith phase. For $1 \leq i \leq l$, c_i is of the form $1 + t_i$, where t_i is the number of moves required in the defragmentation phase of the ith stage. And $c_{l+1} = \texttt{\#block}(\mathbb{S}^l) - 1$. Further, let P_i and Q_i denote the sets $\texttt{Inc}(\pi^i)$ and $\texttt{Inc}(\mathbb{S}^i)$, with sizes p_i, q_i, respectively. From Equation 2, $p_i \geq q_i$. We define the potential function ϕ_i to be $p_i - q_i$. Note that $\phi_0 = \phi_{l+1} = 0$, and $\phi_l = \texttt{\#block}(\mathbb{S}^l) - 1$. Now, the amortized cost of the ith stage, a_i, is given by $a_i = c_i + \phi_i - \phi_{i-1}$. Since $\phi_0 = \phi_{l+1} = 0$, the total real cost $\sum_{i=1}^{l+1} c_i$ equals the total amortized cost $\sum_{i=1}^{l+1} a_i$. Clearly, $a_{l+1} = 0$. We now show that all the remaining phases have an amortized cost of at most 2. That is, for $1 \leq i \leq l$, $a_i \leq 2$. We show this by arguing that $\phi_i - \phi_{i-1} \leq 1 - t_i$. Note that $\phi_i - \phi_{i-1} = (p_i - q_i) - (p_{i-1} - q_{i-1}) = (p_i - p_{i-1}) - (q_i - q_{i-1})$.

Let ρ_i move a block x with $\mathtt{first}(x, \pi^{i-1}) = b$ and $\mathtt{last}(x, \pi^{i-1}) = c$, say, to its predecessor element e; the other case is similar. Let $\mathtt{prev}(b, \pi^{i-1}) = a$ if defined. It is clear that $e \in P_i$. Furthermore, $c \in P_i$ if and only if $e \in P_{i-1}$. Thus the move ρ contributes a 1 to $p_i - p_{i-1}$ if neither a nor c is in P_{i-1} *or* if exactly one of a and c is in P_{i-1} and a is in P_i. In all other cases, it contributes a 0 to $p_i - p_{i-1}$.

Let us consider the moves made in $\mathbb{S}_{i-1}$ to simulate the move ρ_i. The t_i defragmentation moves made here each add an element to Q_i. Therefore, if the last fragment moved is not part of a strictly larger sequence, or equivalently if $c \notin Q_{i-1}$, there is a contribution of t_i to $q_i - q_{i-1}$ due to the defragmentation phase; otherwise it is $t_i - 1$. Moreover, moving the defragmented x to the immediate right of e adds e to Q_i if $e \notin Q_{i-1}$ and adds c to Q_i if $e \in Q_{i-1}$. Therefore, if the defragmented x does not appear as a part of a strictly larger sequence, or equivalently if $a \notin Q_{i-1}$, this move contributes a 1 to $q_i - q_{i-1}$; otherwise it is 0.

In conclusion, the move ρ on π^{i-1} contributes 1 or 0 to $p_i - p_{i-1}$ and the corresponding simulation moves on $\mathbb{S}^{i-1}$ contribute t_i or $t_i - 1$ to $q_i - q_{i-1}$; the latter is true only when both a and c are in Q_{i-1}, in which case they are both in P_{i-1} as well, and so $p_i - p_{i-1} = 0$. Thus $\phi_i - \phi_{i-1} \leq 1 - t_i$ for every $i, 1 \leq i \leq l$.

□

Lemma 5.4 $bs(\pi) \leq bm(\mathbb{S}_\pi)$.

Proof: Let C be a maximum non-crossing set in the digraph associated with $\mathbb{S}_\pi$. From Lemma 4.4, $|C| = n - 1 - bm(\mathbb{S}_\pi)$. We show how, with each carefully chosen block move, we can push up the size of a corresponding non-crossing set by at least 1. Since it starts at $n - 1 - bm(\mathbb{S}_\pi)$ and stops at $n - 1$, at most $bm(\mathbb{S}_\pi)$ moves will suffice. This is essentially what is done in the algorithm of Figure 3.

