Exploiting Locality: Approximating Sorting Buffers

Reuven Bar-Yehuda and Jonathan Laserson

Computer Science Department, Technion, Haifa 32000, Israel
`{reuven, joni}@cs.technion.ac.il`

Abstract. The Sorting Buffers problem is motivated by many applications in manufacturing processes and computer science, among them car-painting and file servers architecture. The input is a sequence of items of various types. All the items must be processed, one by one, by a service station. We are given a random-access sorting buffer with a limited capacity. Whenever a new item arrives it may be moved directly to the service station or stored in the buffer. Also, at any time items can be removed from the buffer and assigned to the service station. Our goal is to give the service station a sequence of items with minimum type transitions. We generalize the problem to allow items with different sizes and type transitions with different costs. We give a polynomial-time 9-approximation algorithm for the maximization variant of this problem, which improves the best previously known 20-approximation algorithm.

1 Introduction

In the sorting buffers problem, the input is a sequence of items of various types. All the items must be processed, one at a time, by a service station. When the service station processes two consecutive items of different types we say that there is a type transition. Type transitions are expensive, and the goal is to give the service station a sequence of items with as few type transitions as possible. To achieve this task we are given a random-access sorting buffer with a limited capacity. Whenever a new item arrives it may be moved directly to the service station or stored in the sorting buffer. Also, at any time items can be removed from the sorting buffer and then assigned to the service station. Thus, the service station processes a sequence of items which is a permutation of the input sequence. Using the sorting buffer, we need to rearrange the input sequence so that the number of type transitions is minimized, or equivalently (for the maximization variant), so that the number of items which are followed by an item of the same type is maximized.

The sorting buffers problem is motivated by many applications in manufacturing processes. For example, during the manufacturing process in a car plant (e.g. the Daimler-Benz car plant in Germany), the cars arrive one after the other, from an assembly-line, to the painting center where each car is painted with its own top coat. If two consecutive cars are to be painted in different colors, a color change is required. Since each such color change causes a waste of paint and requires cleaning chemicals, it makes sense to rearrange the sequence of cars in a way that cars of the same color preferably appear in consecutive positions. For this purpose, a small garage with a limited capacity is built before the painting center, such that cars can be transferred from the assembly line to the garage, and later from the garage to the painting center. The garage acts as a sorting buffer and is used to deliver larger subsequences of cars of the same color.

This problem has also many application in computer science. For example, a file server receives a sequence of read/write requests to files stored on its disk. In addition to the time it takes to read or write the data to a file, more time is wasted by locating the file, opening it and closing it after the request is handled. One can minimize this overhead time by using a sorting buffer to group requests for the same file together and have them handled in sequence. In a similar way, this technique can be implemented in communication networks to group requests which deal with the same server and save the startup cost.

Another application is in computer graphics. During the process of polygon rendering, a set of polygons is processed one by one. A change of attributes in two consecutive polygons is denoted as state-change. As the number of state-changes decreases, the performance improves. By rearranging the sequence of polygons such that polygons with similar attributes are processed consecutively, one can effectively boost performance. In this case also, a sorting buffer can come in handy.

1.1 Our Contribution

We present a polynomial time 9-approximation algorithm for the maximization variant of the sorting buffers problem. This result improves the best previously known 20-approximation algorithm, obtained in [1]. The algorithm we introduce is also applicable to a generalized variant of the problem, in which each item is assigned a size and a nonnegative profit. We gain the profit assigned to an item if at the service station it is followed by another item of the same type (see formal definition in Problem 3). The goal is to gain maximum profit. The generalized problem becomes the original maximization problem if all the profits are equal.

We prove some combinatorial lemmas about the optimal solutions for this problem, and use the Local-Ratio Technique [3] [4] to obtain a polynomial-time 9-approximation algorithm for the generalized problem. This result can be easily converted to a simple solution in the primal-dual schema [5].

1.2 Previous Work

The first constant-approximation algorithm for the sorting buffers problem was given by Kohrt and Pruhs [1]. They gave a 20-approximation algorithm for the maximization variant of the problem. Their algorithm also uses the local-ratio technique. Kohrt et al. also noted that the problem can be solved exactly in polynomial time if either the number of types or the buffer size is constant.

The best approximation result known for the minimization problem is actually an on-line algorithm with a competitive ratio of $O(\log^2 k)$, where k is the size of the buffer. Räcke et al. [2] gave a deterministic bounded-waste strategy which achieved this result.

