Alcom-FT Technical Report Series
ALCOMFT-TR-03-160

A Constant Approximation Algorithm for Sorting Buffers

Jens S. Kohrt[1⋆] and Kirk Pruhs[2⋆⋆]

[1] Department of Mathematics and Computer Science
University of Southern Denmark, Odense, Denmark
svalle@imada.sdu.dk
[2] 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 [3]. Here the objects are cars, and the types are the

⋆ 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 and by a grant from the US Air Force.

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 of the customers choice. It is reported in [3] 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.

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. Here the input is processed from left to right. 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.

Of course, the equivalence of the maximization and minimization problems does not hold in the context of approximation. To see this, consider two examples. In the first example, we have 10,000 objects, with 1,000 color changes in original input, and 500 color changes in the optimal output, that is, 500 color changes are eliminated. Then a 2-approximation algorithm for the minimization problem would need to guarantee at most 1,000 color changes, and in fact, need not even reorder this sequence at all. In contrast, a $\frac{1}{2}$-approximation algorithm for the maximization problem would require that at least 250 color changes are eliminated, or equivalently that the output sequence has at most 750 color changes. In the second example, we have 10,000 objects, with 1000 color changes in original input, and 100 color changes in the optimal output, that is, 900 color changes are eliminated. Then a $\frac{1}{2}$-approximation algorithm for the maximization problem would require that at least 450 color changes are eliminated, or equivalently that the output sequence has at most 550 color changes. In contrast, a 2-approximation algorithm for the minimization problem would require that at most 200 color changes remain in the output sequence. In the first example, one would prefer a $\frac{1}{2}$-approximation algorithm for the maximization problem, and in the second example, one would prefer a 2-approximation algorithm for the minimization problem. So both the minimization and maximization problems are reasonable and are worth studying, and which is most appropriate depends on the input.

It is not known if this sorting buffers problem is *NP*-hard, or is polynomial time solvable. 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 Bounded Waste algorithm introduced in [6]. Bounded Waste can be implemented in polynomial-time, and can also be implemented by an online scheduler that can not see the future sequence.

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 [2] can be generalized to a slightly larger class of problems than indicated in [2]. Essentially our formulation allows arbitrary pairwise restrictions on membership in the solution, while in [2] 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 σ, we let $\mathrm{ALG}(\sigma)$ denote both the resulting schedule and the color savings created by this schedule. Let $\mathrm{OPT}(\sigma)$ denote both the optimal schedule on input σ and the color savings in this solution. We say that the algorithm ALG is a c-approximation algorithm, if for any input sequence σ, $\mathrm{ALG}(\sigma) \geq c \cdot \mathrm{OPT}(\sigma)$. When σ is obvious, we drop it from the notation. 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 OPT and any input sequence σ, 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 (SBMP). *The Sorting Buffers Maximization Problem (SBMP) is defined as follows:*

- *The* input sequence σ *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 σ 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 ith 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):

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

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:

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

– *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:

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

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 color savings, and*
– *No group in A are the last group in a MBOS color savings, and*

– *No group before r_{i+1} in the input sequence are the first group in a MBOS color savings with drop point between r_i and r_{i+1}.*

Proof. Each of the three 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 color savings, and*
– *No group in B are the last group in a MBOS color savings, and*
– *No group before r_{i+1} in the input sequence are the first group in a MBOS color savings 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 color saving.

Lemma 6. *For any input sequence σ, 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 σ be any fixed input sequence, and let $\mathrm{OPT}(\sigma)$ be any fixed optimal schedule for σ. We gradually transform this schedule into a new schedule with the desired properties.

We consider all MBOS of OPT(σ) 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 σ between r_i and r_i's drop point, and B the groups in σ between r_i's drop point and r_{i+1}. That is this part of the input looks like, $\cdots r_i \boxed{\quad A \quad}\boxed{\quad B \quad} r_{i+1} \cdots$. 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. □

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:

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

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:

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

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

Lemma 7. *Let σ 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(σ). 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(σ)/4. □

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 with an approximation factor of at least $\frac{1}{4}$ for SBMP-R-MBS. In section 5, we give a polynomial time algorithm solving SBMP-R-MOS. Either the optimal solution for SBMP-R-MBS is a $\frac{4}{5}$ approximation of the optimal solution for SBMP-R, or the optimal solution for SBMP-R-MOS is a $\frac{1}{5}$ 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-MBS Approximation Algorithm

As part of our approximation algorithm for SBMP-R-MBS we need a generalization of the unified algorithm introduced in [2]. 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, $w(I)$, $0 \le w(I) \le 1$.*
  - *The* profit *$p(I) \ge 0$ 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 [2] 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 [2] 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 [2], one can verify that it still holds in this more general setting. For completeness, we essentially now repeat the basics of the analysis in [2].

The result is based on the local ratio lemma which according to [2] initially was introduced by Bafna et al [1]. A straightforward proof can be found in [2].

Lemma 8 (Local ratio). *Let F be a set of constraints and let p, p_1, and p_2 be profit vectors such that $p = p_1 + p_2$. Then, if x is an r-approximation with respect to (F, p_1) and with respect to (F, p_2), then it is an r-approximation with respect to (F, p).*

As part of the algorithm we need to generalize the problem even further so that negative profits are allowed. Moreover, we introduce a parameter $\alpha > 0$, where the value of α depends on the problem at hand. For any instance I, we also introduce the set $\mathcal{I}(I)$ to denote the set of instances intersecting I but which are not in $A(I)$.

