

A Faster and Simpler 2-Approximation Algorithm for Block Sorting

Wolfgang W. Bein¹, Lawrence L. Larmore¹,
Linda Morales², and I. Hal Sudborough³

¹ School of Computer Science, University of Nevada,
Las Vegas, Nevada 89154, USA**

`bein@cs.unlv.edu` `larmore@cs.unlv.edu`

² Computer Science Department, Texas A&M University-Commerce,
Commerce, TX 75429, USA

`Linda.Morales@tamu-commerce.edu`

³ Department of Computer Science, University of Texas at Dallas,
Richardson, TX 75083, USA

`hal@utdallas.edu`

Abstract. Block sorting is used in connection with optical character recognition (OCR). Recent work has focused on finding good strategies which perform well in practice. Block sorting is \mathcal{NP} -hard and all of the previously known heuristics lack proof of any approximation ratio. We present here an approximation algorithm for the block sorting problem with approximation ratio of 2 and run time $O(n^2)$. The approximation algorithm is based on finding an optimal sequence of absolute block deletions.

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 $x_{i..j}$. If $j < i$, let $x_{i..j}$ be the empty list. We call a nonempty sublist of x consisting of consecutive 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 minimal length sequence of *block move steps*, or simply *block moves* for short, which sorts an initial permutation x , where a block move consists of moving one block of a list 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

** Research of these authors supported by NSF grant CCR-0312093.

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

| | | | |
|-----------------------|----------------------|------|-----|
| 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———— | | |

block moves [7,8,12,12]. In 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 optimal block sorting. Furthermore, optimal block sorting is known to be \mathcal{NP} -hard [3], while the question of \mathcal{NP} -hardness for optimal 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). Thus, the study of block sorting might give further insights 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 the block sorting is as follows: initially, Latifi *et al.* [6] have performed various experiments to test a number of strategies that seem to perform well in practice; however, no ratio better than three has been proved for any of these schemes. (The approximation ratio of three arises trivially: if the list is not sorted, simply pick an arbitrary maximal block and move it to a position where it can be combined into a larger block. The number of maximal 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, [3] Bein *et al.* have shown that the block sorting problem is \mathcal{NP} -hard [3]. In the same paper they show how to

implement the strategies of [6] in linear time, which is significant in practice. However, no approximation ratio (other than the trivial) is given.

The first approximation algorithm with a non-trivial approximation ratio for the block sorting problem is given in [12,13] by Mahajan *et al.* They give 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 $O(n^3)$ 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 a problem, closely related to the block sorting problem, but which is in the polynomial class, the *abs-block deletion problem*. We note that the abs-block deletion problem is not equivalent to the block merging problem; see our final remarks in Section 4. We call the *deletion* of a block an *absolute block deletion*, or *abs-block deletion* for short. The *abs-block deletion problem* is the problem of finding the minimum length sequence of abs-block deletions to transform a list of distinct integers into a monotone increasing list. The *complete abs-block deletion problem* is the same, except that the final list must be empty.

As we will show in the next section, the abs-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 an abs-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 an abs-block deletion sequence of length at most $2m - 1$ for x . From this we derive the 2-approximation algorithm.

2 Finding Optimal Absolute Block Deletion Sequences in Quadratic Time

2.1 Preliminaries

We note the difference between absolute block deletions and relative block deletions. We make both concepts precise here.

Given a list x of distinct integers and a sublist y 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 $r < u < s$ are integers such that r and s are in y , then either u is in y or u is not in x .

Given x , we define a *relative block deletion* (*rel-block deletion*, for short) to be the deletion of a relative block from x .

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

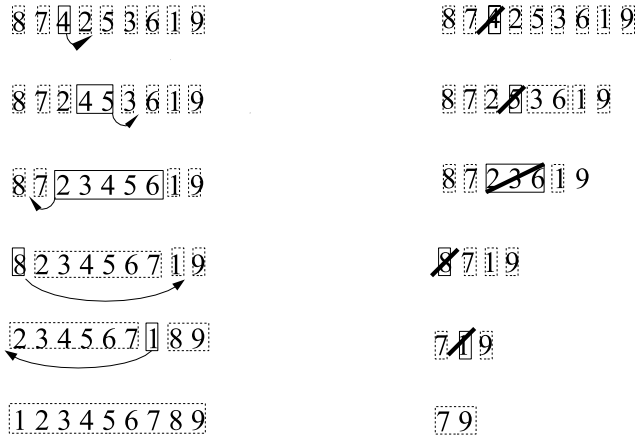


Fig. 1. Block Sorting Moves (left) and corresponding Relative Block Deletions (right)

Theorem 2.1. *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.)

