

# A Constant Approximation Algorithm for Sorting Buffers

Jens S. Kohrt<sup>1\*</sup> and Kirk Pruhs<sup>2\*\*</sup>

<sup>1</sup> Department of Mathematics and Computer Science  
University of Southern Denmark, Odense, Denmark  
svalle@imada.sdu.dk

<sup>2</sup> Computer Science Department  
University of Pittsburgh  
kirk@cs.pitt.edu

**Abstract** We consider an algorithmic problem that arises in manufacturing applications. The input is a sequence of objects of various types. The scheduler is fed the objects in the sequence one by one, and is equipped with a finite buffer. The goal of the scheduler/sorter is to maximally reduce the number of type transitions. We give the first polynomial-time constant approximation algorithm for this problem. We prove several lemmas about the combinatorial structure of optimal solutions that may be useful in future research, and we show that the unified algorithm based on the local ratio lemma performs well for a slightly larger class of problems than was apparently previously known.

## 1 Introduction

We consider an algorithmic problem that arises in some manufacturing applications. The input is a sequence of objects of various types. The scheduler is fed the objects in the sequence one by one, and is equipped with a buffer that can hold up to  $k$  objects. When the scheduler receives a new object, the object is initially placed in the sorting buffer. Then the scheduler may eject any objects in the sorting buffer in arbitrary order. The sorting buffer may never hold more than  $k$  objects, and must end up empty. Thus the output from the sorting buffer is a permutation of the input objects. Informally, the goal of the scheduler is to minimize the number of transitions between objects of different type in the output sequence.

An example situation where this problem arises is the Daimler-Benz car plant in Sindelfingen, Germany [2]. Here the objects are cars, and the types are

---

\* Supported in part by the Danish Natural Science Research Council (SNF) and in part by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT). Most of this work was done while visiting the Computer Science Department at the University of Pittsburgh.

\*\* Supported in part by NSF grant CCR-0098752, NSF grant ANI-0123705, and NSF grant ANI-0325353, and by a grant from the US Air Force.

the final color that that particular car should be painted. Particular cars must be colored particular colors because of custom orders from customers/dealers. For example, a customer can go to <http://mbusa.com/> and order a G55 AMG Mercedes-Benz SUV with any number of possible options, including the exterior paint color. It is reported in [2] that the performance of the final layer painting yield mainly depends on the batch size of cars that have to be painted the same color, and as a consequence the performance of these sorting buffers can have a great impact on the overall performance of the manufacturing.

For concreteness we will adopt terminology appropriate for the Daimler-Benz car plant example, and consider the types of the objects to be colors. The most obvious objective function would be to minimize the number of transitions between objects of different color in the output sequence. The corresponding maximization objective function would be to maximize the number of color transitions removed from the sequence. While it may not be completely obvious, it is not too difficult to see that in an optimal solution there are no color changes introduced into the output sequence that were not in the input sequence. Hence, one can then see that a solution is optimal for the minimization problem if and only if it is optimal for the maximization problem. Of course, the equivalence of the maximization and minimization problems does not hold in the context of approximation. Whether a minimization or a maximization approximation algorithm is most appropriate depends on the input.

As an example of the problem, and the notation that we adopt, consider the following example. The initial sequence is  $r_1g_1r_2g_2r_3g_3b_1r_4b_2r_5b_3r_6$  that contains 12 objects with 11 color changes. The letters denote colors and the subscripts denote different objects of the same color. If  $k \geq 4$ , then an optimal output solution is  $g_1g_2g_3r_1r_2r_3r_4r_5r_6b_1b_2b_3$ . It can be achieved by storing  $r_1$ ,  $r_2$ , and  $r_3$  in the buffer until the buffer contains these red objects and  $b_1$ . Then  $r_1, r_2, r_3$  can be output, and the buffer can store the blue objects until the end. This gives a value of 2 color changes for the minimization objective function, and a value of 9 color savings for the maximization objection function.

