Experimental Evaluation of Reordering Buffer Management Algorithms

by

Medha Vasanth

A Thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of

Master of Computer Science

in

School of Computer Science
Carleton University
Ottawa, Ontario, Canada
April 2015

Copyright ©
2015 - Medha Vasanth
Abstract

In the Reordering Buffer Management (RBM) problem, we are given an input sequence of size $N$ and a buffer $B$ of size $k$ ($k < N$). Each item in the input sequence is characterized by a specific attribute referred to as its “colour”. The defining characteristic of this problem is that a cost is incurred when we switch from one colour to another in the resulting output sequence. This cost can be uniform or non-uniform.

Our goal is to permute the input sequence using the buffer $B$ such that the cost of switching between colours is minimized. In the online version of the RBM problem, the input sequence is not known in advance. We propose a new algorithm in this thesis, Accelerated Threshold (Acc-T) for the online version of the RBM problem with non-uniform costs.

This thesis also presents experimental results for several existing uniform and non-uniform cost algorithms, for the online version of the RBM problem. We compare these algorithms against different combinations of input sequence types, cost functions, buffer sizes and number of colours.
I would like to dedicate my thesis to my grandfather Late Shri. A. S Varadaraja Iyengar who had the conviction that I will be able to pursue an advanced degree in Computer Science.
Acknowledgments

I am greatly indebted to my supervisors Anil Maheshwari and Shalini Periyalwar for accepting me as their student and for their continued support throughout my studies at Carleton University. They have helped me identify my areas of interest and always provided me with insightful feedback on my work. Their encouragement in my choice of wanting to do a co-op during my Master’s degree was phenomenal and to that I am particularly grateful.

I would also like to express my gratitude towards my supervisors and the School of Computer Science, Carleton University for all the generous funding I have received by means of Research and Teaching Assistantships.

The Computational Geometry Laboratory, Carleton University, has been a great place to work in. I would like to thank the entire faculty and all my fellow students at the CG Lab for providing an excellent environment to work in.

Much credit is also due to Gregory Bint for his insightful comments on my thesis and for all the fun times I have had in Canada.

Finally, my Master’s degree would never have seen the light of the day without all the love and blessings from my family.
# Table of Contents

**Abstract** ii

**Acknowledgments** iv

**Table of Contents** v

**List of Tables** viii

**List of Figures** ix

1 **Introduction** 1

  1.1 Motivation ................................................. 1
  1.2 Problem Definition ........................................ 3
    1.2.1 Illustration of the RBM problem ....................... 4
  1.3 Summary of Contributions ................................. 6
  1.4 Organization of the thesis .............................. 7

2 **Background** 8

  2.1 Minimization Variant ..................................... 8
  2.2 Maximization Variant ................................. 13
  2.3 Experimental Evaluations ............................. 14

