

A Quadratic Time 2-Approximation Algorithm for Block Sorting

Wolfgang W. Bein ^{*} Lawrence L. Larmore [†] Linda Morales [‡]
I. Hal Sudborough [§]

Abstract

We give an approximation algorithm for the block sorting problem with an approximation ratio of 2 and run time $O(n^2)$. The approximation algorithm is based on the related concept of block deletion. We show that finding an optimum block deletion sequence can be done in $O(n^2)$ time, while block sorting is known to be \mathcal{NP} -hard. Block sorting has importance in connection with optical character recognition (OCR) and is related to transposition sorting in computational biology.

Keywords: Design and analysis of algorithms; approximation algorithms; block sorting; transposition sorting; optical character recognition.

1 Introduction

Define a *permutation* of length n to be a list $x = (x_1, \dots, x_n)$ consisting of the integers $\{1, \dots, n\}$ where each number occurs exactly once. For any $1 \leq i \leq j \leq n$ we denote the sublist of x that starts at position i and ends with position j by $x_{i\dots j}$. If $j < i$, let $x_{i\dots j}$ be the empty list. We call a nonempty maximal sublist of x consisting of consecutive increasing integers, such as $(3, 4, 5)$ in $(6, 3, 4, 5, 2, 1)$, a *block* of x .

The *block sorting problem* is to find a minimum length sequence of *block moves* which sorts a given permutation x , where a block move consists of moving one block to a different position in the list. For example, the list $(4, 2, 5, 1, 3)$ can be block sorted in two moves: first move (2) to obtain $(4, 5, 1, 2, 3)$, then move $(4, 5)$ to obtain $(1, 2, 3, 4, 5)$.

Sorting problems under various operations have been studied extensively. We mention work on sorting with prefix reversals [4, 5, 9], transpositions [1, 2], and block moves [7, 8, 12, 12]. In

^{*}Center for the Advanced Study of Algorithms, School of Computer Science, University of Nevada, Las Vegas, NV 89154, USA. Email: bein@cs.unlv.edu. This author worked on this project at the University of Texas at Dallas while on sabbatical leave from UNLV.

[†]Center for the Advanced Study of Algorithms, School of Computer Science, University of Nevada, Las Vegas, NV 89154, USA. Email: larmore@cs.unlv.edu.

[‡]Computer Science Department, Texas A&M University-Commerce, Commerce, TX 75429, USA. Email: Linda_Morales@tamuc.edu.

[§]Department of Computer Science, University of Texas at Dallas Richardson, TX 75083, USA. Email: hal@utdallas.edu.

A F C B E D		How ? they did it do		
A C B E F D	move F	How they did	it ?	do
A C B E D	and combine	How they did	-E-	do
A B C E D	move C	How did they	-E-	do
A E D	and combine	—A—	-E-	do
A D E	move D	—A—	do	-E-
A	and combine	—A—		

Table 1: How to block sort “How ? they did it do”

particular, block sorting is not the same as transposition sorting, and thus the 1.5-approximation to optimality obtained by Bafna and Pevzner [1, 2] does not imply a 1.5-approximation to optimum block sorting. Furthermore, optimum block sorting is known to be \mathcal{NP} -hard [3], while the question of \mathcal{NP} -hardness for optimum transposition sorting is currently open. The two problems are related in the sense that every sequence of block sorting moves defines a sequence of transpositions (but not vice-versa). The study of block sorting might give further insight into transposition sorting.

Block sorting is motivated by applications in optical character recognition; see [6, 10, 11]. Text regions, referred to as *zones* are selected by drawing rectangles (or piecewise rectangles, polygons) around them. Here the order of zones is significant, but in practice the output generated by a zoning procedure may be different from the correct text. A situation prone to such misidentification might arise from multi-column documents, for example, as part of “de-columnizing” such a multi-column document. Zones which are not in the correct order must be further processed (sometimes by a human editor). In the OCR community the number of editing operations, such as different kinds of deletions, insertions and moves is used to define a *zoning metric*. We refer the reader to the work in [6, 10, 11] for details, but mention here that the block sorting problem plays a significant part in defining the zoning metric: moving the pieces to the correct order corresponds to a block sorting problem. To evaluate the performance of a given zoning procedure it is of interest to find the minimum number of steps needed to obtain the correct string from the zones generated by the zoning procedure. An example motivating such an application is given in Table 1.