A related problem is studied by Epping and Hochstättler in [7]. In this problem, r queues are used to rearrange the items instead of a random-access sorting-buffer. Epping et al. show equivalence between their problem and the multiple sequence alignment problem known from molecular biology. They provide a dynamic programming algorithm which solves their problem exactly.

Another related problem is the bandwidth-allocation problem, which is studied in [6]. The input is a set of intervals, each with a width and a profit. The goal is to choose a subset of these intervals with maximum total profit such that at any point t, the total width of the intervals intersecting t is not larger than 1. Bar-Noy et al. were able to achieve a 5-approximation algorithm for this NP-hard problem. We will show later that the generalized maximization problem for sorting buffers is also a generalization of the bandwidth-allocation problem, and hence the generalized maximization problem is also NP-hard.

2 Preliminaries

The rest of this paper is organized as follows. In Section 2 we give a formal description of the problem, and make some observations on optimal solutions. These observations allow us to represent the problem differently, as a maximization problem. We also make some observations on a subclass of feasible solutions denoted as "good" and show how to turn any feasible solution to a good one. In Section 3 we generalize the problem by adding a profit function, and introduce the local-ratio schema which will be used on the generalized problem. In Section 4 we provide the rest of the details necessary for applying the schema, and obtain our approximation algorithm.

2.1 The Model

The input is a sequence of items $\sigma = \sigma_1, \sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_n$ which are only characterized by a specific attribute. To simplify things, we will assume that the items are packages, and that they are characterized by color. The input sequence is processed from left to right by a *sorting buffer* which is a random access buffer with storage capacity for k packages. During this process, packages may be stored in the buffer and later they are placed back into the sequence. The resulting sequence is the *output sequence* (this is the sequence given to the service station).

We can formalize the rearrangement process as follows. The process consists of n steps, where at step i ($i = 1, 2, \ldots, n$) at most one of these actions occur:

1. Any subset of the packages currently in the sorting buffer may be removed from the buffer and placed back in the sequence (right after σ_i), in any order.
2. If space permits, σ_i may be removed from the sequence and stored in the sorting buffer.

We assume that the sorting buffer is initially empty, and at the end of the process the buffer has to be empty again. Intuitively, we can picture the buffer as a truck which makes one pass along a line of packages, when the packages are occasionally loaded on and off the truck along the way.

The goal is to rearrange the input sequence in a way that packages with the same color preferably appear at consecutive positions in the output sequence. Let each maximal subsequence of packages of the same color be denoted as *color block*. Between two different color blocks there is a *color change*. Then, the goal is to minimize the number of color changes in the output sequence.

Problem 1 (Minimum Color Changes). Given a sequence of packages σ, rearrange it using a sorting buffer of capacity k to minimize the number of color changes in the output sequence.

A *solution* S to the above problem is a rearrangement of σ. Let the integer $drop_S(\sigma_i)$ denote the rearrangement step of S on which σ_i was removed from the buffer, where $drop_S(\sigma_i) = i$ if σ_i was not stored in the buffer at all. We denote by $B_S(j)$ the set of packages which are in the buffer at the beginning of step j of S.

2.2 Observations About the Optimal Solution

As noted in [2] and in [1], the following two lemmas hold for any input sequence:

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

Lemma 2. *For any optimal solution we may assume that for any color, the order of the packages of this color in the input sequence is preserved in the output sequence.*

Lemma 1 allows us to consider any color block in the input sequence as one big package. In other words, we can now replace every color block of t packages with one package of the same color, and assign that package a *size* of t. Having said that, we can now assume that the input sequence has no adjacent packages of the same color. Furthermore, we can scale the sizes with respect to the sorting buffer capacity, i.e. the buffer will have capacity 1 instead of k, and each package will have a size of $\frac{t}{k}$ instead of t. We will denote by $Size(\sigma_i)$ the size of package σ_i, and for any set of packages A, we will denote by $Size(A)$ the total size of the packages in A.

Now we turn to look at the maximization variant of the problem. If we have to pay one dollar for every color change in the output sequence, then we save a dollar whenever there are two adjacent packages in the output sequence which share the same color. According to Lemma 2, it suffices to consider only dollars saved by these adjacent packages which preserve their order from the input sequence. Each such pair of packages is called a *color-saving*. The number of color changes is minimized when the number of dollars we save is maximized, i.e. when we make the maximum number of color-savings.

Problem 2 (Maximum Color-Savings). Given a sequence of packages in different colors and sizes with no two adjacent packages of the same color, rearrange it using a sorting buffer of capacity 1 to maximize the number of color-savings in the output.