3 **Algorithms for Reordering Buffer Management** 17
List of Tables

1. Summary of Algorithms Implemented ........................................ 21
2. Summary of input parameters and their values for our experiments ... 40
3. Input and Output Parameters used for our experiments ................. 42
4. Summary of Uniform Cost Algorithm Performance ........................ 51
5. Summary of non-uniform cost algorithms performance using medium
   buffers w.r.t switch ratios ................................................ 93
6. Summary of non-uniform cost algorithms performance using medium
   buffers w.r.t cost ratios .................................................. 94
7. Summary of non-uniform cost algorithms performance using large
   buffers w.r.t switch ratios ................................................ 95
8. Summary of non-uniform cost algorithms performance using large
   buffers w.r.t switch ratios for the Sequential Block Sequence ....... 96
9. Summary of non-uniform cost algorithms performance using large
   buffers w.r.t cost ratios .................................................. 97
10. Summary of non-uniform cost algorithms performance using large
    buffers w.r.t cost ratios for the Sequential Block Sequence ........ 98
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Illustrating the RBM problem</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>Impact of Input Sequence Size</td>
<td>44</td>
</tr>
<tr>
<td>3</td>
<td>Uniform Cost Algorithms against Alternation Sequences</td>
<td>46</td>
</tr>
<tr>
<td>4</td>
<td>Uniform Cost Algorithms against Delta Sequences</td>
<td>47</td>
</tr>
<tr>
<td>5</td>
<td>Uniform Cost Algorithms against Random Sequences</td>
<td>48</td>
</tr>
<tr>
<td>6</td>
<td>Uniform Cost Algorithms against Random Block Sequences</td>
<td>49</td>
</tr>
<tr>
<td>7</td>
<td>Uniform Cost Algorithms against Sequential Block Sequences</td>
<td>50</td>
</tr>
<tr>
<td>8</td>
<td>Switch Ratios of Non-Uniform Cost Algorithms against Alternation Sequences with the Cost-Equals-Quadratic-Colour cost function for Small Buffers</td>
<td>54</td>
</tr>
<tr>
<td>9</td>
<td>Switch Ratios of Non-Uniform Cost Algorithms against Alternation Sequences with the Colour-Difference cost function for Small Buffers</td>
<td>55</td>
</tr>
<tr>
<td>10</td>
<td>Switch Ratios of Non-Uniform Cost Algorithms against Alternation Sequences with the Cost-Equals-Quadratic-Colour cost function for Medium Buffers</td>
<td>57</td>
</tr>
<tr>
<td>11</td>
<td>Cost Ratios of Non-Uniform Cost Algorithms against Alternation Sequences with the Colour-Difference cost function for Medium Buffers</td>
<td>58</td>
</tr>
<tr>
<td>12</td>
<td>Cost Ratios of TLC and Acc-T against Alternation Sequences with the Cost-Equals-Colour cost function for Large Buffers</td>
<td>59</td>
</tr>
</tbody>
</table>
13 Cost Ratios of TLC and Acc-T against Alternation Sequences with the Random-Cost cost function for Large Buffers
14 Switch Ratios of Non-Uniform Cost Algorithms against Delta Sequences with the Cost-Equals-Quadratic-Colour cost function for Small Buffers
15 Switch Ratios of Non-Uniform Cost Algorithms against Delta Sequences with the Cost-Equals-Quadratic-Colour cost function for Small Buffers
16 Switch Ratios of Non-Uniform Cost Algorithms against Delta Sequences with the Random-Cost cost function for Medium Buffers
17 Cost Ratios of Non-Uniform Cost Algorithms against Delta Sequences with the Colour-Difference cost function for Medium Buffers
18 Switch Ratios of TLC and Acc-T against Delta Sequences with the Cost-Equals-Quadratic-Colour cost function for Large Buffers
19 Cost Ratios of Acc-T and TLC against Delta Sequences with the Random-Cost cost function for Large Buffers
20 Switch Ratios of Non-Uniform Cost Algorithms against Random Sequences with the Random-Cost cost function for Small Buffers
21 Cost Ratios of Non-Uniform Cost Algorithms against Random Sequences with the Colour-Difference cost function for Small Buffers
22 Switch Ratios of Non-Uniform Cost Algorithms against Random Sequences with the Cost-Equals-Colour cost function for Medium Buffers
23 Cost Ratios of Non-Uniform Cost Algorithms against Random Sequences with the Cost-Equals-Quadratic-Colour cost function for Medium Buffers
24 Switch Ratios of Acc-T and TLC against Random Sequences with the Cost-Equals-Colour cost function for Large Buffers
<table>
<thead>
<tr>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>Cost Ratios of Acc-T and TLC against Random Sequences with the Colour-Difference cost function for Large Buffers</td>
</tr>
<tr>
<td>26</td>
<td>Switch Ratios of Non-Uniform Cost Algorithms against Random Block Sequences with the Cost-Equals-Quadratic-Colour cost function for Small Buffers</td>
</tr>
<tr>
<td>27</td>
<td>Cost Ratios of Non-Uniform Cost Algorithms against Random Block Sequences with the Random-Cost cost function for Small Buffers</td>
</tr>
<tr>
<td>28</td>
<td>Cost Ratios of Non-Uniform Cost Algorithms against Random Block Sequences with the Cost-Equals-Quadratic-Colour cost function for Medium Buffers</td>
</tr>
<tr>
<td>29</td>
<td>Cost Ratios of Non-Uniform Cost Algorithms against Random Block Sequences with the Colour-Difference cost function for Medium Buffers</td>
</tr>
<tr>
<td>30</td>
<td>Switch Ratios of Acc-T and TLC against Random Block Sequences with the Cost-Equals-Quadratic-Colour cost function for Large Buffers</td>
</tr>
<tr>
<td>31</td>
<td>Switch Ratios of Acc-T and TLC against Random Block Sequences with the Random-Cost cost function for Large Buffers</td>
</tr>
<tr>
<td>32</td>
<td>Cost Ratios of Non-Uniform Cost Algorithms against Sequential Block Sequences with the Cost-Equals-Quadratic-Colour cost function for Small Buffers</td>
</tr>
<tr>
<td>33</td>
<td>Switch Ratios of Non-Uniform Cost Algorithms against Sequential Block Sequences with the Colour-Difference cost function for Small Buffers</td>
</tr>
<tr>
<td>34</td>
<td>Switch Ratios for Non-Uniform Cost Algorithms against Sequential Block Sequences with the Cost-Equals-Quadratic-Colour cost function for Medium Buffers</td>
</tr>
</tbody>
</table>
35 Switch Ratios for Non-Uniform Cost Algorithms against Sequential Block Sequences with the Colour-Difference cost function for Medium Buffers .................................................. 89