The research history of block sorting is as follows. Initially, Latifi *et al.* [6] performed various experiments to test a number of strategies that seemed to perform well in practice; however, no ratio better than three was proved for any of these schemes. (The approximation ratio of three arises trivially: if the list is not sorted, simply pick an arbitrary block and move it to a position where it can be combined into a larger block. The number of blocks can be decreased by at most three for any block sorting step, while the trivial algorithm decreases that number by at least one at each step, and hence has approximation ratio three.) As a next step, Bein *et al.* [3] showed that the block sorting problem is \mathcal{NP} -hard. In the same paper they showed how to implement the strategies of [6] in linear time, which is significant in practice. However, no approximation ratio (other than the trivial) was given.

The first approximation algorithm with a non-trivial approximation ratio for the block sorting

problem was given in [12, 13] by Mahajan *et al.* They gave an $O(n^3)$ -time block sorting algorithm with approximation ratio 2. Their algorithm first solves a related problem, the *block merging problem*, by constructing a crossing graph in $O(n^3)$ -time and then deriving a block merging sequence. (Even though not explicitly stated in their paper, it appears that the derivation of the actual sequence does not increase the time complexity of the overall procedure beyond $O(n^3)$.) The solution to the block merging problem is then used to get a 2-approximation for block sorting.

In this paper, we improve that result by giving a quadratic time block sorting algorithm with approximation ratio 2. Central to our method is the block deletion problem. This problem is closely related to the block sorting problem. The *block deletion problem* is the problem of finding the minimum length sequence of block deletions to transform a list of distinct integers into a monotone increasing list. The *complete block deletion problem* is the same, except that the final list must be empty. We show in Section 4 that block deletion is not equivalent to block merging.

As we will show in the next section, the block deletion problem can be solved in $O(n^2)$ time. In Section 3 we show, given a permutation x , that (a) if there is a block deletion sequence of length m for x , there is a block sorting sequence of length m for x and that (b) if there is a block sorting sequence of length m for x , then there is a block deletion sequence of length at most $2m - 1$ for x . From this we derive the 2-approximation algorithm.

2 Optimum Block Deletion in Quadratic Time

2.1 Preliminaries

As mentioned earlier we define a *permutation* of length n to be a list $x = (x_1, \dots, x_n)$ consisting of the integers $\{1, \dots, n\}$ where each number occurs exactly once. For any $1 \leq i \leq j \leq n$ we denote the sublist of x that starts at position i and ends with position j by $x_{i\dots j}$. We say that a list y is a *subsequence* of x if y is obtained from x by deleting any number of elements. For example, $(2, 3)$ is a subsequence of $(2, 4, 3, 1)$, but not a sublist. Since x has no duplicate symbols, a subsequence of x is uniquely characterized by its set of items. By a slight abuse of notation, we shall sometimes identify a subsequence with the set of its items. Define the *closure* of a subsequence of x to be the smallest sublist of x which contains it. For example, the closure of the subsequence $(2, 3)$ of $(2, 4, 3, 1)$ is the sublist $(2, 4, 3)$. If A and A' are subsequences of a list x , we say that A and A' are *separated* if the closures of A and A' are disjoint.

A block deletion sequence for a subsequence y of x consists of a sequence A_1, \dots, A_m of disjoint non-empty subsequences of y such that

1. for all $i = 2, \dots, m$, A_i is a block in $y - \bigcup_{u=1}^{i-1} A_u$, and
2. $y - \bigcup_{u=1}^m A_u$ is a monotone increasing list.

For example, a minimum length block deletion sequence for the list $(1, 4, 2, 5, 3)$ consists of two steps. First delete the block (2) , obtaining $(1, 4, 5, 3)$, then delete the block $(4, 5)$, obtaining the sorted list $(1, 3)$. The left part of Figure 3 shows another example of a block deletion sequence. A complete block deletion sequence for a subsequence y of x consists of a block deletion sequence A_1, \dots, A_m of y such that $y - \bigcup_{u=1}^m A_u$ is the empty list.

