SORTING BY BOUNDED BLOCK-MOVES

Lenwood S. Heath and John Paul C. Vergara

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061-0106

May 15, 1997

Abstract

Given a permutation π, a block-move is an operation that switches two adjacent blocks of elements in π. The problem of finding the minimum number of block-moves required to sort π has applications in computational biology, particularly in the study of genome rearrangements. This paper investigates variants of the problem where bounds are imposed on the lengths of the blocks moved. Algorithms and reduction results are presented for these variants.

1 Introduction

In molecular biology, two species are compared by analyzing the gene sequences in their respective genomes. Phylogenetic trees of these species are constructed based on information about how closely two species are related. A genome typically evolves by mutations that are insertions, deletions, duplications, reversals, and block-moves of its fragments. When comparing closely related species, it can be assumed that these mutations are such that fragments are neither introduced nor removed. One may now think of these mutations as *rearrangements* applied to a *permutation* of genes. A biologist wants to determine the fewest number of mutations required to transform one genome into another as an estimate of the distance between two species. This translates to an equivalent problem on permutations and rearrangements, given that the assumptions stated earlier apply. In fact, the problem can be restated as a sorting problem since the destination genome can be considered as the sorted permutation of genes and one seeks the minimum number of rearrangements required to sort a given permutation (source genome).

Most algorithmic results on this problem involve restricting the operations on a genome to a subset of the operations we have identified. For instance, there has been significant work on sorting by reversals [3, 4, 7, 10, 11]. In this paper, we focus on block-moves. Given a permutation $\pi = \pi_1\pi_2\ldots\pi_n$, and indices i, j, k satisfying $1 \leq i < j < k \leq n+1$, the *block-move* $\beta(i, j, k)$, has the effect of repositioning $\pi_i\pi_{i+1}\ldots\pi_{j-1}$ immediately to the left of element π_k. Equivalently, it has the effect of repositioning $\pi_j\pi_{j+1}\ldots\pi_{k-1}$ immediately to the left of π_i. Consider the permutation $\pi = 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9$, for example. Applying $\beta(3, 5, 8)$ to π transforms π to $\pi \cdot \beta = 1\ 2\ 5\ 6\ 7\ 3\ 4\ 8\ 9$. *Minimum sorting by block-moves* is the problem of determining the block-move distance of a permutation π, i.e., the minimum number of block-moves required to sort π.

Bafna and Pevzner [2] study this problem (they call block-moves transpositions since block-moves transpose two adjacent blocks) and provide a metric on a permutation that counts cycles in a graph representation of the permutation. They derive upper and lower bounds on the block-move distance and present an approximation algorithm with a 1.5 performance guarantee. Guyer, Heath, and Vergara [6] employ some heuristics based on subsequences and devise corresponding algorithms for the problem. They observe experimentally that selecting block-moves that maximally lengthen the longest increasing subsequence in the permutation often produces near-optimal results. It is not known whether this problem is NP-complete or solvable in polynomial time. Aigner and West [1] provide a polynomial-time algorithm for a variant of this problem where the block-moves are restricted to those that re-insert the first element in a permutation.

This paper considers several variants of the minimum sorting by block-moves problem so it is convenient to devise a generalized formulation for minimum sorting problems. Given a permutation π and a rearrangement α, let the product $\pi \cdot \alpha$ denote the resulting permutation when α is applied to π. Let P be a predicate on a rearrangement. A minimum sorting problem that allows only those rearrangements that satisfy P is formally stated as follows.

MINIMUM SORTING BY P $(MinSort_P)$

INSTANCE: A permutation $\pi = \pi_1\pi_2 \dots \pi_n$.

SOLUTION: A sequence of rearrangements $\alpha_1, \alpha_2, \dots, \alpha_k$ such that $P(\alpha_i)$ is true for all i with $1 \le i \le k$, the product $\pi \cdot \alpha_1 \cdot \alpha_2 \cdots \alpha_k$ is the identity permutation, and k is as small as possible.

The above formulation uses the predicate P to restrict the rearrangements to a particular set. Let $P[n]$ denote the set of rearrangements α on a permutation of length n where $P(\alpha)$ holds. We say P *describes* the set $P[n]$.

Even and Goldreich [5] show that $MinSort_P$, when P is considered part of the instance, is NP-hard. Jerrum [8] strengthens this result by showing that the problem is PSPACE-complete. This paper investigates $MinSort_P$ for fixed predicates P, in particular, those that describe block-moves.

Let Bk be the predicate that describes all block-moves, so $MinSort_{Bk}$ denotes minimum sorting by block-moves. In the block-move $\beta(i,j,k)$, adjacent blocks are switched, specifically, the block $\pi_i\pi_{i+1}\dots\pi_{j-1}$ and the block $\pi_j\pi_{j+1}\dots\pi_{k-1}$. $\beta(i,j,k)$ is an m-block-move if $m = k - i$; intuitively m represents the total length of both blocks. For an integer ℓ, the predicate Bk^ℓ describes the set of all m-block-moves, $m \le \ell$. Observe that $Bk^\ell[n] \subseteq Bk[n]$. $\beta(i,j,k)$ is also an (x,y)-block-move where x and y represent the individual lengths of the two blocks, $j-i$ and $k-j$, not necessarily in that order. For integers p and q, the predicate $Bk^{p,q}$ describes the set of all (x,y)-block-moves, where $x \le p$ and $y \le q$.

The simplest bounded case is *minimum sorting by adjacent element swaps.* Such swaps are just block-moves of length 2; that is, the problem is just $MinSort_{Bk^2} = MinSort_{Bk^{1,1}}$. $MinSort_{Bk^2}$ is solvable in polynomial time. In fact, the operations performed by the well-known sorting algorithm *Bubble-Sort* is precisely an optimal sequence of operations in this case.

A summary of the results found in this paper follows. In Section 2, we address $MinSort_{Bk^{1,n-1}}$, minimum sorting by single element re-insertion and provide a polynomial-time algorithm for this problem. Section 3 presents a polynomial-time reduction of $MinSort_{Bk}$ to $MinSort_{Bk^{\lfloor f(n) \rfloor}}$ where $f(n)$ is a bound proportional to n. Section 4 defines minimum sorting by short block-moves. We prove that there exists an optimal sorting sequence of short block-moves such that each block-move in the sequence corrects the relative order of the elements moved. We devise a graph-theoretic model based on this statement and present corresponding algorithms. We also show that the short block-move diameter (worst case distance between two permutations) is $\left\lceil \binom{n}{2} /2 \right\rceil$. Finally, we conclude in Section 5 where we state a conjecture.

2 Single Element Re-insertion

A block-move $\beta \in Bk^{1,n-1}[n]$ is the case where one of the blocks is a single element and the other block is unrestricted. Thus, $MinSort_{Bk^{1,n-1}}$ is the problem where we seek to sort a permutation such that the only allowed rearrangements are those that re-insert a single element. The following result shows that this problem is solvable in polynomial time.

Theorem 1 *$MinSort_{Bk^{1,n-1}}$ is solvable in $O(n^2)$ time.*

Proof: Let $\pi = \pi_1 \pi_2 \ldots \pi_n$ be a permutation. Consider a *longest increasing subsequence* (LIS) in π, i.e., a longest subsequence $\pi_{i_1} \pi_{i_2} \ldots \pi_{i_k}$ such that $1 \leq i_1 < i_2 < \ldots < i_k \leq n$ and $\pi_{i_j} < \pi_{i_{j+1}}$ for $1 \leq j < k$. Observe that a block-move from $Bk^{1,n-1}[n]$ may increase the length of the LIS in the permutation by at most one. The sorted permutation has a LIS of length n. Therefore, a straightforward algorithm that optimally sorts π using rearrangements from $Bk^{1,n-1}[n]$ simply inserts all elements not in the LIS in their proper positions. Consequently, the number of block-moves required to sort π is $n - |LIS(\pi)|$. The algorithm takes $O(n^2)$ time because it is dominated not by the computation of the LIS (computed only once) but by the insertion of elements. Inserting an element in a sequence takes $O(n)$ (as in the familiar Insertion-Sort algorithm) and the number of insertions may reach $n-1$. $\square$

If all that is required is the *length* of the sorting sequence, then obtaining this result is dependent only on the computation of the LIS, which takes $O(n \log \log n)$ time [9].

The results of Guyer, Heath, and Vergara [6] involve longest increasing subsequences. They experimentally determine that an algorithm for $MinSort_{Bk}$ based on LIS produces better results compared to those based on runs (merging consecutive and contiguous subsequences). In fact, the results appear nearly optimal for random permutations using the LIS-based algorithm.

3 A Reduction Result

In this section, we show that many bounded problems are at least as difficult as the original $MinSort_{Bk}$ problem. Consider $MinSort_{Bk\lfloor f(n) \rfloor}$ where $f(n)$ is some function proportional to n; specifically, $f(n) = n/r + c$ for some constant $r \geq 1$ and integer constant c. As an example, $MinSort_{Bk\lfloor n/2 \rfloor}$ is minimum sorting where the block-moves are such that the total lengths of the blocks do not exceed half the permutation length. The following result shows that there is a polynomial reduction from $MinSort_{Bk}$ to a problem such as $MinSort_{Bk\lfloor n/2 \rfloor}$.