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. 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.

An abs-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 $y - \bigcup_{u=1}^m A_u$ is monotone, and each A_v is a block of the subsequence $y - \bigcup_{u < v} A_u$. For example, the minimum length abs-block deletion sequence for the list $(1, 4, 2, 5, 3)$ consists of two steps. First delete the abs-block (2) , obtaining $(1, 4, 5, 3)$, then delete the abs-block $(4, 5)$, obtaining the sorted list $(1, 3)$. The left part of Figure 3 shows another example of an abs-block deletion sequence.

A complete abs-block deletion sequence for a subsequence y of x consists of an abs-block deletion sequence A_1, \dots, A_m of y such that $y - \bigcup_{u=1}^m A_u$ is the empty list.

2.2 A Dynamic Program for Absolute Block Deletion

We first consider the complete abs-block deletion problem for all sublists of x , which we solve in quadratic time. Once the $O(n^2)$ answers to this problem are

obtained, the original abs-block deletion problem can be solved in quadratic time.

For all $1 \leq i, j \leq n$, let $t_{i,j}$ to be the minimum length of any complete abs-block deletion sequence for the sublist $x_{i\dots j}$. Our algorithm first computes all $t_{i,j}$ by dynamic programming. Trivially, $t_{i,i} = 1$, and $t_{i,j} = 0$ for $j < i$.

Lemma 2.2. *If A_1, \dots, A_m is an abs-block deletion sequence for a sublist of y , and $1 \leq u < v \leq m$, then either A_u and A_v are separated, or A_u is a subset 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 . Thus, every item of A_v is entirely before A_u or entirely after A_u in x . 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 subset of the closure of A_v .

Lemma 2.3. *If $A_1, \dots, A_t, A_{t+1}, \dots, A_m$ is any abs-block deletion sequence for a sublist y of x , and if 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 an abs-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 .

Lemma 2.4. *For any $1 \leq i \leq j \leq n$, if there is a complete abs-block deletion sequence for $x_{i\dots j}$ of length m , then there is a complete abs-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 abs-block deletion sequence of $x_{i\dots j}$. Suppose that $x_i \in A_t$ for some $t < m$. For any $v > t$, x_i cannot be an item of the closure of A_v , hence, by Lemma 2.2, A_t and A_v must be separated. By Lemma 2.3, we can transpose A_t with A_v for each $v > t$ in turn, moving A_t to the end of the deletion sequence.

Let z 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 2.5. *Let $1 \leq i \leq j \leq n$. Then*

$$t_{i,j} = \begin{cases} \min \left\{ \begin{array}{l} 1 + t_{i+1,j} \\ t_{i+1,\ell-1} + t_{\ell,j} \end{array} \right\} & \text{if } i < n \text{ and } i < \ell \leq j, \text{ where } \ell = z_{x_{i+1}} \\ 1 + t_{i+1,j} & \text{otherwise} \end{cases} \quad (1)$$

Proof. If $i = j$, the recurrence is trivial, so assume $i < j$. Let $a = x_i$. We first prove that the left side of (1) is less than or equal to the right side. If A_1, \dots, A_u is a complete abs-block deletion of $x_{i+1\dots j}$, then $A_1, \dots, A_u, (i)$ is a complete abs-block deletion of $x_{i\dots j}$, hence $t_{i,j} \leq 1 + t_{i+1,j}$.

If $x_\ell = a + 1$ for some $i < \ell \leq j$, A_1, \dots, A_u is a complete abs-block deletion of $x_{i,\ell-1}$, and A_{u+1}, \dots, A_m is a complete abs-block deletion of $x_{\ell,j}$, then