Problems 1 and 2 are equivalent because we can restrict ourselves to schedules which comply with the assumptions of Lemma 1 and Lemma 2. However, a constant approximation algorithm to the maximization problem is probably not a constant approximation algorithm to the minimization problem, and while we give a constant approximation algorithm for Problem 2, such algorithm for Problem 1 is not known.

We now extend our notation and given $\sigma = \sigma_1, \sigma_2, \ldots \sigma_n$ we use r_i to denote the ith package with color r in σ and $\overline{r_i}$ to denote the index of that package in σ (i.e. $r_i = \sigma_{\overline{r_i}}$). For each color r and index i we call $r_i - r_{i+1}$ a *pair* and we say that r_i is the *first package* of the pair and r_{i+1} the *last package* of the pair. If in the output sequence of a solution S, r_{i+1} appears adjacent to the right of r_i we say that the pair $r_i - r_{i+1}$ is a *color-saving* in S.

As an example of the problem and the notation we adopt, consider the following. The input sequence is $a_1 b_1 c_1 a_2 c_2 b_2 c_3 a_3$ (the letters denote colors and the indexes distinguish between packages of the same color). There are 8 packages in the sequence. Assume all the packages have the same size, and that the buffer has room for 2 packages (i.e. $Size(\sigma_i) = 0.5$ for all $i = 1, 2, \ldots, 8$). One of the optimal solutions S, has the output sequence $a_1 a_2 b_1 b_2 c_1 c_2 c_3 a_3$. S stores b_1 and c_1 in the buffer, drops b_1 after a_2 (at step $\overline{a_2}$), stores c_2, and drops c_1 and c_2 at step $\overline{b_2}$. The output sequence has 3 color-changes and 4 color-savings out of possible 5, with $a_2 - a_3$ the only pair which is not a color-saving.

If $r_i - r_{i+1}$ is a color-saving in S, denote $j = drops_S(r_i)$. If $j < \overline{r_{i+1}} - 1$, we say that it is a *passive* color-saving. In this case, in order to make a color-saving, r_{i+1} is not stored in the buffer, while all the packages $\{\sigma_{j+1}, \sigma_{j+2}, \ldots \sigma_{\overline{r_{i+1}}-1}\}$ are. We call these packages the *clearance zone* of $r_i - r_{i+1}$. Notice that a package cannot be in more than one clearance-zone. In the above example, the color savings $a_1 - a_2$ and $b_1 - b_2$ are passive, with $drops_S(a_1) = \overline{a_1} = 1$ and $drops_S(b_1) = \overline{a_2} = 4 < 6 = \overline{b_2} - 1$. The clearance zone of $a_1 - a_2$ is $\{b_1, c_1\}$ and the clearance zone of $b_1 - b_2$ is $\{c_2\}$.

With this terminology, we can make further assumptions on the optimal solution. We now assume that every package that gets on the buffer does it for a reason - either to make a color-saving, or to help another package make a color-saving (a passive one). We further assume that in the latter case, the package leaves the buffer as soon as it is no longer needed. And lastly, if a package gets on the buffer in order to make a color-saving, but that color-saving is passive (e.g. the package is dropped before reaching its destination), we assume that it is because one of the packages in the clearance zone starts a color-saving (otherwise - why not go all the way and make an active color-saving?).

Lemma 3. *For any optimal solution we may assume:*

1. *If r_i is stored in the buffer then either r_i is the first package of a color-saving or r_i is in the clearance zone of another color-saving.*
2. *Let $c_s - c_{s+1} - c_{s+2} - \cdots - c_{s+t}$ be a maximal sequence of passive color-savings from the same color c. Let r_j be a package in a clearance zone of one of these color-savings, and assume r_j is not the first package of a color-saving. Then, r_j is removed from the buffer at step $\overline{c_{s+t}}$.*
3. *If r_i is stored in the buffer and it is the first-package of a passive color-saving, then one of the packages in the clearance zone of that saving is the first-package of a color-saving.*

Proof. Given any solution S, we can easily transform it into one that follows the Lemma's conditions without loss of performance. We simply prevent S from storing any package that does not satisfy the conditions of part 1, and remove from the cache any package which satisfy the conditions of part 2 as soon as the buffer reaches c_{s+t} (together with all other packages in the buffer of the same color). It is easily seen that these changes in S did not interfere with any of the color-savings it had made. For part 3, if S stores r_i in the buffer and no package in the clearance-zone of $r_i - r_{i+1}$ starts a color-saving, then we can change S to carry r_i all the way to r_{i+1} (without storing any of the packages that were in the clearance-zone). Clearly, this change also does not reduce S's performance. □