It is not known if this sorting buffers problem is *NP*-hard. It is not hard to see that there is an  $O(n^{k+1})$ -time dynamic programming algorithm for this problem. It is also not hard to see that there is an  $O(n^{c+1})$ -time dynamic programming algorithm for this problem, where  $c$  is the number of different colors. So if  $k$  or  $c$  is  $O(1)$ , then the problem can be solved exactly in polynomial time. It seems that there may be real-world applications where the number of colors is not too large. The best approximation result known for the minimization approximation, an approximation ratio of  $O(\log^2 k)$ , is obtained by the polynomial-time online algorithm Bounded Waste introduced in [6].

A related problem, Paint Blocking, is studied in [7]. In this case the input sequence is reordered twice using two different buffers of the same size. After the first reordering the number of transitions between different types of objects is counted, and the second reordering has to return the sequence to its original order. For the minimization problem, a 5-approximation algorithm is given in [7].

Our main algorithmic result is a polynomial-time  $\frac{1}{20}$ -approximation algorithm for the maximization problem. Thus, this is the first constant approximation for either the maximization or minimization version of the sorting buffers problem. In order to obtain this result we have to prove several combinatorial lemmas about the structure of optimal, and near optimal, solutions. We expect that lemmas will be useful in further investigations into this problem. In the process of developing our algorithm, we showed that the analysis of the unified algorithm in [1] can be generalized to a slightly larger class of problems than indicated in [1]. Essentially our formulation allows arbitrary pairwise restrictions on membership in the solution, while in [1] only transitive restrictions are allowed. It is certainly plausible that this generalization might be useful elsewhere.

## 2 Observations about the Optimal Schedule

For any algorithm ALG and any input sequence  $\sigma$ , we let  $\text{ALG}(\sigma)$  denote both the resulting schedule and the color savings created by this schedule. Let  $\text{OPT}(\sigma)$  denote both an optimal schedule on input  $\sigma$  and the color savings in this solution. We say that the algorithm ALG is a  $c$ -approximation algorithm, if for any input sequence  $\sigma$ ,  $\text{ALG}(\sigma) \geq c \cdot \text{OPT}(\sigma)$ . A schedule is *lazy* if a color change is never created in the output sequence if there is another legal move that doesn't create a color change. As noted in [6], one can always change any algorithm into a lazy algorithm without any loss of performance. The following observations are then almost immediate consequences of this observation.

**Lemma 1.** *Consider an arbitrary input sequence. If two objects of the same color are adjacent in the input sequence, then there is an optimal schedule where these two objects are adjacent in the output sequence.*

**Lemma 2.** *For any optimal algorithm and any input sequence, we may assume that for any color, any two objects of this color have the same order in the input sequence and the output sequence.*

One consequence of Lemma 1 is that no color changes are created in the output sequence. Thus it makes sense to talk purely about the reduction of color savings in the optimal without needing to consider the possibility of color changes created in the optimal. This allows us to formally define our problem in a slightly non-intuitive manner, in which the input is a sequence of groups of objects with the same color. The fact that this problem statement is equivalent to the problem statement in the introduction essentially follows from the fact that we can restrict ourselves to lazy schedules.

**Definition 1.** *The Sorting Buffers Maximization Problem (SBMP) is defined as follows:*

- *The input sequence  $\sigma$  consists of a sequence of groups. Each group has a color and a size, i.e., the number of items it contains. No two consecutive groups have the same color.*

- The output sequence is a permutation of the groups in the input sequence.
- The buffer  $B$  can contain a collection of groups with size at most  $k - 1$ . Initially the buffer is empty. Note that the  $k - 1$  bound insures that there is one space in the buffer left to move objects through.
- Any algorithm for SBMP essentially repeats the following two steps until the entire input sequence has been read, and the buffer is empty.
  1. The next group in the input sequence is either appended to the output sequence or is put in the buffer (if space permits).
  2. Optionally one or more groups from the buffer are moved to the output sequence.
- When an algorithm  $ALG$  is run on an input sequence  $\sigma$  it gains a color savings of one each time the last group in the output sequence and the next group to be put there are the same color.

In SBMP, a lazy schedule is now defined as follows: if the last group in the output sequence is of a color  $c$ , then first all groups of objects of color  $c$  in the buffer are output in first-in-first-out order, and then if the next group of objects in the input sequence has color  $c$  then this group is immediately output without being put into the buffer. As before, one can always change any algorithm/schedule into a lazy algorithm/schedule without loss of performance, and thus, we only consider lazy schedules. We now switch notation and use  $r_i$  to denote the  $i$ th group with color  $r$ .