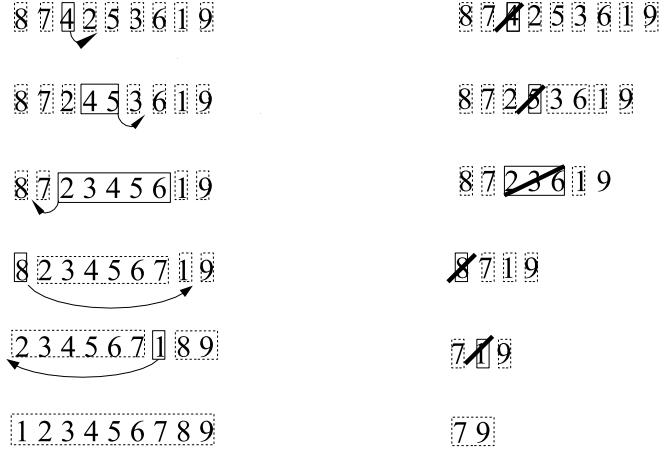


Figure 1: Block Sorting Moves (left) and corresponding Relative Block Deletions (right)

Block sorting has been shown to be equivalent to “relative block deletion”. Given a list x of distinct integers and a sublist $y = (r, \dots, s)$ of x , we say y is a *relative block* (*rel-block* for short) of x if the following conditions hold:

- y is monotone increasing.
- If u is an integer with $r < u < s$, then $u \in x$ implies $u \in y$. That is, any element of the integer interval $\{r, r + 1, \dots, s\}$ not present in y is not present in x either.

For example, in the list $8, 7, 2, 3, 6, 1, 9$ the sublist $2, 3, 6$ is a relative block.

We define a *relative block deletion sequence* for x to be a sequence of rel-block deletions, which transforms x into a monotone sequence. From [3] we have the following result.

Theorem 1 *Let x be a permutation. Then the minimum number of block sorting moves needed to sort x is the same as the minimum length of a rel-block deletion sequence for x .*

Figure 1 shows a sequence of block moves with a corresponding sequence of relative block deletions. Note that the relative block deletion sequence corresponding to Figure 1 is $(4), (5), (2, 3, 6), (8), (1)$.

2.2 A Dynamic Program for Complete Block Deletion

We first consider the complete block deletion problem for all sublists of x , which we solve in quadratic time by dynamic programming. Once the $O(n^2)$ answers to this problem are obtained, the original block deletion problem can be solved in quadratic time. We make use of the following three lemmas:

Lemma 1 *If A_1, \dots, A_m is a block deletion sequence for a sublist y of x , and $1 \leq u < v \leq m$, then either A_u and A_v are separated, or A_u is a subsequence of the closure of A_v .*

Proof: The closure of A_u cannot contain any item of A_v , since otherwise A_u could not be deleted before A_v . If all items of A_v are before A_u or all items of A_v are after A_u , then A_u and A_v are separated. If some items of A_v are before A_u and some items are after A_u , then A_u is a subsequence of the closure of A_v . \square

Lemma 2 *If $A_1, \dots, A_t, A_{t+1}, \dots, A_m$ is a block deletion sequence for a sublist y of x , and A_t and A_{t+1} are separated, then A_t and A_{t+1} may be transposed, i.e., $A_1, \dots, A_{t+1}, A_t, \dots, A_m$ is a block deletion sequence for y .*

Proof: For any u , let $y_u = x - \bigcup_{v < u} A_v$. By definition, A_t is a block of y_t , and A_{t+1} is a block of $y_{t+1} = y_t - A_t$. Since A_t and A_{t+1} are separated, A_{t+1} is also a block of y_t . Thus, A_{t+1} can be deleted before A_t . \square

Lemma 3 *For any $1 \leq i \leq j \leq n$, if there is a complete block deletion sequence for $x_{i\dots j}$ of length m , then there is a complete block deletion sequence for $x_{i\dots j}$ of length m such that x_i is deleted in the last step.*

Proof: Let A_1, \dots, A_m be a complete block deletion sequence of $x_{i\dots j}$. Suppose that $x_i \in A_t$ for some $t < m$. Since x_i is deleted in the t^{th} move of the sequence, all deletions after that must involve blocks whose first symbol occurs to the right of x_i in x . That is, for any $v > t$, x_i cannot be an item of the closure of A_v , hence, by Lemma 1, A_t and A_v must be separated. By Lemma 2, we can transpose A_t with A_v for each $v > t$ in turn, moving A_t to the end of the deletion sequence. \square

We now give a recurrence to compute the the minimum length of a complete block deletion sequence. This recurrence will be used in Algorithm 1.