Corollary 1. *Let r_i and b_j be packages, such that $b_j \in B_S(\overline{r_i})$ in a solution S. If r_i is not stored in the buffer and b_j is not starting a color-saving then $r_{i-1} - r_i$ is a color-saving in S.*

Proof. According to part 1 of Lemma 3, b_j was in the clearance zone of another color-saving $c_s - c_{s+1}$. Let c_{s+t} be the last package in the maximal sequence of passive color-savings to which $c_s - c_{s+1}$ belongs. Notice that since all the color-savings in the above sequence are passive, any package between b_j and c_{s+t} which is not stored in the buffer is the last-package of a color-saving of color c. Now, because b_j is still in the buffer even though it is not starting a color-saving we know (according to part 2 of the lemma) that $\overline{b_j} < \overline{r_i} \leq \overline{c_{s+t}}$. Since r_i is not stored in the buffer, it implies that r_i is the last-package of a color-saving of color c, and specifically, that $r_{i-1} - r_i$ is a color-saving in S. □

2.3 Deleting Pairs from the Input Sequence

We recall that the input sequence is a line of packages of different colors, and a pair consists of two consecutive packages of the same color. Given an input sequence $\sigma = \sigma_1, \sigma_2, \ldots, \sigma_n$ and a pair $r_i - r_{i+1}$ in σ, we can *delete* the pair $r_i - r_{i+1}$ by switching the color of all the packages $\{r_j\}_{j \geq i+1}$ to a new color s (i.e. for each $j \geq i+1$ the package r_j becomes s_{j-i}). Let $\sigma' = \sigma'_1, \sigma'_2, \ldots, \sigma'_n$ be the input sequence after the deletion. It is easily seen that except in the case of

$r_i - r_{i+1}$, a pair $\sigma_a - \sigma_b$ is in σ if and only if the pair $\sigma'_a - \sigma'_b$ is in σ'. As an example, consider the sequence $a_1b_1a_2b_2a_3b_3a_4b_4a_5$. If we delete the pair $a_2 - a_3$, the sequence changes to $a_1b_1a_2b_2c_1b_3c_2b_4c_3$.

If we know that we cannot gain a profit by making a color-saving $r_i - r_{i+1}$, then deleting that pair from the input sequence does not affect the optimum solution. We will use this fact extensively in the following sections, and we will also use it now to make another assumption on the input sequence.

Let $r_i - r_{i+1}$ be a pair in the input sequence. Notice that if $Size(r_i) > 1$ and the total size of the packages between r_i and r_{i+1} is also greater than 1, a feasible solution cannot make the color-saving $r_i - r_{i+1}$. Therefore, we can delete that pair from the input sequence. By repeating this process until no such pairs exist, we get the following:

Corollary 2. *If $r_i - r_{i+1}$ is a pair in the input sequence and $Size(r_i) > 1$ then the total size of the packages between r_i and r_{i+1} is at most 1.*

2.4 Classification of Intersecting Color-Savings

For every package r_i and pair $b_j - b_{j+1}$, if $\overline{r_i} \in [\overline{b_j}, \overline{b_{j+1}}]$ we say that r_i and $b_j - b_{j+1}$ *intersect*. Define $\mathcal{I}(r_i)$ to be the set of pairs intersecting r_i.

Let S be a solution and r_i a package. We classify every color-saving $I \in \mathcal{I}(r_i)$ of S into three types:

- Type A: If $I \in \{r_{i-1} - r_i, \quad r_i - r_{i+1}\}$.
- Type B: If r_i is in the clearance-zone of I.
- Type C: Otherwise.

The following two observations are immediate from the definition:

Lemma 4. *Among the color-savings, there is at most one of type B.*

Proof. Immediate, since r_i cannot be in more than one clearance-zone. □

Lemma 5. *If $b_j - b_{j+1}$ is of type C then $b_j \in B_S(\overline{r_i})$*

Proof. Since $b_j - b_{j+1}$ is not of type A or B it implies $\overline{b_j} < \overline{r_i} \leq drop_S(b_j)$ and the lemma follows. □

2.5 A Good Solution

Given σ, a sequence of packages, let $r_i - r_{i+1}$ be the pair whose first-package is the last to appear in σ ("the pair which starts last"). We say that a solution S is *good* if S either makes the $r_i - r_{i+1}$ color-saving, or, otherwise, it has a reason not to (for example - the buffer is full when r_i is reached). In a sense, a good solution is a solution which is "maximal" with respect to the last pair.