We classify color savings between two groups, say  $r_i$  and  $r_{i+1}$  in a schedule in one of the following three ways:

- *Move out-saving (MOS)*: In this case,  $r_i$  is placed in the output sequence before  $r_{i+1}$  is reached, and further all the groups, that are between  $r_i$  and  $r_{i+1}$  in the input sequence, appear after  $r_{i+1}$  in the output sequence. Hence, all the items between  $r_i$  and  $r_{i+1}$  in the input sequence were in the cache when  $r_{i+1}$  was output. The *out groups* for this MOS, are defined to be the groups between  $r_i$  and  $r_{i+1}$  in the input. An example of a MOS is (here the block label  $A$  is some arbitrary sequence of groups):

$$r_i \boxed{A} r_{i+1} \rightarrow r_i r_{i+1} \boxed{A}$$

Note that for a MOS it is not necessary that  $r_i$  be put in the cache, and thus we will assume that it is not in the optimal solution.

- *Move backward-saving (MBS)*: In this case  $r_i$  is put into the buffer, and is not expelled before group  $r_{i+1}$  is reached. An example of a MBS is:

$$r_1 \boxed{A} r_2 \rightarrow \boxed{A} r_1 r_2$$

- *Move backward and out-saving (MBOS)*: In this case,  $r_i$  placed in the output sequence before  $r_{i+1}$  is reached, and not all the groups, that are between  $r_i$  and  $r_{i+1}$  in the input sequence, appear after  $r_{i+1}$  in the output sequence. An example of a MBOS is:

$$r_i \boxed{A} \boxed{B} r_{i+1} \rightarrow \boxed{A} r_i r_{i+1} \boxed{B}$$

This is a combination of the previous two. At first, the  $r_i$  is put in the buffer, and *moved backward*. Before  $r_{i+1}$  is reached,  $r_i$  is dropped in the output sequence, and the groups in  $B$  are *moved out*.

At the time that  $r_i$  is placed in the output, let  $g_j$  be the next group in the input sequence. Then the *drop point* is defined to be the point immediately before  $g_j$  in the input sequence. The *out groups* for this MBOS are defined to be the groups between the drop point of  $r_i$  and  $r_{i+1}$  in the input.

In all three cases, we say that  $r_i$  is the *first group* of the savings and  $r_{i+1}$  is the *last group*. We now give an illustrative example of these definitions in the following figure:

$$r_1 \boxed{A} r_2 \boxed{B} r_3 \boxed{C} \boxed{D} r_4 \boxed{E} r_5 \boxed{F} r_6 \rightarrow \boxed{A} \boxed{B} \boxed{C} r_1 r_2 r_3 r_4 r_5 r_6 \boxed{D} \boxed{E} \boxed{F}$$

The first two color savings, which correspond to the pairs  $r_1 - r_2$  and  $r_2 - r_3$ , are MBS's. This is because  $r_1$  is moved backward to  $r_2$ , then both of them are moved backward to  $r_3$ . After this the  $r_1 - r_2 - r_3$  group is dropped in the output sequence. The groups in  $D$  are the out groups for the  $r_3 - r_4$  MBOS. Thus, the color savings corresponding to the  $r_3 - r_4$  pair is a MBOS. The drop point for  $r_3$  is between the last group in  $C$  and the first group in  $D$ . The last two color changes are MOS's.

**Lemma 3.** *Let  $r_i$  and  $r_{i+1}$  be any two groups between which there is a MOS, and let  $A$  be the groups between  $r_i$  and  $r_{i+1}$  in the input sequence. Then:*

- *No group in  $A$  are part of a MOS nor are they the last group in a MBOS.*
- *No group before  $r_{i+1}$  in the input sequence are the first group in a MBOS with drop point between  $r_i$  and  $r_{i+1}$ .*

*Proof.* Each of these possibilities involve placing a group in the output sequence between  $r_i$  and  $r_{i+1}$ , which contradicts  $r_i - r_{i+1}$  being a MOS. □