Theorem 2 *Given a permutation x , let $t_{i,j}$ denote the minimum length of any complete block deletion sequence for the sublist $x_{i\dots j}$. The values of $t_{i,j}$ can be computed inductively by the following: Set $t_{i,i} = 1$, $t_{i,j} = 0$ for $i > j$, and for $i < j$ set*

$$t_{i,j} = \begin{cases} \min\{1 + t_{i+1,j}, t_{i+1,\ell-1} + t_{\ell,j}\}, & \text{if } \exists \ell \in \{i+1, \dots, j\} \text{ such that } x_\ell = x_i + 1, \\ 1 + t_{i+1,j}, & \text{otherwise.} \end{cases}$$

Proof: We proceed by induction on $j - i$ to show that $t_{i,j}$ is computed correctly. The base case is trivial, namely $t_{i,i} = 1$, because it takes one block deletion to delete a sublist of length 1.

For the inductive step, assume that $t_{i,j}$ is the length of a minimum block deletion sequence for $x_{i\dots j}$ when $j - i < k$. For $j - i = k$, let $m = t_{i,j}$ and let A_1, \dots, A_m be a corresponding minimum length complete block deletion sequence of $x_{i\dots j}$. By Lemma 3, we can insist that $x_i = a$ is an item in A_m . We consider two cases based on whether there is an ℓ with $i < \ell \leq j$ such that $x_\ell = a + 1$. (The reader might also consult Figure 2.)

If the element $a + 1$ does not occur in the interval $\{i + 1, \dots, j\}$, then the element a is not part of a block in this interval and must be deleted by itself. So, A_1, \dots, A_{m-1} is a complete block deletion sequence of $x_{i+1\dots j}$. Note that it must be of optimum length, for if B_1, \dots, B_r

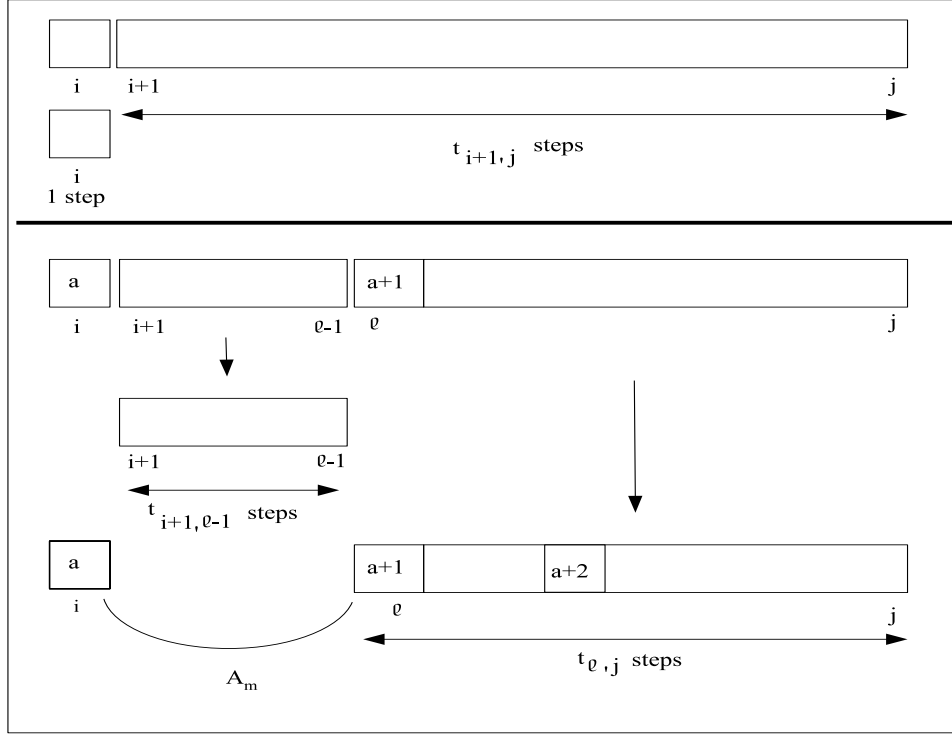


Figure 2: The Recurrence for the $t_{i,j}$ Values

were a shorter complete block deletion sequence of $x_{i+1\dots j}$, then $B_1, \dots, B_r, \{a\}$ would be shorter than A_1, \dots, A_m . Thus, as $j - (i + 1) = k - 1$, by the induction hypothesis $t_{i+1,j} = m - 1$. So $t_{i,j} = 1 + t_{i+1,j}$, which is optimum.