Formally, we prove this by induction on $k = bm(\mathbb{S}_\pi)$. The base case, when $k = 0$, is trivially true. Assume that the lemma holds for all π' with $bm(\mathbb{S}_{\pi'}) < k$, and now let $bm(\mathbb{S}_\pi) = k$. Let C be a maximum non-crossing set in the digraph associated with $\mathbb{S}_\pi$. Since $k > 0$, we know that $\mathbb{S} \neq \mathbb{M}_n$. So Lemma 4.3 allows us to identify a block of $\mathbb{S}_\pi$, and hence of π, that is free with respect to C. Moving this block to either its successor or its predecessor transforms π to, say, σ, and $\mathbb{S}_\pi$ to $\mathbb{S}'$. (Note that $\mathbb{S}_\sigma$ is not necessarily $\mathbb{S}'$. For example, let $\pi = 1\ 3\ 2\ 4$ so that $\mathbb{S}_\pi = \{(1,3),(2,4)\}$. If we move the block 2 in π we get $\sigma = \mathtt{id}_4$ and $\mathbb{S}_\sigma = \mathbb{M}_4$, but the same block move in $\mathbb{S}_\pi$ gives $\{(1,2,3),(4)\}$.) By Lemma 4.3 and Lemma 4.4, $bm(\mathbb{S}') \leq bm(\mathbb{S}_\pi) - 1$. But observe that every edge in the graph associated with $\mathbb{S}'$ is also an edge in $\mathbb{S}_\sigma$. Thus $c(\mathbb{S}_\sigma) \geq c(\mathbb{S}')$ and hence $bm(\mathbb{S}_\sigma) \leq bm(\mathbb{S}') \leq k - 1$. It now follows from the induction hypothesis that $bs(\pi) \leq 1 + bs(\sigma) \leq 1 + bm(\mathbb{S}_\sigma) \leq 1 + (k-1) = k$.

□

To actually construct a sequence of block moves at most twice as long as the optimal, it suffices to run the algorithm of Figure 3 on $\mathbb{S}_\pi$. (Some of the moves may turn out to be redundant.)