Theorem 2 *Let $f(n) = n/r + c$ for some constant $r \geq 1$ and integer constant c. Then, $MinSort_{Bk}$ reduces in polynomial-time to $MinSort_{Bk\lfloor f(n) \rfloor}$.*

Proof: We construct a $MinSort_{Bk\lfloor f(n) \rfloor}$ instance from a $MinSort_{Bk}$ instance. Let $\pi = \pi_1 \pi_2 \ldots \pi_m$ be a $MinSort_{Bk}$ instance. Let m denote the length of the $MinSort_{Bk}$ instance to distinguish it from the length of the resulting instance.

Construct the $MinSort_{Bk\lfloor f(n) \rfloor}$ instance

$$\sigma = \pi_1 \pi_2 \ldots \pi_m \ m+1 \ m+2 \ \ldots \ n$$

where $n = \lceil f^{-1}(m) \rceil = \lceil (m-c)r \rceil$. Observe that in the bounded problem $MinSort_{Bk\lfloor f(n) \rfloor}$, the block-moves allowed are those whose lengths are at most

$$\lfloor f(n) \rfloor = \left\lfloor \frac{1}{r} \lceil (m-c)r \rceil + c \right\rfloor .$$

It can be verified that this value is just m, the length of the $MinSort_{Bk}$ instance π, from which σ is derived. Intuitively, σ is just π followed by additional elements already in their right order and position in the permutation. To prove that this reduction is appropriate, we need to show that a sorting sequence (under $MinSort_{Bk}$ restrictions) of length k for π exists if and only if a sorting sequence (under $MinSort_{Bk\lfloor f(n)\rfloor}$ restrictions) of length k for σ exists.

(Only if) Let $\beta_1, \beta_2, \ldots, \beta_k$ be a sorting sequence for π; that is,

$$\pi \cdot \beta_1 \cdot \beta_2 \cdots \beta_k = 1\ 2\ \ldots\ m.$$

This same sorting sequence applies to σ; that is,

$$\sigma \cdot \beta_1 \cdot \beta_2 \cdots \beta_k = 1\ 2\ \ldots\ m\ m+1\ m+2\ \ldots\ n,$$

because σ is just π concatenated with $m+1\ m+2\ \ldots\ n$. Each β_i repositions only those elements in σ that are also in π. The positions of these elements are preserved in the reduction so the same indices in each β_i apply. Furthermore, each β_i has length at most m, which follows $MinSort_{Bk\lfloor f(n)\rfloor}$ restrictions.

(If) Let $\beta_1, \beta_2, \ldots, \beta_k$ be a sorting sequence for σ; that is,

$$\sigma \cdot \beta_1 \cdot \beta_2 \cdots \beta_k = 1\ 2\ \ldots\ m\ m+1\ m+2\ \ldots\ n.$$

If all of the block-moves ($\beta_i, 1 \le i \le k$) involve only the elements $1\ 2\ \ldots\ m$, then the same sorting sequence applies to π. Unfortunately, this may not always be the case. Instead, we derive a block-move β'_i from each block-move β_i such that the result is a sorting sequence for π. The derivation is as follows. For every block-move β_i that switches two blocks of elements in σ, derive the block-move β'_i that switches the same blocks of elements in π ignoring elements $m+1\ m+2\ \ldots n$, since these elements do not appear in σ. Figure 1 demonstrates an example where β'_i is derived from β_i (observe that element 6 is ignored in the derivation). Since the original sequence sorts σ, the derived sequence sorts π. That is,

$$\pi \cdot \beta'_1 \cdot \beta'_2 \cdots \beta'_k = 1\ 2\ \ldots\ m,$$

$$
\begin{aligned}
\sigma \cdot \beta_i(2,4,7) &= 3\ \boxed{4\ 1}\ \boxed{5\ 2\ 6}\ 7\ 8\ 9\ 10\ \cdot\ \beta_i(2,4,7) \\
&= 3\ \boxed{5\ 2\ 6}\ \boxed{4\ 1}\ 7\ 8\ 9\ 10 \\
&\Longleftrightarrow \\
\pi \cdot \beta_i'(2,4,6) &= 3\ \boxed{4\ 1}\ \boxed{5\ 2}\ \cdot \beta_i'(2,4,6) \\
&= 3\ \boxed{5\ 2}\ \boxed{4\ 1}
\end{aligned}
$$

Figure 1: Deriving a $MinSort_{Bk}$ block-move from a $MinSort_{Bk^{n/2}}$ block-move.

which completes the proof. □

The result immediately extends to other bounds $f(n)$ particularly for the cases where $f(n)$ is proportional to $n^{1/k}$ for some fixed k, for instance, when $f(n) = \frac{1}{r}n^{1/k} + c$. The permutation for $MinSort_{Bk}$ is simply extended accordingly, that is the corresponding length of the $MinSort_{Bk\lfloor f(n) \rfloor}$ instance is

$$n = \lceil f^{-1}(m) \rceil = \lceil ((m-c)r)^k \rceil.$$

The value k being fixed guarantees that the reduction is polynomial.

Theorem 3 *Let $f(n)$ be some function proportional to $n^{1/k}$ for some fixed integer k. Then, $MinSort_{Bk}$ reduces in $O(n^k)$ time to $MinSort_{Bk\lfloor f(n) \rfloor}$.*

This implies that as the time complexity of $MinSort_{Bk}$ is not resolved, it is worthwhile to investigate $MinSort_{Bk^\ell}$ cases where ℓ is a fixed integer. The next section addresses one of these cases.

4 Short Block-moves

In this section, we investigate minimum sorting by short block-moves. We first begin with the definition of some terms. A *short block-move* is a block-move such that the total length of the blocks is at most 3 (a 3-block-move). A $(1,1)$-block-move is a *skip* (an element is moved one position away from its original position). A $(1,2)$-block-move (equivalently, a $(2,1)$-block-move) is a *hop* (an element is moved two positions away). Short block-moves consist of skips and hops and are described by the predicate Bk^3 ($Bk^{1,2}$ describe them as well).

In Section 4.1, we prove a statement concerning an optimal sorting sequence of short block-moves and the relative order between the elements in a permutation. Section 4.2 presents a graph model for this problem based on this statement. In Sections 4.3 and 4.4, we devise algorithms to solve $MinSort_{Bk^3}$ for special types of permutations. Finally, Section 4.5 establishes the short block-move diameter for S_n, the group of length-n permutations.

4.1 Relative order

Recall that when the predicate P describes rearrangements that swap adjacent elements ($P = Bk^2 = Bk^{1,1}$), $MinSort_P$ is polynomial-time solvable, and the sequence of operations performed in *Bubble-Sort* is always an optimal sorting sequence. In such a case, observe that any swap in this sequence corrects the relative order of the two adjacent elements. For example, in the permutation 1 4 2 5 3, an optimal sequence of rearrangements (assuming only adjacent swaps are allowed) swaps 4 and 2, 5 and 3, and finally 4 and 3. In all the three rearrangements, the relative order of the swapped elements are corrected.

A *correcting block-move* is a block-move $\beta(i,j,k)$ applied to some permutation π such that the following property holds: $\pi_a > \pi_b$ for all a and b where $i \leq a < j \leq b < k$. Intuitively, a correcting block-move corrects the relative order of all elements moved. We call a pair of elements in π not in their correct relative order an *inversion*. Observe that for correcting short block-moves, exactly one inversion is eliminated in the case of a skip, and exactly two inversions in the case of a hop. Non-correcting block-moves, on the other hand, *introduce* inversions. We can therefore associate a metric $inversions(\pi,\beta)$ to a block-move β that counts the number inversions that β introduces. Correcting short block-moves have $inversions(\pi,\beta) = 0$ while non-correcting short block-moves have $inversions(\pi,\beta) \in$

$\{1,2\}$. The following is a result involving this metric.

Lemma 1 *Let π be a permutation, and let $\beta_1, \beta_2, \ldots, \beta_k$ be a sequence of short block-moves that sorts π. Furthermore, let i be the smallest integer such that β_i is a non-correcting block-move. Finally, let π^{i-1} be the permutation after the first $i-1$ block-moves are applied to π. Then, there exists an alternate sorting sequence of short block-moves with the following properties:*

- *The sequence begins with $\beta_1, \beta_2, \ldots, \beta_{i-1}, \hat{\beta}_i, \ldots$, where*

$$inversions(\pi^{i-1}, \hat{\beta}_i) = inversions(\pi^{i-1}, \beta_i) - 1.$$

- *The length of the sequence is at most k.*

Proof: Given the sequence of short block-moves $\beta_1, \beta_2, \ldots, \beta_k$ that sorts π, let $\pi^r = \pi \cdot \beta_1 \cdot \beta_2 \cdots \beta_r$ denote the permutation after the first r block-moves have been applied to π; in particular, let $\pi^0 = \pi$.

Since β_i is a non-correcting block-move, there are two elements a, b in π such that:

1. $a < b$,
2. a occurs to the left of b in π^{i-1}, and,
3. a occurs to the right of b in $\pi^i = \pi^{i-1} \cdot \beta_i$.