Now for the case that the element $x_\ell = a + 1$ does occur in the interval $\{i + 1, \dots, j\}$. This means that $a + 1$ and possibly other larger values can be included in A_m when element a is deleted. If element $a + 1$ is not included in A_m then the same argument used in the previous paragraph shows that $m = 1 + t_{i+1,j}$. If on the other hand $a + 1$ is included in A_m , then by Lemma 1, for any $t, (1 \leq t \leq m)$, A_t is either completely before or completely after the element $a + 1$, since A_m is deleted after A_t . By Lemma 2, one can permute the indices so that, for some $u < m$

1. if $t \leq u$, then A_t is a subsequence of $x_{i+1\dots \ell-1}$, and
2. if $u < t \leq m$, then A_t is a subsequence of $x_{\ell+1\dots j}$.

Consequently, A_1, \dots, A_u is a block deletion sequence for $x_{i+1\dots \ell-1}$ and $A_{u+1}, \dots, A_m - \{a\}$ is a block deletion sequence for $x_{\ell+1\dots j}$. Both of these block deletion sequences must be optimum for their respective intervals. That is, for example, if B_1, \dots, B_r were a shorter complete block deletion sequence for $x_{i+1\dots \ell-1}$, then $B_1, \dots, B_r, A_{u+1}, \dots, A_m$ would be a complete block deletion

sequence for $x_{i\dots j}$, contradicting the minimality of m . By the induction hypothesis, as $\ell - 1 - (i + 1) < j - i$ and $j - \ell < j - i$, it follows that $t_{i+1,\ell-1} = u$ and $t_{\ell,j} = m - u$, so $t_{i,j} = t_{i+1,\ell-1} + t_{\ell,j} = u + (m - u) = m$. \square

The resulting dynamic programming algorithm BLOCKDELETION, which is derived from the recurrence of Theorem 2, is shown below.

Algorithm 1 BLOCKDELETION(X)

Let n be the number of elements in x

for $i \leftarrow 1$ **to** n **do**

$t[i, i] \leftarrow 1$

for $k \leftarrow 2$ **to** n **do**

for $i \leftarrow 1$ **to** $n - k + 1$ **do**

Let $x_\ell = x_i + 1$; $j \leftarrow i + k - 1$

if $i < \ell \leq j$

then $t[i, j] \leftarrow \min\{(1 + t[i + 1, j]), (t[i + 1, \ell - 1] + t[\ell, j])\}$

else $t[i, j] \leftarrow 1 + t[i + 1, j]$

return

We now turn to the analysis of the run time of algorithm BLOCKDELETION. Let $z = (z_1, \dots, z_n)$ be the inverse permutation of x , i.e., $x_i = k$ if and only if $z_k = i$. Note that z can be computed in $O(n)$ preprocessing time.

Theorem 3 *Algorithm BLOCKDELETION has run time $O(n^2)$.*

Proof: To prove the theorem we show that if $t_{u,v}$ are already known for all $i < u \leq v \leq j$, then $t_{i,j}$ can be computed in $O(1)$ time. Let $m = t_{i,j}$ for $i < j$ and let A_t be the subsequence of $x_{i\dots j}$ that is deleted at step t of the complete block deletion sequence of $x_{i\dots j}$ of length m . By Lemma 3, we can assume that $x_i \in A_m$. If $|A_m| > 1$, the index $\ell = z_{x_i+1}$, i.e. $x_\ell = 1 + x_i$, can be found in $O(1)$ time, since we have already spent $O(n)$ preprocessing time to compute the array z . The recurrence thus takes $O(1)$ time to execute for each i, j . \square

We finally note that to obtain the actual sequence of steps in the optimum complete block deletion sequence one can store appropriate pointers as the $t_{i,j}$ are computed.