36 Cost Ratios of Acc-T and TLC against Sequential Block Sequences with the Cost-Equals-Colour cost function for Large Buffers ................................. 90

37 Switch Ratios of Acc-T and TLC against Sequential Block Sequences the Random-Cost cost function for Large Buffers ................................. 91
Chapter 1

Introduction

The Reordering Buffer Management (RBM) problem, also known as the Sorting Buffers problem, was first introduced by Räcke, Sohler and Westermann [1], in the year 2002. In this problem, a stream of $N$ items are to be processed by a server. Each of these items are characterized by the same attribute. A change in the attribute value between two consecutive items incurs a cost. For the sake of simplicity we assume the attribute to be its “colour”. Räcke et al. [1], proposed that this cost can be minimized if the server is equipped with a $k$ ($k < N$) slot random access buffer. The buffer can be used to permute or reorder the input stream in such a way that we have long sub-sequences of items of the same colour in the output sequence thereby minimizing the the cost of switching between colours. Our goal is to reorder the input sequence using the buffer such that the cost of switching between colours is minimized.

1.1 Motivation

This problem has many applications in computer science and economics. Recent advances in the area of computer game development has led to many sophisticated games that consist of rendering millions of 3D polygons with different attributes.
This involves switching between different attributes like colour and texture of the polygons being rendered. Constant switching between attributes while rendering can cause delays and disrupt the smooth rendering of different scenes in a video game. In this case it would be beneficial to render all items of one attribute before the other since the cost incurred by switching between textures and colours is substantial. This has necessitated the need for cost efficient strategies to enable smooth rendering of multiple attributes in computer games. Reordering buffers can be used to permute the stream of polygons to render large sub-sequences of the same attribute, thereby minimizing the cost and delays in a graphics rendering system.

Another application of the RBM problem arises in network communication system where a server has to send packets to different nodes on a network. Sending packets to a particular node in the network involves a startup cost incurred at the receiving node. In this case it is beneficial for the server to wait for all the packets for a particular node to arrive at the server rather than constantly switching between the nodes. The buffer would ensure that the server first has all the packets intended for a particular node before data transmission.

A paradigm shift in data storage techniques has resulted in most companies to store large amounts of data on a cloud based server. Standalone nodes have to store and access files from the server on the cloud. Placing a buffer on the cloud server has the benefit of sending large number of files to the designated machine thereby reducing the cost of switching between different standalone nodes.

Reordering buffers can also be used to improve the performance of file servers. A high capacity file server consists of many files which are either open or closed. Read and write accesses to these files can be performed only when the particular file is open. Opening a closed file incurs a cost. Hence a reordering buffer can be used to reorder read and write accesses in such a way that all reads and writes and performed to a file that is currently open.
A preliminary application of the reordering buffers can be seen in the car painting plant [2], where a sequence of cars are to be painted with a final colour. A cost is incurred if two consecutive cars are to be painted with different colours. This cost can be minimized by reordering the sequence of the cars to be painted using a reordering buffer.