Definition 1 (good). *Let $r_i - r_{i+1}$ be the pair which starts last. Then, S is* good *if one of the following is true:*

1. *$r_i - r_{i+1}$ is a color-saving in S.*
2. *$i > 1$ and $r_{i-1} - r_i$ is a color-saving in S.*
3. *If $r_i - r_{i+1}$ is not a color-saving in S, S cannot be trivially changed to include it. Specifically:*
   - *Changing S to store r_i until step $\overline{r_{i+1}} - 1$ will render it infeasible.*
   - *If $B_S(\overline{r_i}) = \emptyset$, then changing S to store all the packages between r_i and r_{i+1} will render it infeasible.*

Notice that if condition 3 is false regarding a solution S, then S can be easily changed, without damaging existing color-savings, to include the $r_i - r_{i+1}$ color-saving and thus become good. We denote by *make_good*(S) the function that applies the above procedure to a solution S and returns the (good) result.

The following lemma states some facts about the state of the buffer after it reaches r_i in a good solution:

Lemma 6. *Let $r_i - r_{i+1}$ be the pair which starts last in σ and let S be a good solution which does not make the $r_i - r_{i+1}$ and $r_{i-1} - r_i$ color-savings. Then, at step $\overline{r_i}$:*

1. *There is no room to store r_i in the buffer (i.e. $Size(B_S(\overline{r_i})) + Size(r_i) > 1$).*
2. *All the packages in $B_S(\overline{r_i})$ are first-packages of color-savings.*

Proof. For part 1, assume on the contrary that it is possible to store r_i in the buffer at step $\overline{r_i}$. Then, since S is good, there is not enough room to store r_i all the way to r_{i+1}. Therefore, there must be another package b_j which S stores in the buffer after step $drops_S(r_i)$. Why is b_j in the buffer? It cannot start a color-saving, since r_i is the last package which starts a color-saving. So according to part 1 of Lemma 3, b_j is in the clearance zone of another color-saving $c_k - c_{k+1}$ (where $\overline{c_k} < \overline{r_i}$), and that clearance zone must lie entirely after $drops_S(r_i)$. To summarize, we have $\overline{c_k} < \overline{r_i} \leq drops_S(r_i) \leq drops_S(c_k)$, which means c_k was stored in the buffer. By part 3 of Lemma 3, it follows that there is a color-saving which starts in the clearance zone of $c_k - c_{k+1}$ and hence after r_i, a Contradiction.

For part 2, let $b_j \in B_S(\overline{r_i})$, and assume on the contrary that b_j is not the first-package of a color-saving. Then, according to Corollary 1, $r_{i-1} - r_i$ is a color-saving in S. contradiction. □

3 Local Ratio Schema

In order to use the local-ratio technique, we must have a profit function we can work with. Thus, we need to further generalize the problem by assigning a *profit* to every pair. When a pair becomes a color-saving, we gain the profit which was assigned to the pair. The goal is to make the maximum profit. This problem is equivalent to the Maximum Color-Savings Problem if we assign each pair a profit of 1.

Problem 3 (Maximum Color Savings with Profits).

Input:
- A sequence of packages in different colors and sizes with no two adjacent packages of the same color.
- A nonnegative profit assigned to every pair in the sequence.

Goal:
Rearrange the sequence using a sorting buffer of capacity 1 to make color-savings with maximum profit.

Notice that as long as the profit is nonnegative, all the lemmas and corollaries which were proved earlier in this paper also apply to optimal solutions of this generalized problem (with the same proofs).

This problem contains the bandwidth-allocation problem [6]. Indeed, we can represent each interval as a pair of packages $r_1 - r_2$ and set its profit to the profit of the interval. We set the size of r_1 as the width of the interval. We organize the packages such that pairs intersect iff their corresponding intervals intersect. Next, we insert a heavy ($Size > 1$) package before the last package of each pair, so no passive color-savings could be made (The heavy packages we add are from distinct colors so no new pairs are created). Now, every color-saving made by a feasible solution in our problem corresponds to a scheduled instance in the bandwidth-allocation problem. Since the bandwidth-allocation problem is NP-hard, it follows Problem 3 is NP-Hard too.