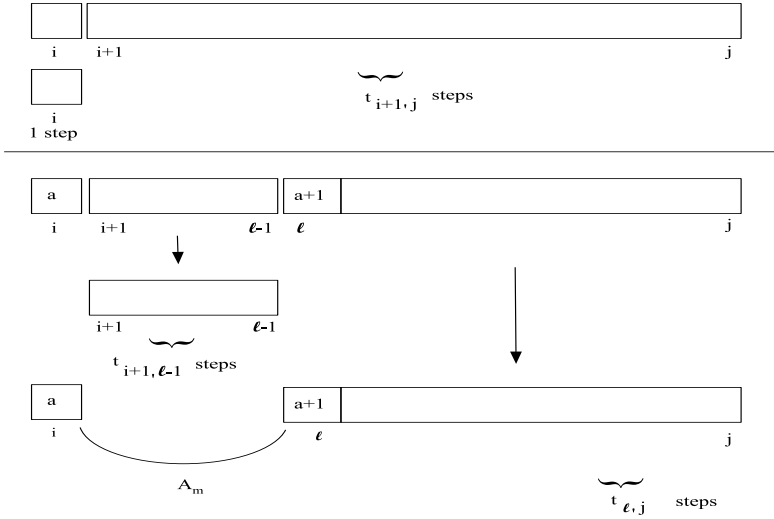


Fig. 2. The Recurrence for the $t_{i,j}$ Values

$A_1, \dots, A_u, A_{u+1}, \dots, A_{m-1}, A_m + \{a\}$ is a complete abs-block deletion of $x_{i,j}$. Thus, the left side of (1) is less than or equal to the right side.

We now show that the left side of (1) is greater than or equal to the right side. Let $m = t_{i,j}$, and let A_1, \dots, A_m be a minimum length complete abs-block deletion sequence of $x_{i\dots j}$. By Lemma 2.4, we can insist that $x_i = a$ is an item of A_m . If $A_m = (a)$, then A_1, \dots, A_{m-1} is a complete abs-block deletion sequence of $x_{i+1\dots j}$, which must be optimal by the optimality principle. Thus, $t_{i+1,j} = m - 1$. Otherwise, A_m contains $a + 1 = x_\ell$, where $i < \ell \leq j$. If $1 \leq t < m$, then A_t is either completely before or completely after $a + 1$, since $a + 1$ is deleted after A_t . By Lemma 2.3, we can permute the indices so that, for some $u < m$, $A_t \subseteq x_{i+1\dots \ell-1}$ if $t \leq u$ and $A_t \subseteq x_{\ell\dots j}$ for $u < t < m$. By the optimality principle, A_1, \dots, A_u is a minimum length complete abs-block deletion sequence for $x_{i+1\dots \ell-1}$, while $A_{u+1}, \dots, A_{m-1}, A_m - \{a\}$ is a minimum length complete abs-block deletion sequence for $x_{\ell\dots m}$. Thus, $u = t_{i+1,\ell}$ and $m - u = t_{\ell,m}$. In any case, the left side of (1) is greater than or equal to the right side.

We now turn to the analysis of the run time of the corresponding dynamic program. We note the following crucial fact:

Lemma 2.6. *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.*

Proof. If $i \geq j$, the result is trivial. Henceforth, assume that $i < j$. Let $m = t_{i,j}$, and let A_t be the subsequence of $x_{i\dots j}$ that is deleted at step t of the complete abs-block deletion sequence of $x_{i\dots j}$ of length m . By Lemma 2.4, we can assume that $x_i \in A_m$. If $|A_m| > 1$, $\ell = z_{x_i+1}$ 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 .

Corollary 2.7. *The values of $t_{i,j}$ for all i, j can be computed in $O(n^2)$ time.*

We finally note that the optimal complete absolute block deletion sequence can be recovered by keeping appropriate pointers as the $t_{i,j}$ are computed.

We now show how to obtain a solution to the abs-block deletion problem for x in $O(n^2)$ -time. Define a weighted acyclic directed graph G with one node for each $i \in \{0, \dots, n + 1\}$. There is an edge of G from i to j if and only if $i < j$ and $x_i < x_j$, and the weight of that edge is $t_{i+1,j-1}$. If there is an abs-block deletion sequence of x of length m , there must be a path from 0 to $n + 1$ in G of weight m , 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 $m = \sum_{u=1}^\ell t_{i_{u-1}+1, i_u-1}$ 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 m block deletions. Thus we have:

Theorem 2.8. *The abs-block deletion problem can be solved in time $O(n^2)$.*

3 Absolute Block Deletion and Block Sorting

We now derive an approximation algorithm “BLOCK2” for the block sorting problem. We say that an algorithm \mathcal{A} has *approximation ratio* C if, for any permutation x of $\{1, \dots, n\}$, \mathcal{A} finds a block sorting of x of length at most $C \cdot p + O(1)$, where p is the minimum number of block sorting moves needed to sort x . Given a permutation x , BLOCK2 performs the following steps:

1. Compute a minimum length abs-block deletion sequence of x . Let A_1, \dots, A_m be the blocks that are deleted in this sequence. For each $1 \leq t \leq m$, let a_t be the first item of A_t .
2. Let x^0 be the augmentation of x , i.e., $x^0_0 = 0$ and $x^0_{n+1} = n + 1$, and $x^0_i = x_i$ for $1 \leq i \leq n$.
3. Let $M = \{0, 1, \dots, n, n + 1\} - \bigcup_{t=1}^m A_t$, the monotone increasing subsequence consisting of the items of x^0 that remain after those deletions. Note that $0, n + 1 \in M$.
4. **Loop:** For each t from 1 to m , do the following:
 - (a) Let B_t be the maximal block of x^{t-1} which contains a_t .
 - (b) If a_t is not the first item of 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 $a_t - 1$.