**Lemma 4.** *Let  $r_i$  and  $r_{i+1}$  be any two groups between which there is a MBOS. Let  $A$  be the groups between  $r_i$  and the drop point for  $r_i$  in the input sequence. Let  $B$  be the groups between the drop point for  $r_i$  and  $r_{i+1}$  in the input sequence. Then:*

- *No group in  $B$  are part of a MOS, nor are they the last group in a MBOS.*
- *No group before  $r_{i+1}$  in the input sequence are the first group in a MBOS with drop point between the drop point of  $r_i$  and  $r_{i+1}$ .*

*Proof.* Similar to Lemma 3. □

**Lemma 5.** *For any two distinct MOS's or MBOS's, the out groups do not overlap.*

*Proof.* For ease of reference, we denote the point in the input sequence succeeding the first group of a MOS the drop point of this savings. Let  $r_i - r_{i+1}$  and  $b_j - b_{j+1}$  be any two distinct pairs of groups between which there are a MOS or MBOS. Without loss of generality we assume that the drop point of  $r_i - r_{i+1}$  occurs before the drop point of  $b_j - b_{j+1}$ . Then, by Lemmas 3 and 4 the drop point of  $b_j - b_{j+1}$  is at the earliest after  $r_{i+1}$  in the input sequence. Thus, the out groups of the two savings do not overlap.

### 3 Reduction to Two Problems

We show that either an  $\Omega(1)$  fraction of the savings in the optimal solution are MOS, or an  $\Omega(1)$  fraction of the savings in the optimal solution are MBS. We then show how to efficiently construct a solution that is constant competitive with respect to the number of MOS color savings, and show how to efficiently construct a solution that is constant competitive with respect to the number of MBS color savings. By taking the better of these two solutions, we get a constant approximation algorithm.

We first note that we can disregard the MBOS.

**Lemma 6.** *For any input sequence  $\sigma$ , there exists a solution for which the total number of MBS's and MOS's equals at least half of the profit of an optimal solution.*

*Proof.* Let  $\sigma$  be any fixed input sequence, and let  $\text{OPT}(\sigma)$  be any fixed optimal schedule for  $\sigma$ . We gradually transform this schedule into a new schedule with the desired properties.

We consider all MBOS of  $\text{OPT}(\sigma)$  one by one in order of their first group starting from the end of input sequence and continuing to the beginning (in the opposite of the usual order). During this sweep we maintain the invariant that any group further towards the end of the input sequence which is the first group in a MBOS in the original optimal schedule either has been turned into the first group of a MBS or it has a unique associated MBS in the schedule. Furthermore, we do not change any MBS or MOS in the original schedule. As shown below we also ensure that no two MBOS's share the same associated MBS. Consequently the resulting schedule has at least half the profit of the optimal solution.

Let  $r_i - r_{i+1}$  be a MBOS under consideration. Let  $A$  be the groups in  $\sigma$  between  $r_i$  and  $r_i$ 's drop point, and  $B$  the groups in  $\sigma$  between  $r_i$ 's drop point and  $r_{i+1}$ . That is this part of the input looks like,  $\dots r_i A B r_{i+1} \dots$ . Note that by Lemma 4 and 5, no group in  $B$  participates in a MOS, neither are they the last group of a MBOS.

First, if any group in  $B$  is the first group of a MBS, then we associate one of the corresponding MBS's with the  $r_i - r_{i+1}$  MBOS. Note, as a direct result of Lemma 5 no other MBOS is associated with this MBS.

If this is not the case, we instead transform the MBOS into a MBS. The transformation is by induction on the number of groups left in  $B$ . The base case is when no groups in  $B$  are left. In this case we have turned this MBOS into a MBS, since  $r_i$  is kept in the buffer until  $r_{i+1}$  is reached in the input sequence. For the induction step, let  $g_j$  be the group in  $B$  furthest to the beginning of the input sequence. If  $g_j$  is not the first group in a savings, then the same profit is gained, if  $g_j$  is before  $r_i$  in the output sequence. Consequently instead of placing  $r_i$  in the output sequence, just before  $g_j$  is met in the input sequence, we keep  $r_i$  in the buffer, output  $g_j$  directly, and only then we place  $r_i$  in the output sequence. Otherwise, if  $g_j$  is the first group of a savings, it must be a MBOS. In this case, we place  $g_j$  into the output sequence as soon as  $g_j$  is encountered, and

then immediately afterwards place  $r_i$  in the output buffer. This may reduce the total profit gained by the solution by one, as the  $g_j - g_{j+1}$  MBOS is deleted. But due to the invariant, this savings already has an associated MBS in the solution which pays for the deletion of this MBOS.  $\square$