2.3 Computing Block Deletion

We remind the reader that for the block deletion problem one needs to find the minimum length sequence of block deletions to transform a permutation into a monotone increasing list. We now show how to obtain a solution to the block deletion problem for x in $O(n^2)$ -time given that all $t_{i,j}$ are known for $1 \leq i \leq j \leq n$. Define a weighted acyclic directed graph G with one node for each $i \in \{0, \dots, n + 1\}$. There is an edge from i to j if and only if $i < j$ and $x_i < x_j$, and the weight of that edge is $t_{i+1,j-1}$. Simply observe that there is a path $\langle 0, i_1, i_2, \dots, i_k, n + 1 \rangle$ in G exactly when x_{i_1}, \dots, x_{i_k} is a monotone increasing list. Furthermore the weight of the edge $\langle i_\ell, i_{\ell+1} \rangle$ gives the minimum number of block deletions necessary to delete elements between

position i_ℓ and $i_{\ell+1}$ in x . Thus there is a block deletion sequence of x of length m , there must be a path from 0 to $n + 1$ in G of weight q , and vice-versa.

Using standard dynamic programming, a minimum weight path from 0 to $n + 1$ can be found in $O(n^2)$ time. Let $0 = i_0 < i_1 < \dots < i_\ell = n + 1$ be such a minimum weight path, and let $w = \sum_{u=1}^{\ell} t_{i_{u-1}+1, i_u-1}$ be the weight of that minimum path. Since every deletion is a block deletion, the entire list can be deleted to a monotone list in w block deletions. Thus we have:

Theorem 4 *The block deletion problem can be solved in time $O(n^2)$.*

3 Block Deletion and Block Sorting

We now explain the relation between Block Deletion and Block Sorting. To this end, we introduce the following notation: Given a permutation x we denote by $bs(x)$ the minimum number of block moves to sort x , and by $bd(x)$ the length of a shortest block deletion sequence for x .

In this section we give an algorithm, BLOCK2, which constructs a block sorting for any given permutation x . BLOCK2 uses the dynamic programming procedure described in the previous section to calculate a block deletion sequence and derives a block sorting from that sequence. We show that algorithm BLOCK2 is a 2-approximation algorithm for the block sorting problem, *i.e.* the number of moves which BLOCK2 produces for any permutation x is not more than twice $bs(x)$.

In the description of the algorithm it is convenient to use the following notation: We write $first(A)$ to denote the first element of a block A . For a permutation x , let \hat{x} be the new list obtained from list x by placing the element 0 at the beginning and the element $n + 1$ at the end, *i.e.*, $\hat{x}_0 = 0$ and $\hat{x}_{n+1} = n + 1$, and $\hat{x}_i = x_i$ for $1 \leq i \leq n$. Algorithm 2 produces a sequence of permutations x^0, x^1, \dots, x^m from a block deletion sequence A_1, \dots, A_m .

Algorithm 2 BLOCK2

1. Block Deletion:

Compute a minimum length block deletion sequence A_1, \dots, A_m of x .

2. Obtaining Block Sorting

Initialization Let $x^0 = \hat{x}$.

Loop For each t from 1 to m , do the following:

1. Let B_t be the block of x^{t-1} which contains $first(A_t)$.
 2. If $first(A_t) \neq first(B_t)$, let $x^t = x^{t-1}$.
Otherwise, let x^t be obtained by deleting B_t from x^{t-1} and reinserting it just after $first(B_t) - 1$.
-

Figure 3 gives an example illustrating how block sorting moves are obtained from a block deletion sequence. Let $M = \{0, 1, \dots, n, n + 1\} - \bigcup_{t=1}^m A_t$ be the monotone increasing subsequence consisting of the items of x^0 that remain after the block deletions. Elements in M are underlined in the figure.

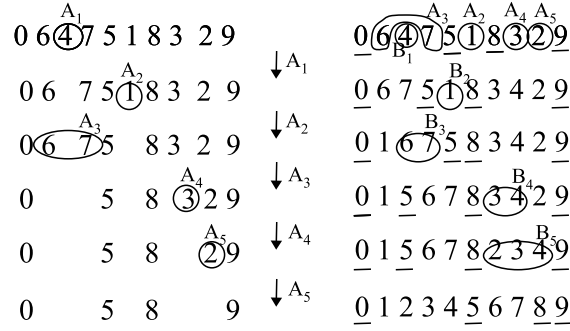


Figure 3: Obtaining Block Sorting Moves from Block Deletions

The correctness of the algorithm follows from the lemma below.