Table 1 enumerates the possible positions of a and b in π^{i-1} and π^i. The goal is to replace β_i with $\hat{\beta}_i$ as enumerated in Table 2. The revised block-move $\hat{\beta}_i$ does not destroy the already correct relative order between elements a and b. That is, for all cases enumerated, $inversions(\pi^{i-1}, \hat{\beta}_i) = inversions(\pi^{i-1}, \beta_i) - 1$. As a consequence, the permutations π^i and $\hat{\pi}^i = \pi^{i-1} \cdot \hat{\beta}_i$ are different only in that a and b are switched. We note Case 1 as an exception in this discussion since it is the case where β_i is a skip, so that it suffices to simply omit β_i instead of replacing it with some $\hat{\beta}_i$. Now, since the sequence of block-moves $\beta_1, \beta_2, \ldots, \beta_k$ eventually sorts the permutation π, there is a subsequent block-move

Case	π^{i-1}	β_i	$\pi^i = \pi^{i-1} \cdot \beta_i$
1	$\ldots\ a\ b\ \ldots$	$\beta_i(p, p+1, p+2)$	$\ldots\ b\ a\ \ldots$
2	$\ldots\ a\ x\ b\ \ldots$	$\beta_i(p, p+1, p+3)$	$\ldots\ x\ b\ a\ \ldots$
3	$\ldots\ a\ x\ b\ \ldots$	$\beta_i(p, p+2, p+3)$	$\ldots\ b\ a\ x\ \ldots$
4	$\ldots\ x\ a\ b\ \ldots$	$\beta_i(p, p+2, p+3)$	$\ldots\ b\ x\ a\ \ldots$
5	$\ldots\ a\ b\ x\ \ldots$	$\beta_i(p, p+1, p+3)$	$\ldots\ b\ x\ a\ \ldots$

Table 1: Possibilities for non-correcting short block-moves.

Case	π^{i-1}	$\hat{\beta}_i$	$\hat{\pi}^i = \pi^{i-1} \cdot \hat{\beta}_i$
1	$\ldots\ a\ b\ \ldots$	—	$\ldots\ a\ b\ \ldots$
2	$\ldots\ a\ x\ b\ \ldots$	$\hat{\beta}_i(p, p+1, p+2)$	$\ldots\ x\ a\ b\ \ldots$
3	$\ldots\ a\ x\ b\ \ldots$	$\hat{\beta}_i(p+1, p+2, p+3)$	$\ldots\ a\ b\ x\ \ldots$
4	$\ldots\ x\ a\ b\ \ldots$	$\hat{\beta}_i(p, p+1, p+2)$	$\ldots\ a\ x\ b\ \ldots$
5	$\ldots\ a\ b\ x\ \ldots$	$\hat{\beta}_i(p+1, p+2, p+3)$	$\ldots\ a\ x\ b\ \ldots$

Table 2: Alternate short block-moves $\hat{\beta}_i$ for β_i.

β_j $(j > i)$ that returns elements a and b to their correct relative order. Table 3 enumerates the possibilities. Consider $\hat{\pi}^{j-1} = \hat{\pi}^i \cdot \beta_{i+1} \cdots \beta_{j-1}$. Since $\hat{\pi}^i$ is just π^i with elements a and b switched, $\hat{\pi}^{j-1}$ is just π^{j-1} with elements a and b switched.

We replace β_j with a block-move $\hat{\beta}_j$ that leaves a and b in their correct relative positions. Table 4 enumerates the corresponding $\hat{\beta}_j$ for each β_j in Table 3. Again, note the exception for Case 1 where $\hat{\beta}_j$ can be viewed as an identity block-move. Confirm that for all cases in the table, the following now holds: $\hat{\pi}^{j-1} \cdot \hat{\beta}_j = \pi_j$. That is, replacing block-moves β_i and β_j with $\hat{\beta}_i$ and $\hat{\beta}_j$ yields the same permutation. Notice that the number of block-moves stays the same (or decreases, if Case 1 holds) but the offending block-move β_i has been revised so that it no longer places a and b in their incorrect relative order. The transformation is summarized in the following equation:

$$\pi \cdot \beta_1 \cdot \beta_2 \cdots \beta_k = \pi \cdot \beta_1 \cdot \cdots \beta_{i-1} \cdot \hat{\beta}_i \cdot \beta_{i+1} \cdots \beta_{j-1} \cdot \hat{\beta}_j \cdot \beta_{j+1} \cdots \beta_k,$$

Case	π^{j-1}	β_j	$\pi^j = \pi^{j-1} \cdot \beta_j$
1	$\ldots\ b\ a\ \ldots$	$\beta_j(q, q+1, q+2)$	$\ldots\ a\ b\ \ldots$
2	$\ldots\ b\ y\ a\ \ldots$	$\beta_j(q, q+1, q+3)$	$\ldots\ y\ a\ b\ \ldots$
3	$\ldots\ b\ y\ a\ \ldots$	$\beta_j(q, q+2, q+3)$	$\ldots\ a\ b\ y\ \ldots$
4	$\ldots\ y\ b\ a\ \ldots$	$\beta_j(q, q+2, q+3)$	$\ldots\ a\ y\ b\ \ldots$
5	$\ldots\ b\ a\ y\ \ldots$	$\beta_j(q, q+1, q+3)$	$\ldots\ a\ y\ b\ \ldots$

Table 3: Possibilities where β_j corrects the order of a and b.

Case	$\hat{\pi}^{j-1}$	$\hat{\beta}_j$	$\hat{\pi}^{j-1} \cdot \hat{\beta}_j = \pi_j$
1	$\ldots\ a\ b\ \ldots$	—	$\ldots\ a\ b\ \ldots$
2	$\ldots\ a\ y\ b\ \ldots$	$\hat{\beta}_j(q, q+1, q+2)$	$\ldots\ y\ a\ b\ \ldots$
3	$\ldots\ a\ y\ b\ \ldots$	$\hat{\beta}_j(q+1, q+2, q+3)$	$\ldots\ a\ b\ y\ \ldots$
4	$\ldots\ y\ a\ b\ \ldots$	$\hat{\beta}_j(q, q+1, q+2)$	$\ldots\ a\ y\ b\ \ldots$
5	$\ldots\ a\ b\ y\ \ldots$	$\hat{\beta}_j(q+1, q+2, q+3)$	$\ldots\ a\ y\ b\ \ldots$

Table 4: Revised block-moves $\hat{\beta}_j$ for β_j.

where $inversions(\pi^{i-1}, \hat{\beta}_i) = inversions(\pi^{i-1}, \beta_i) - 1$. □

The above lemma allows us to transform an optimal sorting sequence into a sequence that contains only correcting block-moves. The goal is to show that only correcting block-moves need to be considered when solving $MinSort_{Bk^3}$.

Theorem 4 *For a permutation π, there exists an optimal sequence of short block-moves $\beta_1, \beta_2, \ldots, \beta_k$ that sorts π such that each block-move is a correcting block-move.*

Proof: We convert the given optimal sequence to an alternate sequence of correcting short block-moves by repeatedly applying the transformation process given in Lemma 1. The conversion replaces a non-correcting block-move β_i by a correcting block-move in either one or two applications of Lemma 1, depending on the value of $inversions(\pi^{i-1}, \beta_i)$. Since the

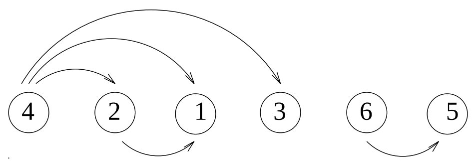

Figure 2: The permutation graph for $\pi = 4\ 2\ 1\ 3\ 6\ 5$.

block-moves $\beta_1, \beta_2, \ldots, \beta_{i-1}$ are unchanged and remain as correcting block-moves at each step, the conversion process will eventually terminate.

It is enlightening to note that Case 1 never holds for either β_i or β_j in any step of the conversion, since this would imply a shorter resulting sorting sequence (a contradiction, since we start with an optimal sequence). $\square$

4.2 A graph-theoretic model

Using Theorem 4 from the previous section, it is sufficient to consider only correcting block-moves. This leads to the following graph representation of a permutation π.

The *permutation graph* of π is the directed graph $G_\pi^p = (V, A)$ where $V = \{\pi_1, \pi_2, \ldots, \pi_n\}$ and $(\pi_i, \pi_j) \in A$ whenever $\pi_i > \pi_j$ and $i < j$. Intuitively, the arcs of a permutation graph indicate those pairs of elements that are not in their correct relative order. Figure 2 illustrates the permutation graph for $\pi = 4\ 2\ 1\ 3\ 6\ 5$. Permutation graphs have been studied extensively studied (e.g., [12, 13]) and are examples of transitive graphs.

It can be verified that a correcting short block-move corresponds to the removal of one or two arcs from the corresponding permutation graph. Specifically, a skip corresponds to the removal of a single arc in the graph (i.e., the relative order between the two elements is corrected). On the other hand, a hop corresponds to the removal of two adjacent arcs in the graph since two relationships are corrected. In either case, the other arcs in the graph are not affected since the relationships between the other vertices remain the same. Consider again the permutation $\pi = 4\ 2\ 1\ 3\ 6\ 5$, whose permutation graph is shown in

Block-move	Arcs	Permutation graph
$\beta_1(1,3,4)$	$(4,1),(2,1)$	Figure 3a
$\beta_2(2,3,5)$	$(4,2),(4,3)$	Figure 3b
$\beta_3(5,6,7)$	$(6,5)$	Figure 3c

Table 5: Arcs that correspond to an optimal block-move sequence ($\pi = 4\ 2\ 1\ 3\ 6\ 5$).