**Definition 2 (SBMP-R).** *Let the Reduced Sorting Buffers Maximization Problem (SBMP-R) be defined as the Sorting Buffers Maximization Problem (SBMP) (Definition 1), except that no group can participate in more than one color savings, and that profit is only gained for savings of type MBS or MOS.*

Note that in SBMP each group can participate in up to two color savings, one in front and one in back. As an example of this, look at the following input and output sequence:

$$r_1 \boxed{A} r_2 \boxed{B} r_3 \boxed{C} r_4 \rightarrow \boxed{A} \boxed{B} \boxed{C} r_1 r_2 r_3 r_4$$

A total of four groups are involved in the color savings. For SBMP, a total color savings of three is gained, whereas for SBMP-R, only a total colors savings of two is gained (the two blocks are  $r_1 - r_2$  and  $r_3 - r_4$ ). For SBMP-R, another solution gives rise to the same profit:

$$r_1 \boxed{A} r_2 \boxed{B} r_3 \boxed{C} r_4 \rightarrow \boxed{A} r_1 r_2 \boxed{B} \boxed{C} r_3 r_4$$

By only looking at SBMP-R instead of SBMP, we loose an approximation factor of  $\frac{1}{4}$ .

**Lemma 7.** *Let  $\sigma$  be any input sequence for SBMP and SBMP-R. Let  $OPT(\sigma)$  be an optimal solution for SBMP, and let  $OPT_R(\sigma)$  be an optimal algorithm for SBMP-R. Then  $OPT_R(\sigma) \geq \frac{1}{4} OPT(\sigma)$ .*

*Proof.* By Lemma 6 there exists a schedule for which the total number of MOS's and MBS's is at least half of  $OPT(\sigma)$ . Let  $S$  be any such schedule, and divide the resulting output sequence of  $S$  into maximal runs of groups of the same color between which there is a MOS or MBS. So runs are either broken by a color change or a MBOS. For any such run with  $i \geq 2$  groups,  $S$  has a profit of  $i - 1$ . This run can be divided into  $\lfloor i/2 \rfloor$  disjoint pairs of color savings. Thus, for  $OPT_R$  the profit is at least  $\lfloor i/2 \rfloor \geq \frac{i-1}{2}$ , that is, at least one half of the profit gained by  $S$  on MOS's and MBS's. By Lemma 6 this is at least  $OPT(\sigma)/4$ .  $\square$

Let SBMP-R-MOS be the problem of maximizing the number of color savings of type MOS where each group participates in at most one savings. Similarly, let SBMP-R-MBS be the problem of maximizing the number of color savings of type MBS where each group participates in at most one savings. In section 4, we give a polynomial time algorithm solving SBMP-R-MOS. In section 5, we give a polynomial time algorithm with an approximation factor of at least  $\frac{1}{4}$  for SBMP-R-MBS. Either the optimal solution for SBMP-R-MOS is a  $\frac{1}{5}$  approximation of the optimal solution for SBMP-R, or the optimal solution for SBMP-R-MBS is a  $\frac{4}{5}$  approximation of the optimal solution for SBMP-R. Hence, the better of our solutions in section 4 and section 5 is a  $\frac{1}{5}$  approximation of  $OPT_R$ . Consequently we have an  $\frac{1}{20}$  approximation of SBMP.

**Theorem 1.** *There is an algorithm with an approximation factor of at least  $\frac{1}{20}$  for SBMP running in polynomial time.*

## 4 A SBMP-R-MOS Algorithm

The next piece of the puzzle is a greedy algorithm solving SBMP-R-MOS exactly. Recall this problem is to maximize the number of MOS.

As before, we may assume that the ordering for groups of the same color does not change. Consequently it is sufficient to consider only MOS between two groups  $r_i$  and  $r_{i+1}$  of the same color, where no group of this color occurs between the two in the input sequence. Further, the total size of the groups in between  $r_i$  and  $r_{i+1}$  has to be at most  $k - 1$ , or else they cannot be in the buffer. Again one can show, as in Lemma 3, that we may assume that no pair of MOS occur inside one another.