1.2 Problem Definition

In the Reordering Buffer Management (RBM) problem, we have an input sequence of \( N \) items. Each of these items are characterized by a particular attribute, which we refer to as the colour of the item. The defining characteristic of this problem is that there is a cost incurred when switching between two items of different colours. In the case of uniform cost RBM problems, the same cost is incurred when we switch from one colour to another. While in the case of non-uniform cost RBM problems, different costs are incurred when we switch from one colour to the next. There are \( C \) different colours that can occur in the input stream. Our goal is to permute this input stream in such a way that the cost of switching between items of different colours is minimized in the resulting output sequence.

Räcke, Sohler and Westermann [1], proposed that this problem can be solved by using a \( k \) \( (k < N) \) slot random access buffer that can be used to permute the input sequence. Every incoming item is first stored in the buffer. When the buffer gets filled up with the first \( k \) items, we choose to evict an item from the buffer and add it to the output sequence making room for the next item from the input sequence. We evict one item at each step of the algorithm. The colour to be evicted from the buffer is based on the reordering algorithm in question. We select a new colour only when the buffer no longer contains items of the currently chosen colour. This process of selecting the colour to evict and adding it to the output sequence is performed until
the entire input sequence is processed and the buffer is empty.

All reordering algorithms should satisfy the condition that the buffer is empty at the beginning and end of the reordering, that is, all the input items have to be processed by the buffer and added to the output sequence.

In the applications listed in Section 1.1 the network communication systems, cloud based servers and the car painting example correspond to real world examples of the RBM problem with uniform cost. File server and the 3D polygon rendering applications correspond to examples of the RBM problem with non-uniform costs.

1.2.1 Illustration of the RBM problem

We illustrate the RBM problem with the following example. Consider an input sequence with 10 items ($N = 10$), buffer size of 4 ($k = 4$) and 3 colours ($C = 3$). We use the integers 1, 2 and 3 to represent the three colours that can be present in the input sequence $I$, where $I = 1, 2, 3, 1, 2, 3, 1, 2, 3, 1$. As we can see, there is a colour change between every pair of items in the input sequence. Each such colour change is referred to as a switch and we say that there are 9 switches in the input sequence. For this example, we assume that a uniform (unit) cost is incurred when we switch from one colour to the next, hence this sequence incurs a cost of 9.

For the first 4 ($k$) steps, we fill the buffer with the first 4 items (1, 2, 3, 1) of the input sequence. Eviction begins at Step 5 when the buffer is full. Assume that we use a greedy algorithm that evicts items of the colour that has the most occurrences in the buffer. At step 5, the greedy algorithm selects the colour 1, evicts one of the two items of colour 1 and appends it to the output sequence, making room for the next item from the input sequence 2, to enter the buffer. At this point, we do not make a colour change since our buffer still has one other item with the colour 1, which is evicted in the following step. Note that items are always added at the end of the buffer, whereas items can be evicted from any slot from the buffer.
The greedy algorithm continues to evict items of the colour that has the most occurrences in the buffer until the entire input sequence is processed. A snapshot of the greedy algorithm at Steps 9 and 10 are illustrated in Figure 1. At this point, the buffer has processed the first 9 items of the input sequence (1, 2, 3, 1, 2, 3, 1, 2, 3); the next item 1 is awaiting entry into the buffer. The output sequence currently has items $O = 1, 1, 2, 2, 2$, while the buffer has items $B = 3, 3, 1, 3$. At Step 9, the greedy algorithm evicts the last item of colour 2 and 3 enters the buffer. Following which 3 is chosen as the new colour to evict and one item of colour 3 is evicted making room for the last item of the input sequence. The buffer continues to process items in this manner until all the items have been appended to the output sequence.

![Diagram](image.png)

**Figure 1:** The buffer $B$ in action. The items that are currently in the buffer are represented by solid lines and the item that will enter the buffer after an item is evicted is represented by dashed lines. At Step 9, the colour that is assigned to the output sequence is 2, 2 is evicted and 3 enters the buffer. At Step 10, the greedy algorithm changes the colour to be evicted to 3 and evicts one item of that colour, and 1 enters the buffer. The figure also shows the input sequence and the reordered output sequence.

After reordering, the resulting output sequence is $O = 1, 1, 2, 2, 2, 3, 3, 1, 1$, consisting 3 colour switches, which also means that the cost incurred is 3. With a buffer of size 4, we can reduce the number of switches (and cost) from 9 to 3. We compute the switch ratio, which is the number of switches present in the output sequence divided by the number of switches present in the input sequence, i.e., $3/9 = 0.3333$. We
use these ratios to compare different algorithms later on in the thesis.