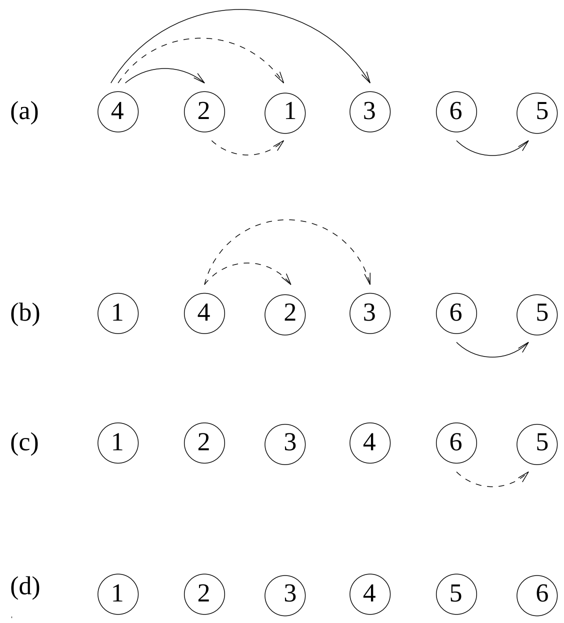

Figure 3: Effects of correcting short block-moves on a permutation graph.

Figure 2. An optimal sorting sequence for π is:

$$\beta_1(1,3,4), \beta_2(2,3,5), \beta_3(5,6,7).$$

Table 5 shows the correspondences of these block-moves with the arcs of G_π^p and Figure 3 illustrates the effect on the permutation graph of π. The following proposition summarizes the potential use of this graph-theoretic model.

Proposition 1 *Let π be a permutation, and let $\beta_1, \beta_2, \ldots, \beta_k$ be an optimal sequence of correcting short block-moves that sorts π. The sequence corresponds to a set D_π of arc-*

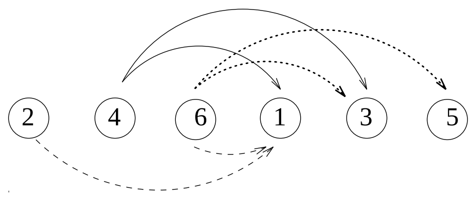

Figure 4: $(2,1)-(6,1)$ cannot be realized because of element 4.

disjoint subgraphs of G^p_π where each subgraph is either a single arc or a pair of adjacent arcs.

The set D_π in Proposition 1 is called an *arc-partition* of G^p_π, since it partitions the arcs of the graph into single arcs or pairs of adjacent arcs. The idea is to reverse the process, that is, to first obtain the edge-partition and then derive the optimal sequence. It should be evident that the goal is to maximize the number of arc-pairs in the partition since this minimizes the total number of subgraphs in G^p_π, and, equivalently, number of block-moves that sort π. We obtain this lower bound.

Proposition 2 *Let π be a permutation, and let $|A|$ denote the number of arcs in G^p_π. Then, the length of an optimal sequence of block-moves that sorts π is at least $\lceil |A|/2 \rceil$.*

The pairing of arcs in G^p_π is not arbitrary. In fact, the only arcs that can be paired are those whose heads are identical or whose tails are identical so that the pair may correspond to a hop in a sorting sequence. In addition, it is possible that a pair of arcs that satisfy this property may not be realized as a hop. Consider the example in Figure 4. The arc-pair $(2,1)-(6,1)$ can not be a realized as a hop because the element 4 exists in between elements 2 and 6 and will therefore not "get out of the way" since it is already in its correct relative order with respect to 2 and 6. We are, of course, making the important assumption that only correcting block-moves are acceptable in a sorting sequence.

A strategy would therefore be to construct yet another graph G^a_π, which we call the *arc graph* of π. G^a_π is an undirected graph whose vertices are the arcs of G^p_π and whose edges connect two vertices whenever the arcs can be paired as a *feasible hop*. Arc-pairs are

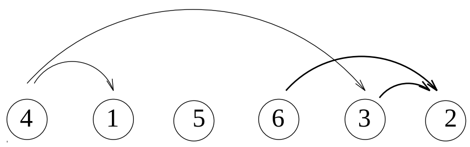

Figure 5: $(4,1)-(4,3)$ is disabled when $(6,2)-(3,2)$ is applied.

feasible hops when the arcs have identical heads or tails, and the situation illustrated in Figure 4 does not occur.

We begin by obtaining a maximum matching in G^a_π (recall that the intention is to maximize the hops, i.e., arc-pairs). Then, from the matching, we derive the corresponding set D_π of subgraphs in G^p_π, and then finally we derive an optimal sorting sequence for π. Recall that matched arcs correspond to hops, unmatched ones to skips. Unfortunately, the feasibility of arc-pairs is not "static"; that is, when a sorting sequence is derived from D_π and then applied to π, it is possible to *disable* an arc-pair. Consider the permutation in Figure 5. When the hop for arc-pair $(6,2)-(3,2)$ is applied, element 2 is placed between 1 and 6, thereby making the arc-pair $(4,1)-(4,3)$ infeasible.

The counterexample just given is discouraging since a potentially polynomial-time strategy involving maximum matching does not work in all cases. However, this graph-theoretic strategy does give us a lower bound better than that given in Proposition 2 for the length of an optimal sorting sequence for a permutation.

Proposition 3 *Let π be a permutation, and let M be a maximum matching in the arc graph G^a_π of π. Furthermore, let U denote the set of unmatched vertices in the matching. Then, the length of an optimal sequence of block-moves that sorts π is at least $|M|+|U|$.*

In the next two sections, we show that the graph model described here is accurate for particular types of permutations.

4.3 Sorting a Bitonic Permutation

A *bitonic* permutation is a permutation that consists of two subsequences, an increasing one and a decreasing one. A *concave-bitonic* permutation is a permutation that first has the increasing subsequence followed by the decreasing one. A *convex-bitonic* permutation is a permutation that first has the decreasing subsequence followed by the increasing one. We deal with concave-bitonic permutations first and then argue later on that the discussion applies for convex-bitonic permutations, by symmetry.

Let $indegree(\pi, e)$ denote the in-degree of an element e in the permutation graph G^p_π. Clearly, $\sum_{i=1}^{n} indegree(\pi, \pi_i) = |A|$, the total number of arcs in G^p_π. An element $e = \pi_p$ with $indegree(\pi, e) = d$ is a *left-pending* element if for all $e' \in \{\pi_{p-d}, \pi_{p-d+1} \ldots \pi_{p-1}\}$, $e' > e$. That is, there are d contiguous elements to the immediate left of element e whose values are greater than e. Left-pending elements are important in that these elements may be readily positioned by applying correcting block-moves that move the element to the left. We state the following lemma without proof.

Lemma 2 *Let π be a permutation with left-pending element $e = \pi_p$ and $indegree(\pi, e) = d$. Suppose d is even. Then, the sequence of correcting hops,*

$$\beta_1(p-2, p, p+1), \beta_2(p-4, p-2, p-1), \ldots, \beta_{d/2}(p-d, p-d+2, p-d+3),$$

when applied to π, results in a permutation $\pi' = \pi \cdot \beta_1 \cdot \beta_2 \cdots \beta_{d/2}$, where $indegree(\pi', e) = 0$.

The reduction of $indegree(\pi, e)$ from d to 0 corresponds to the elimination of arcs in G^p_π. From Proposition 2 in the previous section, this is desirable since this potentially leads to a sorting sequence of length $\lceil |A|/2 \rceil$ provided all the elements are left-pending and there is a way to deal with odd in-degree elements. For bitonic permutations this property is possible.

Consider the concave-bitonic permutation $\pi = 3\ 4\ 5\ 9\ 8\ 7\ 6\ 2\ 1$. In the following diagram, the second row indicates the in-degrees for the elements in π (those with no values have 0 in-degree).

π:	3	4	5	9	8	7	6	2	1
indegree:					1	2	3	7	8
Stage 1:	3	4	5	7	9	8	6	2	1
				0		1	3	7	8
Stage 2:	1	3	4	5	7	9	8	6	2
	0						1	3	7
Stage 3:	1	3	4	5	7	8	6	9	2
						0	2		7
Stage 4:	1	3	4	5	6	7	8	9	2
					0				7
Stage 5:	1	3	2	4	5	6	7	8	9
			1						
Stage 6:	1	2	3	4	5	6	7	8	9
		0							

Figure 6: Bitonic Sorting Example.