Instead of solving SBMP-R-MOS directly, we transform it into the problem of finding a maximum independent set in an interval graph. That is, the input is a sequence of intervals over the real line, and the desired output is a maximal cardinality disjoint set of intervals. If one orders the intervals by increasing right endpoint, and greedily selects intervals, it is a standard exercise to show that this greedy algorithm exactly solves the problem [3,4].

We now construct our reduction. For each possible MOS in the input sequence, we create a corresponding interval starting at the first group and ending at the last group of the MOS (inclusive). Then two intervals overlap, if and only if they cannot occur together. This is the case if they either share a group or if they would occur inside each other, if they were both used. Then the maximum MOS corresponds exactly to the maximal independent set in the resulting interval graph.

## 5 A SBMP-R-MBS Approximation Algorithm

As part of our approximation algorithm for SBMP-R-MBS we need a generalization of the unified algorithm introduced in [1]. First, we explain the generalization, and then we apply this result on our own problem by using a reduction from SBMP-R-MBS to The Resource Allocation Maximization Problem.

**Definition 3 (RAMP).** *The Resource Allocation Maximization Problem is defined as follows:*

- *The input consists of a number of instances  $I$ , each requiring the utilization of some limited resource. The amount of resource available is fixed over time, and its size is normalized to one.*
- *Instances  $I$  are each defined by the following four properties:*
  - *A half-open time interval  $[s(I), e(I))$  during which the instance is to be executed.  $s(I)$  and  $e(I)$  are the start-time and end-time of the instance.*
  - *The amount of resource necessary or the width of the instance,  $w(I)$ ,  $0 \leq w(I) \leq 1$ .*

- The profit  $p(I) \geq 0$  gained by using this instance
  - The activity  $A(I)$  which is a set of instances (always including  $I$ ) which cannot occur at the same time as  $I$ . For any two instances  $I$  and  $J$ , then  $I \in A(J)$  if and only if  $J \in A(I)$ .
- The output or a feasible schedule is a set of instances  $X$ , such that for each instance  $I$  in  $X$  the set  $A(I) \cap X$  only contains  $I$ , and for each time instance  $t$  the total width of all instances in  $X$  which contains  $t$  is at most one. The total profit of  $X$  is the sum of the profit of the individual instances in  $X$ .
- The goal of RAMP is to find a feasible schedule with maximal profit.

Note that the only way our problem differs from the original problem [1] is the way activities are defined. In our setting each instance  $I$  has its own set of other instances  $A(I)$  which cannot occur at the same time as  $I$ . As an example, in our problem it is possible to have three instances,  $I_1$ ,  $I_2$ , and  $I_3$ , such that  $I_1 \in A(I_2)$  and  $I_2 \in A(I_3)$ , but  $I_1 \notin A(I_3)$ . In [1] this cannot be the case, since activities are transitive, i.e.,  $I_1 \in A(I_2)$  and  $I_2 \in A(I_3)$  implies  $I_1 \in A(I_3)$ .

By following the analysis in [1], one can verify that it still holds in this more general setting. The reader is referred to [5] for a more detailed analysis. The main result about the obtained approximation ratio is the same as in [1]:

**Lemma 8 (Lemma 3.2 from [1]).** *Let  $w_{\min}$  ( $w_{\max}$ ) be the minimum (maximum) width of any instance in the input, and let  $\alpha$  be some value larger than zero. The approximation factor of the unified algorithm is at least*

$$\frac{\min \{1, \alpha \cdot \max \{w_{\min}, 1 - w_{\max}\}\}}{1 + \alpha}$$

As noted in [1] the unified algorithm can easily be implemented in polynomial time. This result also applies for our slightly modified unified algorithm.

We now use this result to make an approximation algorithm for our original problem, SBMP-R-MBS, by reducing it to the generalized problem.