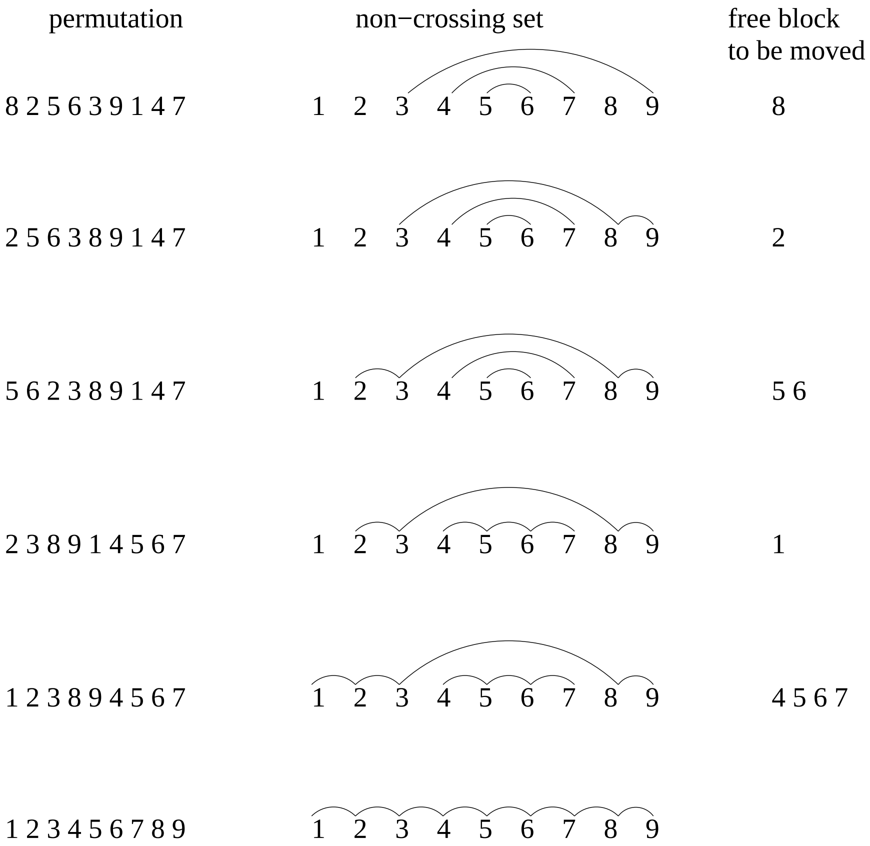

Figure 5: Increasing the non-crossing-set size via moves of free blocks

For example, when $\pi = 8\ 2\ 5\ 6\ 3\ 9\ 1\ 4\ 7$, $\mathbb{S}_\pi = \{(8), (2\ 5\ 6), (3\ 9), (1\ 4\ 7)\}$, and the associated digraph is shown in Figure 1. Now $c(\mathbb{S}_\pi) = 3$ so $bm(\mathbb{S}_\pi) = 9 - 1 - c(\mathbb{S}_\pi) = 5$. Consider the maximum non-crossing set $C = \{(3, 9), (4, 7), (5, 6)\}$. Figure 5 illustrates how block moves are chosen to increase the size of C with every move, so that block sorting in $n - 1 - |C|$ moves is achieved. (For simplicity, the arcs are drawn as undirected edges.)

6 Equivalent Relaxations of Block Sorting and Block Merging

We introduce some natural relaxations of BLOCK-SORTING and BLOCK-MERGING, and show that they are equivalent to the original problems.

Replacing each block of a permutation by a single representative element gives the kernel. Formally, the kernel of π, denoted $\texttt{ker}(\pi)$, is the permutation σ obtained by replacing each block α by the rank of $\texttt{first}(\alpha, \pi)$ in $\{\texttt{first}(\beta, \pi) \mid \beta \text{ is a block of } \pi\}$, where for $a \in S$, rank of a denotes $|\{b \in S \mid b \leq a\}|$. For example, for the permutation $\pi = 7\ 6\ 5\ 2\ 3\ 4\ 8\ 1$, $\texttt{ker}(\pi) = 5\ 4\ 3\ 2\ 6\ 1$. Clearly, $bs(\pi) = bs(\texttt{ker}(\pi))$.

In [4], Christie shows that the following intuitive idea can be formally proved true: an optimal transpostion sorting sequence need never break apart an existing block (he called a block a strip), since the block would have to be put together again. His formal proof involved establishing that $t(\pi) = t(\mathtt{ker}(\pi))$.

We show analogous results for BLOCK-SORTING and BLOCK-MERGING. Suppose we relax the notion of a legal move: in a single move we can pick up any block and place it anywhere else (not necessarily next to its predecessor or successor). We call such a move a *strip move*. Let the length of the shortest strip-move sorting sequence be denoted $s(\pi)$. (This is the strip move distance referred to in [18, 19].) Suppose we relax it further: in a single move we can pick up any block or substring of a block, and place it anywhere else. We call such a move a *substrip move*. Let the length of the shortest such sorting sequence be denoted by $s'(\pi)$. Clearly $s'(\pi) \leq s(\pi) \leq bs(\pi)$. We show that these relaxations do not make sorting any easier; in fact $s'(\pi) = s(\pi) = bs(\pi)$. The proof requires a straightforward adaptation of the proof of [4]. (For an alternative graph-theoretic proof that $bs(\pi) = s(\pi)$, see [19].)

A (sub)strip move in π is said to be a *canonical (sub)strip move* if it places a (sub)strip x to the immediate right of $\mathtt{pred}(x, \pi)$ or to the immediate left of $\mathtt{succ}(x, \pi)$. (In particular, if the substrip x is neither a prefix nor a suffix of any strip of π, then x cannot be moved in a canonical substrip move.) Note that a canonical strip move is precisely a block move.