We are now going to examine a general instance of the above problem. Let $\mathcal{P}$ be the set of all pairs in the input sequence σ. Given a solution S, let x be a vector of the boolean variables $\{x_I | I \in \mathcal{P}\}$ such that $x_I = 1$ iff I is a color-saving in S ($x_I = 0$ otherwise). We call x the *color-savings vector* of S. The profit made by a solution S can be represented by the inner product $p \cdot x$ where x is the color-savings vector of S and p is the profit vector, with p_I the profit gained if I is a color-saving in S.

A solution S is an *r-approximation* to an instance of Problem 3, if $p \cdot x \geq \frac{1}{r} \cdot p \cdot x^*$, where x is the color-savings vector of S and x^* is the color-savings vector of an optimal solution. An algorithm is an *r-approximation algorithm* if for every instance of the problem it computes an r-approximation.

Theorem 1 (Local Ratio Theorem). *Let σ be the input sequence of an instance of Problem 3, and let p, p_1, and p_2 be profit vectors such that $p = p_1 + p_2$. Let S be a solution to the above instance, and let x be its color-savings vector. Then, if S is an r-approximation with respect to p_1 and with respect to p_2, then S is also an r-approximation with respect to p.*

Proof. Let S^*, S_1^*, S_2^* be optimal solutions of the instance with respect to the profit vectors p, p_1, and p_2 respectively, and let x^*, x_1^*, x_2^* be their corresponding color-savings vectors. Then:

$$p \cdot x = p_1 \cdot x + p_2 \cdot x \geq \frac{1}{r} \cdot p_1 \cdot x_1^* + \frac{1}{r} \cdot p_2 \cdot x_2^* = \frac{1}{r} \cdot (p_1 \cdot x_1^* + p_2 \cdot x_2^*) \geq \frac{1}{r} p \cdot x^*$$

□

3.1 Schema

We present a generic schema based on the local-ratio technique to approximate the maximum color-savings problem.

1. Delete all pairs with zero profit from the input sequence. Let $\mathcal{P}$ be the set of all the remaining pairs.
2. If $\mathcal{P} = \emptyset$, return the *empty solution* (no package is stored in the buffer).
3. Decompose p by $p = p_1 + p_2$ (The decomposition will be discussed later).
4. Solve the problem recursively using p_2 as the profit function. Let S' be the solution returned.
5. return $S = make\_good(S')$.

We now analyze the quality of the solution produced by the above schema.

Lemma 7. *Let r be a constant. Suppose that the method for decomposing the profit function is such that:*

1. *p_2 is nonnegative.*
2. *There is a pair $I \in \mathcal{P}$ such that $p_2(I) = 0$.*
3. *Every good solution is an r-approximation with respect to p_1.*

Then, the solution S returned by the schema is an r-approximation.

Proof. First of all, since in each recursive call one of the pairs has a zero profit ($p_2(I) = 0$), at least one pair is deleted in every call. Thus the number of recursive calls is bounded by the finite number of pairs, and hence the algorithm terminates in polynomial time.

Second, the first step in which pairs with zero profit are deleted clearly does not change the optimal value. Thus, it is sufficient to show that S is an r-approximation with respect to the new input sequence. The proof is by induction on the number of recursive calls. At the basis of the recursion, the returned solution is optimal (and hence an r-approximation), since no pairs remain in the input. For the inductive step, assume that S' is an r-approximation with respect to p_2. Then, since $S = make\_good(S')$ has (at least) all the color-savings in S' and p_2 is nonnegative, it follows that S is an r-approximation with respect to p_2. Since S is good, it is also an r-approximation with respect to p_1. By the Local-Ratio Theorem, it is an r-approximation with respect to p. □

4 Applying the Schema

We call a pair a *heavy pair* if its first-package has a size greater than $\frac{1}{2}$, and a *light pair* otherwise. We are now going to apply the above schema to two types of instances of the Maximum Color-Savings Problem with Profits - a light type and a heavy type. In the light type all the pairs are light and by applying the schema we will obtain a 6-approximation. In the heavy type, all the pairs are heavy and we will obtain a 3-approximation.

Using these results, the following algorithm returns a 9-approximation solution. Let σ be the input sequence and p the profit function. Then:

1. Let σ' be the resulting sequence after deleting all the heavy pairs in σ.
2. Apply the schema to σ' (light instance) and let S' be the returned solution.
3. Let σ'' be the resulting sequence after deleting all the light pairs in σ.
4. Apply the schema to σ'' (heavy instance) and let S'' be the returned solution.
5. Return the solution, between S' and S'', which gains maximum profit with respect to p.