Lemma 4 *For any permutation x algorithm BLOCK2 produces a block sorting for x with no more than $bd(x)$ block moves.*

Proof: We show that the list x^m produced by Algorithm 2 is sorted. For a contradiction, assume that x^m is not sorted. Define a *jump* of x^t to be an item $x_i^t = a$ such that $x_{i-1}^t \neq a - 1$. We say that a jump $x_i^t = a$ is an *inversion* of x^t if $x_{i-1}^t > a$. Note that if a is a jump of x^t , it is a jump of x^u for all $u < t$, since no iteration of the loop of BLOCK2 creates a new jump.

We first claim that, if a is any jump of x^m , then $a \in M$. If $a \in A_t$, then a is not a jump of x^t , hence not a jump of x^m . This proves the claim.

Suppose x^m is not sorted. That is, x^m has an inversion $x_i^m = a$. By the claim, $a \in M$. Let b be the smallest item in the maximal block B of x^m that contains x_{i-1}^m . We know that $b > a$, since $x_{i-1}^m > a$, B does not contain a , and B contains all integers between b and x_{i-1}^m . By the claim, $b \in M$. But b is before a in x^m , and hence also in M , since the items of M are never moved. This is a contradiction since M is the monotone increasing sequence remaining at the end of a block deletion sequence. So, x^m is sorted.

Finally, observe that $bd(x) = m$, since A_1, \dots, A_m is a minimum length block deletion sequence for x .

□

To show that algorithm BLOCK2 gives an approximation for the block sorting problem we show that the number of block deletions can be bounded in terms of the number of block sorting moves.

Lemma 5 *Given permutation x , then $bd(x) \leq 2bs(x) - 1$.*

Proof: We show that if there is a block sorting for a permutation x which has m steps, then there is a block deletion sequence for x of length at most $2m - 1$. Suppose we are given a block sorting sequence B_1, B_2, \dots, B_m of x of length m , where B_t is the block moved at the t^{th} step. Let M be the monotone increasing subsequence of x consisting of all items of x which are never

moved. In the construction below, we think of each B_t as a set of integers. Let \mathcal{B} be the forest whose nodes are B_1, \dots, B_m , where B_t is a child of B_u if and only if $B_t \subseteq B_u$. We now place one credit on each root of \mathcal{B} and two credits on each B_t which is not a root of \mathcal{B} . Thus, we place at most $2m - 1$ credits. Each B_t which is not a root will pass one credit up to its parent. The block deletion sequence is obtained from B_1, B_2, \dots, B_m by deleting for each i ($1 \leq i \leq m$), the fragments of block B_i remaining in x . Note that, since we are deleting blocks rather than moving them, the movement of a block B_i may be translated into the deletion of several fragmentary blocks, where the missing portions are the parts of B_i that have already been deleted.

We claim that each B_t has enough credits to pay for its deletion in a block deletion sequence for x , where each block deletion takes one credit. Suppose we have deleted B_1, \dots, B_{t-1} . Then B_t may have been partially deleted. The remaining items form a rel-block of the remaining list, but not necessarily a block. However, the number of ‘‘holes’’ in B_t cannot exceed the number of children of B_t in the forest \mathcal{B} . That is, if B_t has r children, the undeleted items of B_t form the disjoint union of at most $r + 1$ intervals, and are thus the items of at most $r + 1$ blocks in x . To delete these blocks, we use $r + 1$ credits on B_t . However, B_t received r credits from its r children and was initially given 2 credits. Hence, B_t has a total of $r + 2$ credits. So, B_t uses $r + 1$ credits and passes one up to its parent. After all block deletions, the remaining list is M , which is sorted. \square

Theorem 5 *Algorithm BLOCK2 has an approximation ratio of 2 and has time complexity $O(n^2)$.*

Proof: The approximation ratio is an immediate corollary of Lemma 4 and Lemma 5: Given any permutation x and let $b_{\text{Block2}}(x)$ be the number of block moves produced by BLOCK2 to sort x . Then we have

$$bs(x) \leq b_{\text{Block2}}(x) \leq bd(x) \leq 2bs(x) - 1.$$

Regarding the time complexity, it takes $O(n^2)$ time to find a minimal block deletion sequence. The remaining parts of the algorithm, such as additional steps to keep track of intermediate results, take $O(n^2)$ time. \square