### 1.3 Summary of Contributions

In this thesis, we propose a new algorithm, Accelerated Threshold (Acc-T) for the Reordering Buffer Management Problem with non-uniform costs. In addition, we implement different existing reordering buffer algorithms and perform various experiments that study the behaviour of different algorithms for different kinds of input sequences, buffer sizes, number of colours and cost functions. We use the following input parameters for our experiments:

- Input Sequences Types - Alternation, Delta, Random, Random Block and Sequential Block Sequence (refer to Section 4.1 for details)
- Input Sequence Sizes (100, 1000, 10,000 and 100,000)
- Buffer Size (2, 5, 10, 20, 50, 100 and 500)
- Number of Colours (2, 5, 10, 20 and 50)
- Cost Function Type - Cost-Equals-Colour cost function, Cost-Equals-Quadratic-Colour cost function, Random-Cost cost function, Colour-Difference cost function and Uniform-Cost cost function (refer to Section 4.2 for details)

When we run our algorithms, we record the number of switches present in the input and output sequences, and the cost incurred for the input and output sequences on each run of the algorithm. Using these recorded values, we compute the output/input switch ratio, which is the ratio of the number of switches in the output sequences to the number of switches in the input sequence. Similarly, we also compute the output/input cost ratio. These ratios are used to compare different algorithms for
different combinations of input sequence types, cost functions, buffer sizes and number of colours.

Our experimental results reveal that, Acc-T achieves lower switch ratios across all combinations of input sequences, cost functions, buffer sizes and number of colours. In addition, we observe that our algorithm, achieves a slightly better cost ratio for some combination of input sequence and cost function combinations, refer to Section 4.7 for details.

1.4 Organization of the thesis

The rest of the thesis is organized as follows. In Chapter 2, we review related literature. In Chapter 3, we present the pseudocodes for the algorithms that we implement and propose our new algorithm Acc-T for the Reordering Buffer Management problem. In Chapter 4, we describe the implementation details and the experimental results for the different algorithms. Chapter 5 concludes the thesis with a summary of our results and directions for future work.
Chapter 2

Background

In this chapter, we briefly review some existing algorithms for different variations of the RBM problem. In Section 2.1, we review some of the algorithms proposed for the minimization variant of the RBM problem. In this case, the objective is to minimize the number of switches in the output sequence thereby minimizing the cost incurred.

In Section 2.2, we look at some of the algorithms that have been proposed for the maximization variant of the RBM problem. In this case, the goal is to maximize the number of colour switches that are eliminated from the input sequence. In Section 2.3, we look at some experimental results for previously proposed algorithms.

2.1 Minimization Variant

The problem of Reordering Buffer Management was first introduced by Räcke, Sohler and Westermann in 2002 [1]. It was initially called the Sorting Buffers problem, where a service station had a buffer called the sorting buffer. This buffer was used to permute the input sequence. Each item in the input sequence was characterized by a particular attribute, which we refer to as the “colour” of the item. A random access buffer was used to permute these items such that the output sequence had long sub-sequences of items with the same colour. There was also a cost involved
in switching from one colour to another. In their problem formulation, the cost of switching between colours was set to be uniform, in that the cost of switching to any colour was the same. Their model is also based on the online version of the problem where the input sequence is not known in advance. So the buffer has to work with partial knowledge of the input sequence. In their paper, they prove that algorithms like First In First Out (FIFO), Least Recently Used (LRU) and Largest Colour First (LCF) are not suitable for the sorting buffers problem and prove a lower bound of $\Omega(\sqrt{k})$ for FIFO and LRU, and a lower bound of $\Omega(k)$ for LCF, where $k$ is the buffer size. They also propose a deterministic algorithm called Bounded Waste (BW). As is the case with all online algorithms, they use competitive analysis proposed by Sleator and Tarjan [3], and prove that their algorithm has a competitive ratio of $O(\log^2(k))$. We present further details in Section 3.3.1.