Figure 3 gives an example which illustrates how block sorting moves are obtained from an abs-block deletion sequence. Elements in M are underlined in the figure.

Clearly, BLOCK2 uses at most m block moves. Our result then follows from the lemma below.

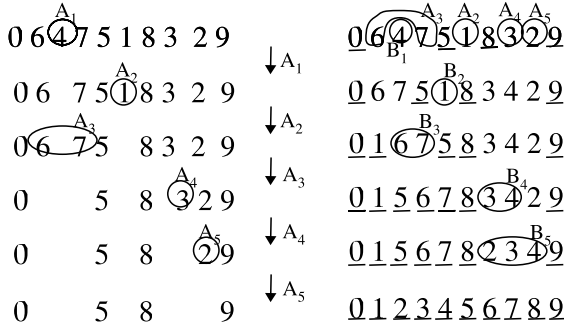


Fig. 3. Obtaining Block Sorting Moves from Abs-Block Deletions

Lemma 3.1. *The list x^m is sorted.*

Proof. 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.

If x^m is not sorted, then x^m must have 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 in M , since the items of M are never moved, and M is increasing: contradiction. Thus x^m is sorted.

We now show that BLOCK2 gives a 2-approximation for the block sorting problem.

Lemma 3.2. *If there is a block sorting for a permutation x which has m steps, then there is an abs-block deletion sequence for x of length at most $2m - 1$.*

Proof. Suppose we are given a block sorting sequence of x of length m . Let B_t be the block that is 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 in the subtree rooted at 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 then passes one credit up to its parent.

We claim that each B_t has enough credits to pay for its deletion in an abs-block deletion sequence for x , where each abs-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 an abs-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 the $r + 1$ credits on B_t . After all block deletions, the remaining list is M , which is sorted.

As an immediate corollary of Lemma 3.2, we have

Theorem 3.3. *Algorithm BLOCK2 has an approximation ratio of 2.*

Regarding the time complexity of BLOCK2 we have:

Theorem 3.4. *The time complexity of BLOCK2 is $O(n^2)$.*

Proof. It takes $O(n^2)$ time to find a minimal abs-block deletion sequence. The remaining parts of the algorithm, such as additional steps to keep track of intermediate results, take $O(n^2)$ time.

4 Final Remarks and Open Problems

The block merging problem of Mahajan *et al.* [12] is defined as follows: At each step, a configuration of the problem consists of a set of lists of integers, where the set of all integers in the lists is $\{1, \dots, n\}$, and no integer appears in two lists, or more than once in one list. One or more of the lists may be empty. 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. We are given an initial configuration, and we want 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 abs-block deletion are related, but they are not identical: the set of lists $\{(3, 7, 9), (4, 8), (1, 5), (2, 6)\}$ takes 8 steps to sort by block merging, as follows:

$$\begin{aligned} &\{(3, 7, 9), (4, 8), (5), (1, 2, 6)\} \\ &\{(1, 2, 3, 7, 9), (4, 8), (5), (6)\} \\ &\{(7, 9), (1, 2, 3, 4, 8), (5), (6)\} \\ &\{(7, 9), (8), (1, 2, 3, 4, 5), (6)\} \\ &\{(7, 9), (8), \epsilon, (1, 2, 3, 4, 5, 6)\} \\ &\{(1, 2, 3, 4, 5, 6, 7, 9), (8), \epsilon, \epsilon\} \\ &\{(9), (1, 2, 3, 4, 5, 6, 7, 8), \epsilon, \epsilon\} \\ &\{(1, 2, 3, 4, 5, 6, 7, 8, 9), \epsilon, \epsilon, \epsilon\} \end{aligned}$$

However, abs-block deletion takes 4 steps to get to a monotone sequence from, $(3, 7, 9, 4, 8, 5, 1, 2, 6)$: delete (2), delete (8), delete (1), and then delete (4, 5, 6).

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 abs-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. To appear in *International Journal of Foundations of Computer Science*.
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.