Theorem 2. *The solution returned by the above algorithm is a* 9*-approximation.*

Proof. Let S^* be the optimal solution, with profit P^*. Let P' and P'' be the profits S^* gained from light pairs and heavy pairs, respectively, such that $P^* = P' + P''$. Then, if $P' \geq \frac{2}{3}P^*$, S' is a 9-approximation. Otherwise, $P'' \geq \frac{1}{3}P^*$ and S'' is a 9-approximation. Hence, the better solution of the two is always a 9-approximation. □

4.1 Applying the Schema on a Heavy Instance

Consider an instance of the Maximum Color-Savings problem with profits, in which all the pairs are heavy. In order to apply the schema it remains to show how to decompose the nonnegative profit function p to $p = p_1 + p_2$ such that all the conditions of Lemma 7 are satisfied. Let $r_i - r_{i+1} \in \mathcal{P}$ be the pair which starts last (recall that $\mathcal{P}$ refers to the pairs in the input sequence after pairs with zero profit have been deleted). Now, we can define the profit function p'_1 as follows:

$$p'_1(I) = \begin{cases} 1 & I \in \mathcal{I}(r_i) \\ 0 & Otherwise \end{cases} .$$

Claim. Every good solution is a 3-approximation with respect to p'_1

Proof. First, we will show that the profit of a good solution is at least 1. Let S be a good solution. If either one of the color-savings $r_{i-1} - r_i$ and $r_i - r_{i+1}$ are made by S then we are done. Otherwise, by Lemma 6, every package in the buffer at step $\overline{r_i}$ is the first package of a color-saving. Since all pairs are heavy, the buffer is either empty or has exactly one package. In the latter case, it follows that the package in the buffer is the first package of a color-saving which intersects r_i, and hence here also S makes a profit of 1.

We are left with the case the buffer is empty when it reaches r_i. This case is not possible: By Lemma 6, there is no place in the buffer to store r_i, which implies $Size(r_i) > 1$. But if that is true, S can be trivially changed to store all the packages between r_i and r_{i+1} in the empty buffer (because by Corollary 2 their total size is no more than 1). This contradicts the fact that S is a good solution which does not make the $r_i - r_{i+1}$ color-saving.

Second, we will prove that the maximum profit is at most 3. Let S be any feasible solution. Classify the color-savings of S in $\mathcal{I}(r_i)$ to 3 types, as in Section 2.4. S can make a profit of at most 2 from type A color-savings. If r_i is not stored in the buffer, S does not profit from type B color-savings and gains at most 1 (because all pairs are heavy) from type C. If r_i is stored in the buffer, S does not profit from type C color-savings and gains at most 1 from type B. In both cases, S profits no more than 3. □

We note that for every $\epsilon \geq 0$, every good solution is a 3-approximation with respect to $\epsilon p'_1$. It is easily seen that by choosing $\epsilon_0 = max\{\epsilon | p - \epsilon p'_1 \geq 0\}$ to define $p_1 = \epsilon_0 p'_1$ and $p_2 = p - \epsilon_0 p'_1$ we ensure that one of the pairs has a p_2-profit of 0 and still keep all the prices nonnegative. This decomposition satisfies all the conditions of Lemma 7, and it allows us to apply the schema on any heavy instance of the problem to receive a solution which is a 3-approximation.

4.2 Applying the Schema on a Light Instance

Consider an instance of the Maximum Color-Savings problem with profits, in which all the pairs are light. In order to obtain a 6-approximation we are going to decompose the problem once more. For each color r, a pair $r_i - r_{i+1}$ is *even* (*odd*, respectively) if i is even (odd). We call an instance of the maximum color-savings with profits problem *reduced* if every package belongs to at most one pair, or in other words, if there are at most 2 packages of each color. We observe that if we delete all the even (odd) pairs, we are left with a reduced instance. We will later show that by applying the schema to a reduced-light instance, we can obtain a 3-approximation. The following algorithm will thus yield a 6-approximation:

1. Let σ' be the resulting sequence after deleting all the even pairs in σ.
2. Apply the schema to σ' (reduced-light) and let S' be the returned solution.
3. Let σ'' be the resulting sequence after deleting all the odd pairs in σ
4. Apply the schema to σ'' (reduced-light) and let S'' be the returned solution.
5. Return the solution, between S' and S'', which gains maximum profit with respect to p.