π:	3	4	5	9	8	7	6	2	1
indegree:					1	2	3	7	8

Observe that the elements in the increasing sequence all have in-degree 0. Also, all of the elements in the decreasing sequence (not including the peak element) have in-degree $d > 0$ and are left-pending elements. This suggests an algorithm to optimally sort π. Figure 6 shows a trace of how to sort π for this example. First, move all even in-degree elements to the left using hops (as in Lemma 2). In the example, we move element 7 (in-degree 2) and 1 (in-degree 8), using 1 and 4 hops, respectively. These are shown in separate stages (Stages 1 and 2) in the figure. At this point, unless the permutation is already sorted, π is still bitonic, and only elements with odd in-degrees remain (in the decreasing portion of the permutation). The parities of two consecutive elements with odd in-degrees can be "fixed" (made even) using a single hop. This is done by moving both elements one position to the left, thereby updating both in-degrees to an even number. After this, Lemma 2 now applies individually to both elements. Using our example, elements 8 and 6 are processed

BITONIC-SORT(π). Minimum sorting by short block-moves for concave-bitonic permutations.

INPUT: A bitonic permutation π.
OUTPUT: The size of the shortest sorting sequence of block-moves.

$(e_1, e_2, \ldots, e_x) \leftarrow$ all elements e_i in π such that $indegree(\pi, e_i)$ is even
for $i \leftarrow 1$ **to** x **do**
    $(a_i, \pi) \leftarrow$ LEFTHOPS(π, e_i)
$(e_1, e_2, \ldots, e_y) \leftarrow$ all elements e_i in π such that $indegree(\pi, e_i)$ is odd
$y' \leftarrow y$ DIV 2
for $i \leftarrow 1$ **to** y' **do**
    begin
        $\pi \leftarrow$ FIXPARITY(π, e_{2i-1}, e_{2i})
        $(b_i, \pi) \leftarrow$ LEFTHOPS(π, e_{2i-1})
        $(c_i, \pi) \leftarrow$ LEFTHOPS(π, e_{2i})
    end
$d_y \leftarrow 0$
if y is odd **then**
    begin
        $(d_y, \pi) \leftarrow$ LEFTHOPS(π, e_y)
        $\pi \leftarrow$ LEFTSKIP(π, e_y)
        $d_y \leftarrow d_y + 1$
    end
return $\sum_{i=1}^{x} a_i + \sum_{i=1}^{y'} (1 + b_i + c_i) + d_y$

Figure 7: Algorithm BITONIC-SORT.

next as illustrated in Stage 3 (parities are fixed) and Stage 4 (6 is moved to the left) in the figure. The above suggests processing the odd-degree elements in consecutive pairs leaving the possibility of an extra odd-degree element. This is handled by hops to the left (Stage 5) plus a single skip (Stage 6), as illustrated by element 2 in our example (3 hops + 1 skip).

Algorithm BITONIC-SORT is shown in Figure 7. The algorithm uses three subroutines: LEFTHOPS, FIXPARITY, and LEFTSKIP, described as follows.

LEFTHOPS(π, e). If e is a left-pending element with in-degree d, then this function applies $\lfloor d/2 \rfloor$ hops to π, specifically, correcting hops that move e to its left. This function returns the number of hops applied to π as well as the permutation that results.

FIXPARITY(π, e_1, e_2). This function returns the permutation that results after consecutive elements e_1 and e_2 in π are moved one element to the left. Suppose $\pi_p = e_1$ ($\pi_{p+1} = e_2$). Then, the precise block-move applied to π is $\beta(p-1, p, p+2)$. If π_{p-1} is greater than both e_1 and e_2, then the parities of the in-degrees of e_1 and e_2 are shifted after the operation.

LEFTSKIP(π, e). This function returns the permutation that results after element e in π is moved one element to the left.

There are three parts in the algorithm. The first part applies Lemma 2 to all elements with even in-degrees (lines 1–3). The second part positions the remaining odd in-degree elements (lines 4-11). In this part, the elements are grouped into consecutive pairs, their parities are shifted from odd to even by a single hop (line 8), and then positioned accordingly using Lemma 2 (lines 9 and 10). The third part (lines 12–18) handles the case where an unpaired odd in-degree element remains. Here, it is positioned using hops (line 15) and a single skip (line 16). Line 19 returns the total number of operations performed in all three parts.

Theorem 5 *Let π be a concave-bitonic permutation, and let $|A|$ be the number of arcs in G^p_π. Algorithm* BITONIC-SORT *returns $\lceil |A|/2 \rceil$, the length of an optimal sorting sequence of short block-moves for π.*

Proof: In a concave-bitonic permutation, all elements in π with in-degree $d > 0$ are left-pending elements. This allows us to use Lemma 2 whenever d is even. In particular, in the first part of Algorithm BITONIC-SORT, LEFTHOPS will perform exactly $a_i = d/2$ correcting hops in line 3 of the algorithm, thereby eliminating d arcs from G^p_π. In the second part of the algorithm, the elements concerned have odd in-degree and this is remedied by a call to FIXPARITY, where, in a single correcting hop, a pair of consecutive elements both become of even in-degree. If the original in-degrees of these elements are d_1 and d_2, then line 8 performs 1 hop (fixing the parities of these elements), line 9 performs $\lfloor d_1/2 \rfloor$ hops, and line 10 performs $\lfloor d_2/2 \rfloor$ hops. This amounts to $(d_1 + d_2)/2$ hops and the elimination of $d_1 + d_2$ arcs from the permutation graph. Finally, the third part of the algorithm is the case where an unpaired odd in-degree element remains. If the in-degree of the element is d, then line 15 performs $d_y = \lfloor d/2 \rfloor$ hops eliminating $d - 1$ arcs. This still leaves a single arc which is eliminated by a call to LEFTSKIP in line 16, after which the permutation is sorted (no arcs remain in G_π^p).

Since every operation except perhaps the last one is a correcting hop and eliminates 2 arcs from the G_π^p, then the algorithm performs $\lceil |A|/2 \rceil$ short block-moves (an additional skip is necessary if $|A|$ is odd). From Proposition 2, it follows that the result returned by Algorithm BITONIC-SORT is optimal. □

Theorem 5 (and its proof) applies to convex-bitonic permutations as long as the symmetry is correctly handled. In particular, for convex-bitonic permutations we deal with *out-degrees* instead of in-degrees, *right-pending* instead of left-pending elements, and hops and skips to the right (not left).

4.4 Sorting Two Increasing Sequences

Section 4.2 suggests that, to solve $MinSort_{Bk^3}$, first obtain a maximum matching in the arc graph G_π^a (feasible arc-pairs from G_π^p), and then derive an optimal sequence of short block-moves from the matching. Unfortunately, the derived sequence might not be "realizable" because of the possibility that a hop corresponding to an arc-pair can be disabled by other block-moves. The bitonic permutations of Section 4.3 are an exception—the solution presented there does imply a maximum matching (all arcs are matched except perhaps one) and the corresponding sequence can indeed be realized. In this section, we present another type of permutations where it can be guaranteed that the corresponding derived sequence can be realized.

A *2-strip-increasing* permutation is a permutation that consists of two consecutive in-

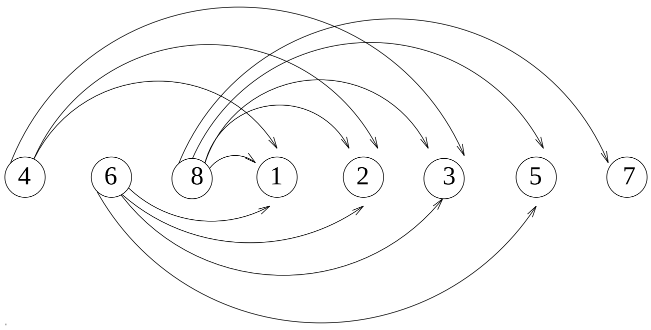

Figure 8: The permutation graph for $\pi = 4\ 6\ 8\ 1\ 2\ 3\ 5\ 7$.

creasing sequences. The permutation $\pi = 4\ 6\ 8\ 1\ 2\ 3\ 5\ 7$ is 2-strip-increasing. Figure 8 illustrates the permutation graph G_π^p for this example. Permutation graphs for 2-strip-increasing permutations have the following special property: all arcs (x, y) are such that x is in the first increasing sequence and y is in the second sequence. Feasible arc-pairs are also straightforward to determine. They take the following forms:

- $(a, b) - (a, c)$ where a is from the first sequence and b, c are elements that appear consecutively in the second sequence, or,
- $(b, a) - (c, a)$ where a is from the second sequence and b, c are elements that appear consecutively in the first sequence.

In both cases above, element a is either the common head or the common tail of both arcs. Elements b and c have to appear consecutively because if they did not, the arc-pair would be infeasible as an element between b and c would exist that is already in its correct relative order with elements a and b (because the sequence that contain these elements is increasing). In Figure 8, $(6, 3) - (6, 5)$ and $(6, 2) - (8, 2)$ are feasible arc-pairs, while $(4, 2) - (8, 2)$ is not because of element 6.

As discussed in Section 4.2, feasible arc-pairs may be disabled, that is, it is possible that there is a sequence of correcting block-moves that, when applied to π will cause some element x satisfying $b < x < c$ to occur between elements b and c, thereby rendering the arc-pair an infeasible hop. The following lemma shows that this does not occur for 2-strip-increasing permutations.