Lemma 6.1 $s(\pi) = s'(\pi)$. *Furthermore, every optimal substrip move sequence is actually a strip move sequence.*

Proof: We know that $s'(\pi) \leq s(\pi)$, so we need to prove that $s(\pi) \leq s'(\pi)$ and that optimal substrip move sequences move only strips. We first prove the following claim.

Claim 6.2 *No canonical substrip move increases the substrip move distance. That is, if applying a canonical substrip move to σ results in π, then $s'(\pi) \leq s'(\sigma)$.*

Proof of Claim: This proof follows from an adaptation of Christie's proof that $t(\mathtt{ker}(\pi)) \leq t(\pi)$ [4]. Let the canonical move applied to σ move substrip x to its predecessor (the move to successor case is symmetric). Consider a substrip sorting sequence $\rho_0, \rho_1, \ldots, \rho_s$, with $\sigma^0 = \sigma$, σ^{i+1} obtained by applying ρ_i to σ^i, and $\sigma^{s+1} = \mathtt{id}_n$. We mark the positions occupied by x in σ and π with a special symbol $\star$. Now perform the moves ρ_i on π^i starting with $\pi^0 = \pi$, with the proviso that the $\star$s are considered glued to $a = \mathtt{first}(x, \sigma) - 1$ and they only move with it. Thus if substrip move ρ_i moves a, move all the $\star$s with it; if ρ_i does not move a, do not move any $\star$. All non-$\star$ elements are moved exactly as in ρ_i. It is easy to see that the modified moves continue to be strip or substrip moves. The ρ sequence correctly sorts σ; in particular, it places all the non-$\star$ elements in correct relative order. Hence the corresponding sequence on π also places the non-$\star$ elements in correct relative order. The $\star$ elements are in correct relative order because they were glued to a. So the modified sequence correctly sorts π, showing that $s'(\pi) \leq s'(\sigma)$.

See the example below. On the left, a substrip move sorting sequence for σ with 3 moves is given. On the right, the corresponding sequence for π is given, where π is obtained by moving substrip 6 7 in σ to its successor. On the right, 6 and 7 are represented by $*$ and # respectively.

$$\begin{array}{lcl}
8\ 1\ \underline{4\ 5\ 6\ 7}\ 2\ 9\ 3 & \quad & 6*\ 7*\ 8\ 1\ \underline{4\ 5}\ 2\ 9\ 3 \\
\underline{8}\ 1\ 2\ 9\ 3\ 4\ 5\ 6\ 7 & & \underline{6*\ 7*\ 8}\ 1\ 2\ 9\ 3\ 4\ 5 \\
1\ 2\ \underline{8\ 9}\ 3\ 4\ 5\ 6\ 7 & & 1\ 2\ \underline{6*\ 7*\ 8\ 9}\ 3\ 4\ 5 \\
1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9 & & 1\ 2\ 3\ 4\ 5\ 6*\ 7*\ 8\ 9
\end{array}$$

$\Box$(*Claim*)

Let us see why this claim suffices to prove the lemma. We induct on $s'(\pi)$. The base case is trivial, and we now assume that the statement holds for all permutations σ with $s'(\sigma) \leq s$. Let $s'(\pi) = s + 1$, and consider an optimal substrip sorting sequence for π. Let the first move, ρ, move a substrip x of π and result in permutation σ, where $s'(\sigma) = s$. If x is a proper substrip, then let uxv be the strip of π containing x, where at least one of u, v, is not ϵ. Now reinserting x between u and v is a canonical substrip move applied to σ giving π. So by the claim above, $s'(\pi) \leq s'(\sigma) = s$, contradicting our assumption that $s'(\pi) = s + 1$.

Hence x must be a strip; i.e. the first move in any optimal substrip move sorting sequence for π must be a strip move, resulting in a permutation σ. By the induction hypothesis, $s(\sigma) = s$ and every optimal substrip move sorting sequence for σ moves only strips.

Thus in every optimal substrip move sorting sequence for π, the first move is a strip move by our claim, and the remaining moves are strip moves by induction; thus the entire sequence moves only strips. $\Box$

The above proof is constructive; given a substrip move sorting sequence of length l, we can efficiently find a strip move sorting sequence of length at most l.