4 Final Remarks and Open Problems

The block merging problem of Mahajan *et al.* [12] is defined as follows: Given a permutation, at each step a configuration of the problem consists of a set of sorted lists of integers, where each integer is in exactly one list. One or more of the lists may be empty. The initial configuration is the set of lists given by the maximal sorted sublists of the permutation. A move consists of deleting a block from one of the lists and inserting that same block into one of the other lists in such a way that the moved block merges with another block. The object is to find the minimum number of moves to reach the configuration where there is only one non-empty list and it is sorted. It is entirely possible that block merging and block deletion are related, but they are not identical: For example the permutation $x = (2, 5, 7, 3, 6, 1, 4)$, has the following block merging sequence:

$$\begin{aligned} &\{(2, 5, 7), (3, 6), (1, 4)\} \\ &\{(2, 5, 6, 7), (3), (1, 4)\} \end{aligned}$$

$$\{(5, 6, 7), (2, 3), (1, 4)\}$$
$$\{(5, 6, 7), \epsilon, (1, 2, 3, 4)\}$$
$$\{(1, 2, 3, 4, 5, 6, 7), \epsilon, \epsilon\}$$

It is easily verified that a solution such as the one above with four steps is optimum for block merging. However, with block deletions one can obtain a monotone sequence in only 3 steps: from $(2, 5, 7, 3, 6, 1, 4)$ delete (6) , delete (1) , delete $(3, 4)$ to obtain $(2, 5, 7)$.

We have given a better non-trivial approximation algorithm with a provable approximation ratio for the block sorting problem. There may be, however, room for further improvement. We mention two lines of further investigation:

1. We conjecture that a polynomial time approximation algorithm with a ratio better than 2 exists.
2. It would be interesting to see how our algorithm compares with some of the heuristics given in [6]. All of those heuristics lack proof of any approximation ratio; their advantage is that they have linear run time. Indeed, it would be desirable to give a 2-approximation with run time better than $O(n^2)$.

Finally we mention that the study of block merging, block sorting and block deletions might lead to insights for other sorting problems, such as sorting by transpositions.

References

- [1] V. Bafna and P.A. Pevzner. Sorting by transposition. *SIAM Journal on Discrete Mathematics*, 11:224–240, 1998.
- [2] V. Bafna and P.A. Pevzner. Genome rearrangements and sorting by reversals. *SIAM Journal on Computing*, 25:272–289, 1999.
- [3] W.W. Bein, L.L Larmore, S. Latifi, and I.H Sudborough. Block sorting is hard. *International Journal of Foundations of Computer Science*, 14(3):425–437, 2003.
- [4] A. Caprara. Sorting by reversals is difficult. In *Proceedings 1st Conference on Computational Molecular Biology*, pages 75–83. ACM, 1997.
- [5] W. H. Gates and C. H. Papadimitriou. Bounds for sorting by prefix reversal. *Discrete Mathematics*, 27:47–57, 1979.
- [6] R. Gobi, S. Latifi, and W. W. Bein. 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*, pages 253–259. CSREA Press, 2000.
- [7] L.S. Heath and J.P.C. Vergara. Sorting by bounded block-moves. *Discrete Applied Mathematics*, 88:181–206, 1998.

- [8] L.S. Heath and J.P.C. Vergara. Sorting by short block-moves. *Algorithmics*, 28 (3):323–354, 2000.
- [9] H. Heydari and I. H. Sudborough. On the diameter of the pancake network. *Journal of Algorithms*, 25(1):67–94, 1997.
- [10] J. Kanai, S. V. Rice, and T.A. Nartker. Automatic evaluation of OCR zoning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17 (1):86–90, 1995.
- [11] S. Latifi. How can permutations be used in the evaluation of automatic zoning evaluation? In *ICEE 1993*. Amir Kabir University, 1993.
- [12] M. Mahajan, R. Rama, V. Raman, and S. Vijayakumar. Approximate block sorting. *International Journal of Foundations of Computer Science*, 17 (2): 337– 355, 2006.
- [13] M. Mahajan, R. Rama, and S. Vijayakumar. Merging and sorting by strip moves. In *Proceedings of the 23rd International Foundations and Software Technology and Theoretical Computer Science Conference (FSTTCS)*, volume 2914 of *Lecture Notes in Computer Science*, pages 314–325. Springer, 2003.