Lemma 3 *Let π be a 2-strip-increasing permutation. For a given feasible arc-pair in G^p_π, there is no sequence of correcting short block-moves that, when applied to π, will cause the arc-pair to be disabled.*

Proof: Let $(a,b)-(a,c)$ be a feasible arc-pair. Since (a,b) and (a,c) are arcs in G^p_π, the relationships $a > b$ and $a > c$ hold, by definition. We show that an element x satisfying $b < x < c$ can never occur between element b and c as a result of other correcting block-moves. Such an element x must be less than a (by transitivity). Furthermore, given the structure of the permutation, the only place x can appear is to the left of a in the first increasing sequence. This means that such an element is already in its correct relative order with a so that a sequence of correcting block-moves will never place x to the right of a. However, elements b and c are to the right of a so x can never occur between b and c unless the hop that corresponds to arc-pair $(a,b)-(b,c)$ has already been applied to the permutation. A symmetric argument applies to the other form for an arc-pair, $(b,a)-(c,a)$.

□

Since disabling can be prevented as long as correcting short block-moves are performed, we now explore the possibility of a matching-based algorithm. We start with a matching M in the arc graph G^a_π of a 2-strip-increasing permutation π. M yields a set D_π of arc-pairs (matched arcs) and single arcs (unmatched) in G^p_π. We call these arcs and arc-pairs *potential* block-moves and classify them as follows:

- The single arc (a,b) is the potential skip $B(a \sim b)$.
- The arc pair $(a,b)-(a,c)$ is the potential *right* hop $B(a \sim b, c)$. Here, we call arc (a,c) the *longer* arc.
- The arc pair $(b,a)-(c,a)$ is the potential *left* hop $B(b, c \sim a)$. Here, (b,a) is the longer arc.

For convenience, the arc (a, b) for a skip may be called the longer arc of that skip. It is therefore appropriate to view the set D_π of subgraphs in G_π^p as a set of potential block-moves $\{B_1, B_2, \ldots, B_{|D_\pi|}\}$. The notation given above is convenient because we now describe a block-move in terms of the elements (values) that it rearranges. Our previous notation for block moves specifies positions instead of values.

Let $pos(\pi, e)$ denote the integer i such that $\pi_i = e$ (the position of e in π). The *length* of an arc (a, b) is the value $pos(\pi, b) - pos(\pi, a)$. The *range* of an arc (a, b) is the interval $[pos(\pi, a), pos(\pi, b)]$. The *span* of a potential block-move B in π is the range of its longer arc. The *length* of a potential block-move is the length of its longer arc. We use the following function names for the properties defined above: $length(\pi, arc)$, $range(\pi, arc)$, $span(\pi, B)$, and $Blength(\pi, B)$. A potential skip in π is an *actual* skip if its length is 1. A potential hop in π is an *actual* hop if its length is 2. Any actual skip or hop can be immediately applied to π.

Up to this point, we have managed to "fix" π. That is, all definitions and properties related to potential block-moves is dependent only on the original 2-strip-increasing permutation π. In addition, there is a set D_π of subgraphs derived from some given matching M that has remained fixed in the preceding discussion. In the following discussion, we start applying actual block-moves to π that affects D_π.

Let π be a 2-strip-increasing permutation. The product $\pi \cdot B$, where B is an actual block-move, is the result of applying the block-move B to π. B is described by elements so we define its effect as switching the elements separated by the $\sim$ in B's parameter list. The product $\pi \cdot B_1 \cdot B_2 \cdots B_k$ is a *realizable* product if each B_i is an actual block-move with respect to π^{i-1}, that is $Blength(\pi^{i-1}, B_i) = 1$, if B_i is a skip, or $= 2$, if B_i is a hop. Note that π^r has a similar definition as before, that is, it is the permutation that results after the first r block-moves have been applied to π. As block-moves are applied to π, inversions are corrected and, consequently, arcs or pairs of arcs are removed from G_π^p. Considering this, define $D_\pi^i = D_\pi - \{B_1, B_2, \ldots, B_i\}$, that is, D_π^i contains the remaining potential block-moves (those that have not been realized) after the specified block-moves

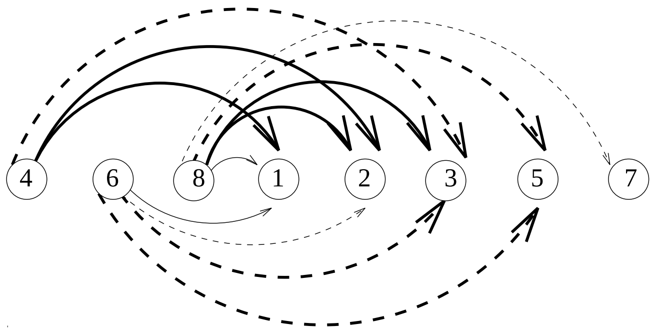

Figure 9: A maximum matching of feasible arc-pairs

Block-move	Type	Span	Arc-ranges
$B_1(4 \sim 1, 2)$	Hop (bold)	[1,5]	[1,4],[1,5]
$B_2(4, 6 \sim 3)$	Hop (bold-dashed)	[1,6]	[1,6],[2,6]
$B_3(6, 8 \sim 1)$	Hop (regular)	[2,4]	[2,4],[3,4]
$B_4(6 \sim 2)$	Skip (dashed)	[2,5]	[2,5]
$B_5(6, 8 \sim 5)$	Hop (bold-dashed)	[2,7]	[2,7],[3,7]
$B_6(8 \sim 2, 3)$	Hop (bold)	[3,6]	[3,6],[3,5]
$B_7(8 \sim 7)$	Skip (dashed)	[3,8]	[3,8]

Table 6: Potential block-moves derived from a given matching.

have been applied to π. The goal is to convert the set D_π to a realizable sorting sequence, so we want to reach a point where $D_\pi^k = \{\}$.

Consider Figure 9 which illustrates a matching (in fact, a maximum matching) of arcs in the permutation graph given in Figure 8. The figure also exhibits the corresponding set D_π of arcs and arc-pairs (or potential block-moves). The unmatched arcs are drawn in regular dashed lines, while the arc-pairs are given by regular lines, bold lines, and bold-dashed lines. The corresponding potential block-moves for the example in the figure are listed in Table 6. The table also shows values for spans and ranges for each potential block-move.

The *dependency* graph $G^d(\pi, D_\pi)$ of a given permutation π and a set D_π is a directed graph whose vertices are the potential block-moves in D_π. The directed edges are all the

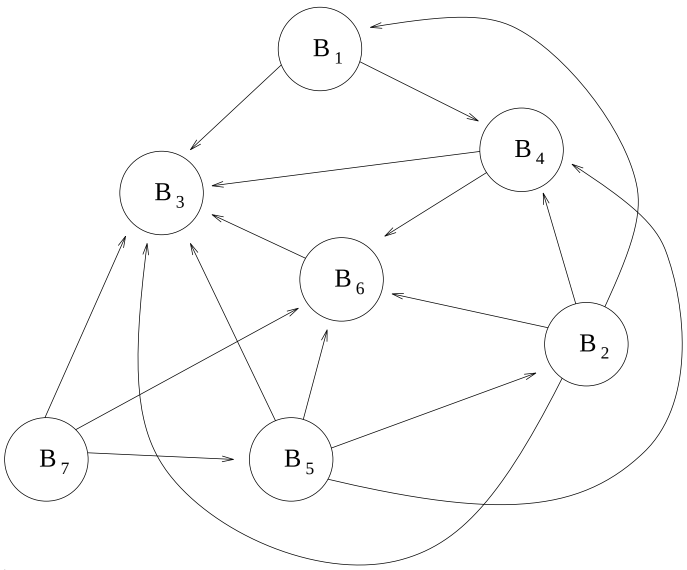

Figure 10: Dependency graph of potential block-moves.

(B_i, B_j) such that the range of an arc in B_j is contained in the span of B_i. Figure 10 illustrates the dependency graph for the list of block-moves given in Table 6.

We are now ready to present an algorithm for sorting 2-strip-increasing permutations. Algorithm MATCH-SORT, shown in Figure 11, is an algorithm that returns a sequence of correcting short block-moves that sorts the given permutation π. The algorithm first obtains a matching of G^a_π (line 1), derives the set of potential block-moves D_π (line 2), and then constructs the dependency graph H for these block-moves (line 3). The algorithm then proceeds with repeatedly selecting a block-move that does not depend on other block-moves (line 8), applies this to the permutation (line 9), and then removes the block-move from the dependency graph (line 10). At the end, the graph H is empty and the sorting sequence obtained is returned (line 12).

Consider the example $\pi = 4\ 6\ 8\ 1\ 2\ 3\ 5\ 7$, illustrated in the previous figures. The dependency graph for this example is shown in Figure 10. If we execute Algorithm MATCH-SORT with this permutation, the sequence of block-moves is $B_3, B_6, B_4, B_1, B_2, B_5, B_7$. Note that, as potential block-moves are selected, they are removed from the current dependency

Match-Sort(π). Minimum sorting by short block-moves using arc matching.

INPUT: A 2-strip-increasing permutation π.
OUTPUT: The shortest sorting sequence of short block-moves, $B_1, B_2, \ldots, B_k$.