Corollary 6.3 *No canonical (sub)strip move increases the strip move distance. That is, if applying a canonical substrip move to σ results in π, then $s(\pi) \leq s(\sigma)$.*

We now establish that making canonical strip moves, i.e. block moves, is good enough.

Lemma 6.4 $bs(\pi) = s(\pi) = s'(\pi)$.

Proof: It suffices to show that $bs(\pi) \leq s(\pi)$; we prove this by induction on $s = s(\pi)$. The base case is vacuously true. Now let π be a permutation with $s(\pi) = s + 1$, and let π' be the permutation resulting from applying the first move ρ of an optimal strip move sorting sequence to π. Clearly, $s(\pi') = s$. So by the induction hypothesis, π' can be sorted by a block move sorting sequence, say $\rho_1, \rho_2, \ldots, \rho_s$.

If ρ is itself a block move, there is nothing to prove. Otherwise, let ρ move a strip x to some non-canonical place. Then x must be a strip in π'. Let σ be the permutation obtained from π or from π' by moving x to its predecessor (if it exists, otherwise successor). Note that σ can be obtained from π' or π in a single block move. By Corollary 6.3, $s(\sigma) \leq s(\pi') = s$. So by the induction hypothesis, $bs(\sigma) = s(\sigma) \leq s$. By construction, $sc(\pi) \leq 1 + bs(\sigma) \leq 1 + s = s(\pi)$, completing the induction. $\Box$

Unlike in Lemma 6.1, it is not true that the moves of every optimal strip move sorting sequence must be canonical (block moves). For example, sorting 1 4 3 2 6 5 as 1 3 2 4 6 5 to 1 2 3 4 6 5 to 1 2 3 4 5 6 is optimal, but the first move is not canonical. However, the mere existence of optimal canonical strip move sequences gives the following result.

Corollary 6.5 $s(\pi) = s(\mathtt{ker}(\pi))$.

Analogously, consider the relaxation for BLOCK-MERGING. Suppose that in a single move we can pick up any block and place it in any other sequence (not necessarily next to its predecessor or successor), as long as the resulting new sequence is also increasing. Let the length of the shortest such merging sequence be denoted $m(\mathbb{S})$. Clearly, for every instance $\mathbb{S}$ of BLOCK-MERGING, $m(\mathbb{S}) \leq bm(\mathbb{S})$. We show that this is in fact an equality.

Lemma 6.6 *For every instance* $\mathbb{S}$ *of* BLOCK-MERGING, $bm(\mathbb{S}) = m(\mathbb{S})$.

Proof: We need to prove that $bm(\mathbb{S}) \leq m(\mathbb{S})$. We will show that $n - 1 - c(\mathbb{S}) \leq m(\mathbb{S})$; the result then follows from Lemma 4.4.

Observe that in the proof of Lemma 4.4, the second claim can easily be strengthened to the following:

Claim 6.7 *Let* $\mathbb{S}$ *be an instance of* BLOCK-MERGING, *and let* ρ *be a move that picks up a block and places it elsewhere to give a legal* BLOCK-MERGING *instance* $\mathbb{S}'$. *Then* $c(\mathbb{S}') \leq c(\mathbb{S}) + 1$.

This suffices to establish that $n - 1 - c(\mathbb{S}) \leq m(\mathbb{S})$. □

7 Conclusion

Our main contribution has been to present a polynomial time 2-approximation algorithm for BLOCK-SORTING, and to introduce and solve efficiently BLOCK-MERGING along the way.

There are several open questions. The most important ones are: (1) Can we obtain better approximation algorithms for BLOCK-SORTING? Is it APX-hard? and (2) How do block moves relate to transpositions? – Does sorting by transposition reduce to BLOCK-SORTING? Does $bs(\pi)$ approximate $t(\pi)$ to any factor better than 3?