Algorithm 1 (The unified algorithm) *The algorithm is first given a set of all instances with the original profit. It then solves the problem recursively.*

1. *Delete all instances with non-positive profit*
2. *If no instances remain, return the empty schedule. Otherwise, proceed to the next step.*
3. *Select an instance $\tilde{I}$ with minimum end-time. Decompose p by $p = p_1 + p_2$. To do this it is sufficient to define p_1:*

$$p_1(I) = p(\tilde{I}) \cdot \begin{cases} 1 & , \quad I \in A(\tilde{I}) \\ \alpha \cdot w(I) & , \quad I \in \mathcal{I}(\tilde{I}) \\ 0 & , \quad \textit{otherwise} \end{cases}$$

4. *Solve the problem recursively using p_2 as the profit function. Let S' be the schedule returned.*
5. *If $S' \cup \{\tilde{I}\}$ is a feasible schedule, return $S = S' \cup \{\tilde{I}\}$. Otherwise return $S = S'$.*

Note that before the recursive step $p_2(\tilde{I})$ is zero. Thus, at the next step $\tilde{I}$ is deleted, and the algorithm eventually comes to a halt.

As noted in [2] the unified algorithm can easily be implemented in polynomial time. This result also applies for our slightly modified unified algorithm.

Definition 4. *For any feasible schedule S and for any instance I, we say that S is I-maximal, if either S contains I or adding I to S would make S infeasible.*

Lemma 9. *Let r be a constant. Suppose that the method for choosing $\tilde{I}$ and decomposing the profit function is such that $p_2(\tilde{I}) = 0$, and for all profit functions p, every $\tilde{I}$-maximal schedule is a r-approximation with respect to p_1. Then the schedule S returned by the algorithm is an r-approximation.*

Proof. This can be shown by induction on the number of instances considered using the Local Ratio Lemma 8 and the fact the S is $\tilde{I}$-maximal. We refer the reader to [2]. □

The actual performance guarantee r we get depends on α. By lemma 9 we only need to show that the way we choose $\tilde{I}$ and decompose the profit function leads to an r-approximation with respect to p_1.

We derive an upper bound b_{opt} on the profit gained by an optimal algorithm, and a lower bound b_{max} on the profit gained by the unified algorithm both with respect to p_1. The ratio of the two $r = b_{\text{max}}/b_{\text{opt}}$ is a lower bound on the approximation factor of our algorithm.

First note, that in both cases only instances in $A(\tilde{I}) \cup \mathcal{I}(\tilde{I})$ contribute to profit by the definition of p_1. For the optimal algorithm, the contribution of the instances in $A(\tilde{I})$ is at most $p(\tilde{I})$, as at most one of them is in a feasible schedule. Since we choose instances in order of end-time, all instances in $\mathcal{I}(\tilde{I})$ must intersect the end-point of $\tilde{I}$. As the total width of these instances are at most one, they contribute at most $p(\tilde{I})\alpha$. In total, $b_{\text{opt}} = (1+\alpha)p(\tilde{I})$.

For our algorithm, note that the schedule either contains an instance in $A(\tilde{I})$ or contains a nonempty set $X \subseteq \mathcal{I}(\tilde{I})$ of instances that prevent $\tilde{I}$ from being added to the schedule. In the former case the profit is at least $p(\tilde{I})$. In the latter case the profit can be bounded from below as follows. Denote by w_{min} and w_{max} the minimum respective maximum width of any instance in the input. Then the total width of instances in X is at least $1 - w(\tilde{I}) \geq 1 - w_{\text{max}}$ as else $\tilde{I}$ could be added without problems. Further, as X is a nonempty set, the total width is at least w_{min}. Referring to the definition of p_1, we do in total have a lower bound on the profit gained of $b_{\text{max}} = \min\{1, \alpha \max\{w_{\text{min}}, 1 - w_{\text{max}}\}\}p_1(\tilde{I})$.

Lemma 10 (Lemma 3.2 from [2]). *The approximation factor of the unified algorithm is at least $b_{\text{max}}/b_{\text{opt}}$ for all provable bounds b_{max} and b_{opt}. These can* *always be chosen such that the ratio is at least*

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

We now use the above results 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 11. *There is an algorithm 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 10 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 we might as well set the width of all instances to one, and Lemma 10 is now maximized with $\alpha = 1$ giving a performance guarantee of $\frac{1}{2}$. However note that in this special case all instances in $A(\tilde{I})$ also intersect $\tilde{I}$, thus, at most one instance in $A(\tilde{I}) \cup \mathcal{I}(\tilde{I})$ can be used at one time. The profit of the optimal algorithm is then bounded by $b_{\text{opt}} = 1 \cdot p_1(\tilde{I})$, and in total we get a ratio of $b_{\max}/b_{\text{opt}} = 1$.

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. $\square$

5 A SBMP-R-MOS Algorithm

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

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 [4,5].

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.

References

1. Vineet Bafna, Piotr Berman, and Toshihiro Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM Journal on Discrete Mathematics*, 12(3):289–297, 1999.
2. 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 (JACM)*, 48(5):1069–1090, 2001.
3. 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.
4. 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.
5. 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.
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.