$M \leftarrow$ a maximum matching in G^a_π
$D_\pi \leftarrow$ (set of arc-pairs in M) $\cup$ (unmatched arcs in G^a_π).
$H \leftarrow G^d(D_\pi, \pi)$ $\triangleright$ construct the dependency graph of D_π with respect to π
$k \leftarrow 0$
while $|V_H| \neq 0$ **do**
    begin
        $k \leftarrow k + 1$
        $B_k \leftarrow$ vertex $B \in V_H$ such that out-degree of B in H is 0
        $\pi \leftarrow \pi \cdot B_k$
        $H \leftarrow H - B$
    end
return $B_1, B_2, \ldots, B_k$

Figure 11: Algorithm Match-Sort.

graph It can be verified that the sequence indicated sorts π.

To prove correctness for this algorithm, three assertions must be shown:

1. When a potential block-move in the dependency graph has out-degree 0, then it is an actual block-move;

2. It suffices to compute the dependency graph just once; new dependencies are never created; and,

3. There are no cycles in the dependency graph so that there will always be a vertex with out-degree 0.

We prove these three assertions separately, and then combine the results in a theorem at the end of this section.

The following lemma allows us to detect the existence of an actual block-move using the dependency graph.

Lemma 4 *Let π be a 2-strip-increasing permutation, and let D_π be a corresponding set of potential block-moves. Furthermore, suppose the product $\pi^i = \pi \cdot B_1 \cdot B_2 \cdots B_i$ is realizable where each B_j in the product is from D_π. Let $B \in D_\pi^i$ be a potential block-move. If the out-degree of B is 0 in the dependency graph $G^d(D_\pi^i, \pi^i)$, then B is an actual block-move.*

Proof: Since B has out-degree 0 in $G^d(D_\pi^i, \pi^i)$, no arcs exists that have range within the span of B in π^i. B may either be a skip or a hop.

Suppose $B = B(a \sim b)$ is a skip. To obtain a contradiction, suppose $Blength(\pi^i, B) > 1$ so that there is at least one element x between a and b. Since $a > b$, either $a > x$ or $x > b$. In either case, there is an additional arc involving x within the span of $B(a \sim b)$, which is a contradiction. We conclude that $Blength(\pi^i, B) = 1$ and that B is actual.

Suppose B is a hop, particularly a right hop $B = B(a \sim b, c)$. We exclude the existence of an element between a and b as for skips. An element x between b and c exists only if $b < x < c$ (otherwise an arc in the span would result) but this makes B a disabled hop which cannot occur by Lemma 5. We conclude that $Blength(\pi^i, B) = 2$, and that B is actual. A symmetric argument applies for the case where $B = B(b, c \sim a)$ is a left hop. □

The next lemma states that arcs and potential block-moves retain their relationship as long as they both exist in the current permutation.

Lemma 5 *Let π be a 2-strip-increasing permutation, and let D_π be a corresponding set of potential block-moves. Furthermore, let (a, b) be an arc in G_π^p, and let B be a block-move in D_π. Finally, suppose the product $\pi^i = \pi \cdot B_1 \cdot B_2 \cdots B_i$ is realizable and none of the B_j's in this product is B or contains arc (a, b). Then, $range(\pi, (a, b))$ is in $span(\pi, B)$ if and only if $range(\pi^i, (a, b))$ is in $span(\pi^i, B)$.*

Proof: All we need to show is, as actual block-moves are applied to π, that an arc (a, b) inside the span of a given block-move B never gets out of the span and that an arc (a, b)

not inside the span of B never enters its span. Let (c, d) be the longer arc in B.

Consider the first case where (a, b) is inside the span of B. In the permutation π, c and a occur in the first increasing sequence, in that order, while b and d occur in the second increasing sequence, also in that order since the range of (a, b) is in the range of (c, d). This means $c > a$ and $b > d$. Suppose the arc (a, b) did get out of the span at some i where the permutation $= \pi^i$. This will have to be because a is moved to the left of c or b is moved to the right of d. This is a contradiction because such happens as a result of a non-correcting block-move but all of the B_j's applied to π are correcting block-moves.

Consider the second case where (a, b) is not inside the span of B. This means that either a is to the left of c in π or b is to the right of d in π. The first case means a and c are in the first increasing sequence so that $a < c$, already in their correct order. The second case means d and b are in the second increasing sequence so that $d < b$, likewise already in their correct order. In either case, entering the span would entail a non-correcting block-move, again a contradiction.

We conclude that $range(\pi, (a, b))$ is in $span(\pi, B)$ if and only if $range(\pi^i, (a, b))$ is in $span(\pi^i, B)$. $\square$

The above result implies the following corollary.

Corollary 1

$$G^d(D_\pi^{i+1}, \pi^{i+1}) = G^d(D_\pi^i, \pi^i) - B_{i+1},$$

where $B_{i+1} \in D_\pi^i$ is an actual block-move applied to π^i.

That is, the effect of applying a block-move B to the permutation affects the dependency graph only as far as B is concerned; in particular, the dependencies (edges) not concerning B are retained in the dependency graph and no new dependencies are introduced.

We now prove a final lemma.

Lemma 6 *A dependency graph derived from a 2-strip-increasing permutation has no cycles.*

Proof: Let π be a 2-strip-increasing permutation, and let $G^d(D_\pi, \pi)$ be a dependency graph of potential block-moves. Recall that given two block-moves B_i and B_j, the directed edge (B_i, B_j) exists in the graph if the range of an arc in B_j is contained in the span of B_i. Let $m = Blength(\pi, B_i)$. The arc of B_j that is inside the span is at most length $m-1$. The length of B_j may be at most one more, because the heads or tails of the arcs are just next to each other for potential hops in π. Hence, $Blength(\pi, B_j) \leq m$. Figure 12 illustrates the cases where B_i and B_j have the same length—notice that B_i may be a skip but B_j must be a hop. It can be verified that the cases shown in the figure are the only cases. Notice that if such a case ($Blength(\pi, B_i) = Blength(\pi, B_j)$) does happen, the block-move B_j inside the span is closer to one end of the permutation than B_i. In particular, for right hops, B_j is closer to the right end of the permutation and for left-hops, B_j is closer to the left end of the permutation (see Figure 12).

What we have obtained is some metric f on block-moves so that we could claim that, in $G^d(D_\pi, \pi)$, whenever (B_i, B_j) is an edge, then $f(B_i) > f(B_j)$. Now $f(B)$ is just $Blength(\pi, B)$ where ties are broken by determining the distance of B from the right end of π in the case of right hops and the distance of B from the left end of π in the case of left hops. This "distance from an end of the permutation" serves as a kind of secondary metric in case of ties.

With the metric f, $f(B_i) > f(B_j)$ whenever (B_i, B_j) is an edge. This immediately implies that the dependency graph has no cycles since a cycle would cause an edge to contradict this relationship. □

We wrap up this section with a correctness theorem for Algorithm Match-Sort.

Theorem 6 *Let π be a 2-strip-increasing permutation π, and let M be a maximum matching of the arc graph G^a_π of π. The length of the optimal sequence of short block-moves that sorts π is $M + u$ where u is the number of unmatched arcs in G^p_π. Algorithm* Match-Sort *computes such a sequence.*

Proof: Using a maximum matching of G^a_π maximizes correcting hops and therefore mini-

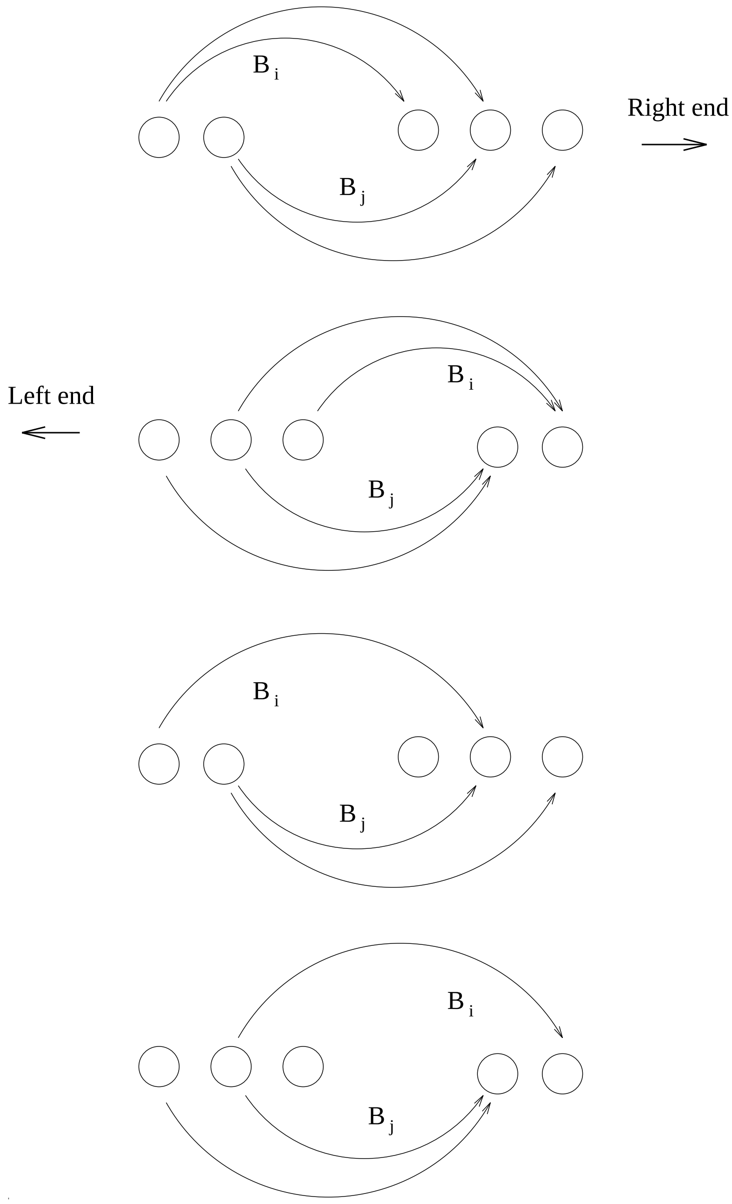

Figure 12: Cases where B_i and B_j have the same length.