Another set of questions arises in the fixed-parameter-tractability framework [6]. As described in Corollary 2.3, both BLOCK-SORTING and sorting by transpositions are in FPT. However, one could ask more meaningful questions by pitching the parameter above the guaranteed value. (Parameterizing above the guarantee was first proposed in [17] as a more meaningful way of parameterizing many natural problems.) Here, the guaranteed value is $\lceil (\mathtt{\#block}(\pi) - 1)/3 \rceil$. So the question to be asked is, given parameter k, can one decide if $bs(\pi)$ (or $t(\pi)$) is at most $\lceil (\mathtt{\#block}(\pi) - 1)/3 \rceil + k$ in time $f(k)n^{O(1)}$? We do not yet know the answer to this question.

References

[1] V. Bafna and P. Pevzner. Genome rearrangements and sorting by reversals. *SIAM Journal on Computing*, 25:272–289, 1996.

[2] V. Bafna and P. Pevzner. Sorting by transpositions. *SIAM Journal on Discrete Mathematics*, 11(2):224–240, 1998.

[3] W.W. Bein, L.L. Larmore, S. Latifi, and I.H. Sudborough. Block sorting is hard. *Intl. Jl. of Foundations of Computer Science*, 14:425–437, 2003.

[4] D. A. Christie. *Genome Rearrangement Problems*. PhD thesis, University of Glasgow, 1999.

[5] D.A. Christie and R. Irving. Sorting strings by reversals and by transpositions. *SIAM Journal on Discrete Mathematics*, 14(2):193–206, 2001.

[6] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, Heidelberg, Germany, 1997.

[7] N. Eriksen. $1+\epsilon$ approximation for sorting by reversals and transpositions. *Theoretical Computer Science*, 289:517–529, 2002.

[8] H. Erikson, K. Eriksson, J. Karlander, L. Svensson, and J. Wästlund. Sorting a bridge hand. *Discrete Mathematics*, 241:289–300, 2001.

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

[10] W. H. Gates and C. H. Papadimitriou. Bounds for sorting by prefix reversals. *Discrete Mathematics*, 27:47–57, 1979.

[11] Q. P. Gu, S. Peng, and H. Sudborough. A 2-approximation algorithm for genome rearrangements by reversals and transpositions. *Theoretical Computer Science*, 210(2):327–339, 1999.

[12] T. Hartman. A simpler 1.5 approximation algorithm for sorting by transpositions. In *Proceedings of 14th Annual Symposium on Combinatorial Pattern Matching, LNCS 2676*, pages 156–169. Springer-Verlag, 2003.

[13] L. S. Heath and J. P. C. Vergara. Sorting by bounded block moves. *Discrete Applied Mathematics*, 88:181–206, 1998.

[14] L. S. Heath and J. P. C. Vergara. Sorting by short block moves. *Algorithmica*, 28:323–352, 2000.

[15] J. Kanai, S.V. Rice, and T.A. Nartker. Automatic evaluation of OCR zoning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17:86–90, 1995.

[16] J. Kececioglu and D. Sankoff. Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement. *Algorithmica*, 13:180–210, 1995.

[17] M. Mahajan and V. Raman. Parametrizing Above Guaranteed Values: MaxSat and MaxCut. *Journal of Algorithms*, 31(2):335–354, 1999.

[18] M. Mahajan, R. Rama, V. Raman and S. Vijayakumar. Merging and Sorting by Strip Moves. In *Proceedings of the 23rd International Foundations of Software Technology and Theoretical Computer Science Conference (FSTTCS), LNCS 2914*. pages 314–325. Springer-Verlag, 2003.

[19] M. Mahajan, R. Rama, and S. Vijayakumar. Towards constructing optimal strip move sequences. In *Proceedings of the 10th International Computing and Combinatorics Conference (COCOON), LNCS 3106*. pages 33–42. Springer-Verlag, 2004.

[20] P. Pevzner. *Computational Molecular Biology: An Algorithmic Approach*. MIT Press, Cambridge, MA, USA, 2000.

[21] J. P. C. Vergara. *Sorting by Bounded Permutations*. Ph.D Thesis, Virginia Polytechnique Institute and State University, 1997.