Lemma 8. *The solution returned by the above algorithm is a* 6*-approximation.*

Proof. Let S^* be the the optimum solution, with profit P^*. Let P' and P'' be the profits S^* gained from even and odd pairs, respectively ($P^* = P' + P''$). Then, either $P' \geq \frac{1}{2}P^*$ or $P'' \geq \frac{1}{2}P^*$. Since S' and S'' are 3-approximations with respect to σ' and σ'', the better solution of the two is a 6-approximation. □

Applying the Schema on a Reduced-Light Instance. It remains to show how to apply the schema on a reduced-light instance to obtain a 3-approximation. As in the previous subsection, we need to show how to decompose the nonnegative profit function p by $p = p_1 + p_2$ such that all the conditions of Lemma 7 are satisfied. Since the instance is reduced, all the pairs in $\mathcal{P}$ are of the form $b_1 - b_2$ where b is a color. Let $r_1 - r_2 \in \mathcal{P}$ be the pair which starts last, and define $\delta \triangleq 1 - Size(r_1)$ (notice that $\delta \geq \frac{1}{2}$). We define p'_1 as follows:

$$p'_1(b_1 - b_2) = \begin{cases} \delta & b_1 - b_2 = r_1 - r_2 \\ Size(b_1) & b_1 - b_2 \in \mathcal{I}(r_1) \setminus \{r_1 - r_2\} \\ 0 & Otherwise \end{cases} .$$

Claim. Every good solution is a 3-approximation with respect to p'_1

Proof. First, we will show that the profit of a good solution is at least δ. Let S be a good solution. If $r_1 - r_2$ is a color-saving in S then we are done. Otherwise, by Lemma 6 we know that $Size(B_S(\overline{r_1})) > 1 - Size(r_1) = \delta$. Let b_i be a package in $B_S(\overline{r_1})$. Then, by part 2 of Lemma 6, b_i is the first-package of a color-saving in S. Since the instance is reduced it follows that $i = 1$, $b_2 \notin B_S(\overline{r_1})$, and hence $b_1 - b_2$ is a color-saving in S which intersects r_1. Therefore, S gains $p'_1(b_1 - b_2) = Size(b_1) = Size(b_i)$ for every $b_i \in B_S(\overline{r_1})$. It follows that S makes a profit of at least $Size(B_S(\overline{r_1})) > \delta$.

Second, we will prove that the maximum profit is at most 3δ. Let S be any feasible solution. Classify the color-saving of S in $\mathcal{I}(r_1)$ into 3 types, as in Section 2.4. S can make a profit of at most δ from type A color-savings (namely $r_1 - r_2$). If r_1 is not stored in the buffer, S does not profit from type B color-savings and gains at most $Size(B_S(\overline{r_1})) \leq 1$ from type C, for a total of no more than $\delta + 1$. If r_1 is stored in the buffer, S can profit at most $Size(B_S(\overline{r_1})) \leq 1 - Size(r_1) = \delta$ from type C color-savings and at most $\frac{1}{2}$ from type B (because there is no more than one color-savings of type B, and it is light), for a maximum total of $2\delta + \frac{1}{2}$. In both cases, S profits no more than 3δ. □

As before, by choosing $\epsilon_0 = \max\{\epsilon | p - \epsilon p'_1 \geq 0\}$ to define $p_1 = \epsilon_0 p'_1$ and $p_2 = p - \epsilon_0 p'_1$ we get the required decomposition, and obtain a 6-approximation algorithm for heavy instances.

References

1. J. S. Kohrt and K. Pruhs. A constant approximation algorithm for sorting buffers. *Proceedings of the Sixth Latin American Symposium (LATIN 2004)*, volume 2976 of Lecture Notes in Computer Science, pages 193-202. Springer-Verlag, 2004.
2. H. Räcke, C. Sohler, and M. Westermann. Online Scheduling for Sorting Buffers. *Proceedings of the 10th ESA (Rome)*, pp. 820–832, 2002.
3. R. Bar-Yehuda. One for the price of two: a unified approach for approximating covering problems. *Algorithmica* 27, 131–144, 2000.
4. R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Discrete Mathematics 25*, 27-46, 1985.
5. R. Bar-Yehuda and D. Rawitz. On the Equivalence between the Primal-Dual Schema and the Local-Ratio Technique. *Proceedings of RANDOM-APPROX 2001*, p.24–35, 2001.
6. A. Bar-Noy, R. Bar-Yehuda, A. Freund , J. Naor , B. Schieber. A unified approach to approximating resource allocation and scheduling, *Journal of the ACM (JACM)*, v.48 n.5, p.1069-1090, September 2001.
7. Th. Epping, W. Hochstättler. Storage and Retrieval of Car Bodies by the Use of Line Storage Systems. *Technical report* btu-lsgdi-001.02, BTU Cottbus, Germany, 2002.