mizes the total number of correcting short block-moves applied to the permutation π so this is clearly optimal (Lemma 3) assuming the block-moves do sort π. The question is whether the hops and skips derived are actual correcting hops and correcting skips that could be applied to π. The dependency graph contains all these short block-moves and Lemma 4 shows that as long as there is a vertex in the graph with out-degree 0, then the corresponding block-move can be applied to the current permutation. Algorithm Match-Sort repeatedly extracts such block-moves from the dependency graph (line 8). Lemma 5 guarantees that all that is needed to update the dependency graph is remove the just-applied block-move (line 10). Finally, we are certain that there is always a block-move with out-degree 0 in the dependency graph until the vertex set is exhausted (line 5) because Lemma 6 shows that there are no cycles in this graph. □

4.5 Short block-move diameter

Let S_n be the group of all permutations of length n. The short block-move *diameter* of S_n is the worst-case distance between two permutations in S_n. This can be restated as finding the short block-move distance of a permutation π that is farthest from the identity permutation.

A candidate for such a permutation is the decreasing sequence, for example, $\pi = 8\ 7\ 6\ 5\ 4\ 3\ 2\ 1$. Decreasing permutations can be considered "degenerate" bitonic permutations (in the example, 8 could be the increasing part, the rest of the elements, the decreasing part). Algorithm Bitonic-Sort therefore applies to the permutation and returns a result of $\lceil |A|/2 \rceil$, where $|A|$ is the number of arcs in the permutation graph. The permutation graph of a decreasing permutation has an arc for every pair of elements in π, for a total of $\binom{n}{2}$ arcs. Hence, the length of the optimal sorting sequence for a decreasing permutation is roughly half this value. More precisely, we have this lower bound.

Lemma 7 *The short block-move diameter for length-n permutations is at least:*

$$\left\lceil \binom{n}{2} / 2 \right\rceil .$$

It remains to devise an algorithm that sorts any permutation π with at most this number of short block-moves.

Consider the following algorithm. Given an unsorted permutation π, start with the two smallest elements e_1 and e_2 in π; that is, $e_1 \leftarrow 1, e_2 \leftarrow 2$. We have the following cases.

Case 1. e_1 has even degree. The element e_1 is clearly left-pending so that Lemma 2 applies and we perform hops to the left. After which we update the two elements: $e_1 \leftarrow e_2$, $e_2 \leftarrow e_2 + 1$.

Case 2. e_1 has odd degree. We have the following subcases:

- **Case 2a. e_1 is the last unpositioned element.** Move e_1 to the left using 1 skip and then left hops.
- **Case 2b. e_2 has even degree and is to the left of e_1.** Lemma 2 applies to e_2 (perform left hops). e_1 remains the same, $e_2 \leftarrow e_2 + 1$.
- **Case 2c. e_2 has odd degree and is to the left of e_1.** Move e_1 to the left using hops until it is to the left of e_2. Then, perform FixParity on e_1 and e_2. After this, Lemma 2 applies to both elements (perform left hops). $e_1 \leftarrow e_2 + 1$, $e_2 \leftarrow e_1 + 1$.
- **Case 2d. e_2 has even degree and is to the right of e_1.** Move e_1 to the left using 1 skip and then left hops. Move e_2 to the left using hops (Lemma 2). $e_1 \leftarrow e_2 + 1$, $e_2 \leftarrow e_1 + 1$.
- **Case 2e. e_2 has odd degree and is to the right of e_1.** Move e_1 to the left using 1 skip and then left hops. $e_1 \leftarrow e_2$, $e_2 \leftarrow e_2 + 1$.

The algorithm performs the above instructions iteratively, until the permutation is sorted.

Notice that in general, hops are used in the sort. Only in cases 2a, 2d and 2e are skips used. Case 2a occurs at most once; for cases 2d and 2e, e_2 is to the right of e_1, meaning e_1 and e_2 are already in their correct relative order in π (note that $e_1 < e_2$).

We use this fact to obtain an upper bound for the diameter.

Theorem 7 *The short block-move diameter for length-n permutations is exactly:*

$$\left\lceil \binom{n}{2} /2 \right\rceil .$$

Proof: Let s be the number of skips performed by the above algorithm on some permutation π. This means that there are at least $s-1$ arcs that are *not* in the permutation graph of π (elements already in their correct relative order are revealed per skip except perhaps the last one). Let $|A|$ denote the number of arcs in the permutation graph. Since there are at most $\binom{n}{2}$ arcs in any permutation graph,

$$|A| \leq \binom{n}{2} - (s-1)$$

Let h be the number of hops performed by the algorithm. Each skip corresponds to a single arc so that a total of $|A| - s$ arcs remain for hops. A hop corresponds to two arcs so that:

$$h = (|A| - s)/2 \leq \left(\binom{n}{2} - 2s + 1 \right) /2 = \left(\binom{n}{2} + 1 \right) /2 - s.$$

The short-block move diameter is at most the total number of hops and skips performed by the algorithm:

$$\begin{aligned} h + s &\leq \left(\binom{n}{2} + 1 \right) /2 - s + s \\ &= \left(\binom{n}{2} + 1 \right) /2 \\ &= \left\lceil \binom{n}{2} /2 \right\rceil . \end{aligned}$$

This value is also a lower bound for the short block-move diameter as shown in Lemma 7 so we conclude that this value is exactly the short block-move diameter for length-n permutations. □

5 Conclusions

Some bounded variants for minimum sorting by block-moves are solvable in polynomial time and have straightforward solutions such as $MinSort_{Bk^2}$ and $MinSort_{Bk^{1,n-1}}$. We have shown that when the bound given is not fixed but is dependent on the length of the permutation, the corresponding variant is at least as difficult as the unrestricted problem.

The natural bounded variant to investigate next is $MinSort_{Bk^3}$, minimum sorting by short block-moves, and although we have presented algorithms that solve this problem for special types of permutations, a general polynomial-time algorithm remains to be devised. We end this paper with the conjecture that such an algorithm exists.

Conjecture 1 *$MinSort_{Bk^3}$ is solvable in polynomial time.*

We suspect that the graph model presented here for $MinSort_{Bk^3}$ still applies to general permutations. However, a more sophisticated strategy that handles potential, disabled, and actual short block-moves is necessary.

References

[1] M. Aigner and D. B. West, *Sorting by insertion of leading element*, Journal of Combinatorial Theory A, 45 (1987), pp. 306–309.

[2] V. Bafna and P. A. Pevzner, *Sorting permutations by transpositions*, in Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, 1995, pp. 614–623.

[3] ——, *Genome rearrangements and sorting by reversals*, SIAM Journal on Computing, 25 (1996), pp. 272–289.

[4] A. Caprara, *Sorting by reversals is difficult.* Submitted to the SIAM Journal on Computing, 1996.

[5] S. Even and O. Goldreich, *The minimum-length generator sequence is NP-hard*, Journal of Algorithms, 2 (1981), pp. 311–313.

[6] S. Guyer, L. S. Heath, and J. P. C. Vergara, *Subsequence and run heuristics for the sorting by transpositions problem.* Accepted to the Fourth DIMACS International Algorithm Implementation Challenge, 1995.

[7] L. S. Heath and J. P. C. Vergara, *Some experiments on the sorting by reversals problem*, Tech. Rep. TR 95-16, Department of Computer Science, Virginia Polytechnic Institute and State University, 1995.

[8] M. R. Jerrum, *The complexity of finding minimum-length generator sequences*, Theoretical Computer Science, 36 (1985), pp. 265–289.

[9] D. B. Johnson, *A priority queue in which initialization and queue operations take $O(\log\log D)$ time*, Mathematical Systems Theory, 15 (1982), pp. 295–309.

[10] J. Kececioglu and D. Sankoff, *Exact and approximation algorithms for the inversion distance between two chromosomes*, in Fourth Annual Symposium on Combinatorial Pattern Matching, Lecture Notes in Computer Science, vol. 684, 1993, pp. 87–105.

[11] ——, *Efficient bounds for oriented chromosome inversion distance*, in Fifth Annual Symposium on Combinatorial Pattern Matching, Lecture Notes in Computer Science, 1994, pp. 307–325.

[12] A. Pneuli, A. Lempel, and S. Even, *Transitive orientation of graphs and identification of permutation graphs*, Canadian Journal of Math, 23 (1971), pp. 160–175.

[13] D. Rotem and J. Urrutia, *Permutation graphs*, Networks, 12 (1982), pp. 429–437.