First note, that we may not, as in SBMP, assume that the ordering for groups of the same color does not change. This can be seen in the following example:

$$r_1 \boxed{A} r_2 \boxed{B} r_3 b_1 \boxed{C} b_2 r_4 \rightarrow \boxed{A} \boxed{B} r_2 r_3 \boxed{C} b_1 b_2 r_1 r_4$$

where  $r_1$  contains one item, and all other groups each contain  $k - 2$  items. If we want to move  $r_3$  to  $r_4$ , then we cannot move  $b_1$  to  $b_2$ . Thus, the only feasible solution with a profit of three is the one given above.

Further note, that as only MBS's are allowed, we may assume that the first group of a MBS is moved no farther backward than to the last group: moving the two groups farther back in the sequence does not give rise to more profit.

Also note that we can see the input sequence as a time line. Then a MBS starts/ends at the time corresponding to its first/last group (inclusive).

With the above in mind, we now construct the reduction. For each color  $r$  and for each pair of groups of this color  $r_i$  and  $r_j$  where  $r_i$  occurs before  $r_j$  in the input sequence and where the size of  $r_i$  is at most  $k - 1$ , we create an instance

$I_{r_i r_j}$  corresponding to the possible  $r_i - r_j$  MBS. The width of  $I_{r_i r_j}$  is the number of items in the first group normalized by  $k - 1$ , i.e.,  $w(I_{r_i r_j}) = \text{size}(r_i)/(k - 1)$ . As all MBS's have a profit of one,  $p(I_{r_i r_j}) = 1$ . Further, the activity  $A(I_{r_i r_j})$  contains  $I_{r_i r_j}$  as well as those instances that use either  $r_i$  or  $r_j$ , i.e., exactly the instances which cannot occur at the same time as  $I_{r_i r_j}$ .

**Lemma 9.** *There is an algorithm running in polynomial time with an approximation factor of at least  $\frac{1}{4}$  for SBMP-R-MBS.*

*Proof.* Suppose all instances have a width of at most  $\frac{1}{2}$ , i.e.,  $w_{\max} \leq \frac{1}{2}$ . In this case  $\alpha = 2$  maximizes Lemma 8 with a performance guarantee of  $\frac{1}{3}$ .

Next, suppose all instances have a width of at least  $\frac{1}{2}$ , i.e.,  $w_{\min} > \frac{1}{2}$ . In this case no pair of intersecting instances may both be used. Consequently this problem is equivalent to the problem of finding the maximal independent set in an interval graph. Similar to Section 4, this can be solved exactly by a simple greedy algorithm.

In the general case we solve the problem separately for the instances of width at most a half and for the instances of width more than a half. Either the optimal solution for the former case is at least  $\frac{3}{4}$  of the optimum, or the optimal solution for the latter case is at least  $\frac{1}{4}$  of the optimum. Hence, the better of the two is at least a  $\frac{1}{4}$  approximation of SBMP-R-MBS.  $\square$

## References

1. Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph (Seffi) Naor, and Baruch Schieber. A unified approach to approximating resource allocation and scheduling. *Journal of the ACM*, 48(5):1069–1090, 2001.
2. Sven Brückner, Jo Wyns, Patrick Peeters, and Martin Kollingbaum. Designing agent for manufacturing control. In *Proceedings of the 2nd AI & Manufacturing Research Planning Workshop*, pages 40–46, 1998.
3. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, chapter 16. MIT Press and McGraw-Hill Book Company, 2nd edition, 2001.
4. U. I. Gupta, D. T. Lee, and Joseph Y.-T. Leung. An Optimal Solution for the Channel-Assignment Problem. *IEEE Transactions on Computers*, 28(11):807–810, 1979.
5. Jens S. Kohrt and Kirk Pruhs. A constant approximation algorithm for sorting buffers. Technical Report PP-2003-21, Department of Mathematics and Computer Science, University of Southern Denmark, Odense, 2003.
6. Harald Räcke, Christian Sohler, and Matthias Westermann. Online scheduling for sorting buffers. In Rolf H. Möhring and Rajeev Raman, editors, *Proceedings of 10th Annual European Symposium (ESA)*, volume 2461 of *Lecture Notes in Computer Science*, page 820 ff. Springer, September 2002.
7. Joel Scott Sokol. *Optimizing Paint Blocking in an Automobile Assembly Line: An Application of Specialized TSP's*. PhD thesis, Department of Electrical Engineering and Computer Science, MIT, June 1999.