While Bounded Waste was an efficient strategy for uniform costs, it performed poorly under the non-uniform cost scenarios. Englert and Westermann [4], proposed a new deterministic strategy called Maximum Adjusted Penalty (MAP) to handle non-uniform costs. They use the following mathematical notation. Colours are integers in the range $1 \ldots C$. If $i$ denotes the item from the input sequence that is currently being processed, $c(i)$ denotes the colour of item $i$, and $w(c)$ denotes the cost of the colour $c$, they defined their cost function to be the following: $w(c) = |c(i) - c(i+1)|$. This cost function is specifically suited for applications such as disk scheduling and paging. As in the case of Bounded Waste, they also used competitive analysis and prove that the competitive ratio of their algorithm is $O(\log(k))$, where $k$ is the size of the buffer. They also prove a lower bound of $k$ on the competitive ratio for the Bounded Waste algorithm for non-uniform costs. While the competitive ratio was improved to $O(\log(k))$, this algorithm is computationally very intensive and hence not applicable in practical scenarios. The details of this algorithm are presented in Section 3.3.4.
Improving the results of Englert and Westermann [4], Rabani and Elgrabli [5] propose an algorithm called Threshold or Lowest Cost (TLC) for the online version of the RBM problem that achieves a better competitive ratio than that of MAP, but is not as computationally intensive. Their algorithm is also designed for the non-uniform cost scenarios, but has a considerably better competitive ratio. Their algorithms maintains and updates counters for items in the buffer and compares the values of these counters with the cost associated with each colour to select a new colour for eviction. The analysis of this algorithm uses an intricate linear programming relaxation scheme. Using this scheme, they prove that the competitive ratio of TLC is $O\left(\frac{\log k}{\log \log k}\right)$. We present the details of this algorithm in Section 3.3.5.

Adamaszek et al [6] improved the results obtained by Rabani and Elgrabli [5] and gave lower bounds for deterministic and randomized algorithms. Their contributions are two fold. Firstly, they give a $\Omega\left(\sqrt{\log k/\log \log k}\right)$ lower bound on the competitive ratio of deterministic algorithms and a $\Omega(\log \log k)$ lower bound on the competitive ratio of randomized algorithms. Secondly they also propose a deterministic algorithm and prove the competitive ratio to be $O(\sqrt{\log k})$. This is the best known result for deterministic algorithms.

Limited results have been presented for the offline version of the RBM problem. Rabani and Elgrabli [7] improved on the results presented by Adamaszek [6] and proposed the first constant factor approximation algorithm for the minimization variant of the offline version of the RBM problem. It is to be noted that the offline version of the problem is proven to be NP-hard [8-10]. They also prove that this algorithm does better than any online algorithm for the minimization problem. Their algorithm is a linear program (LP) that is based on the same complex linear programming relaxation scheme that they proposed in [5]. The objective function is designed to minimize the number of switches in the output sequence thereby minimizing the cost incurred by the algorithm. The constraints of the linear program ensure that the
following requirements are satisfied: the buffer should be empty at the end of the algorithm, only one item is removed in each time step and the algorithm should not switch colours when the active colour is still present in the buffer. They prove that these constraints guarantee that the linear program produces the correct cost incurred by the algorithm. The conversion of the fractional solution provided by the LP to integral solution without increasing the cost incurred by the algorithm by more than a constant factor is done in different time phases. Each time phase begins at the end of the previous time phase. The goal of each time phase to evict one or more colours until the target is reached and the entire input sequence has been reordered. In each time phase, the fractional solution increases its cost by a small constant factor. Their elaborate analysis proves that all colours are evicted with only a constant factor increase in the cost when transforming from the fractional to the integral solution.

The same relaxation proposed in [5] has also been used by Rabani and Elgrabli [11], in designing a randomized algorithm for the online version of the RBM problem. Using the same constraints as mentioned earlier [7], they formulate their randomized algorithm which consists of linear and dual programs that works in two phases, which run in parallel. The online linear program produces a feasible solution. This feasible solution is continually being fed into a “rounding” procedure that produces the integral solution. In other words, the integral solutions is produced “on the fly” as the linear program outputs the feasible fractional solution. In their analysis, they prove that the competitive ratio of this algorithm is $O(\log \log k)$. To the best of our knowledge this is the first algorithm that uses randomization to propose an algorithm for the online version of the RBM problem. The main drawback of this algorithm is that it is only applicable to uniform cost scenarios.

Im and Moseley [12] used linear programming relaxation techniques to overcome the bottleneck of uniform costs [11] and designed a new algorithm for the online version of the RBM problem with non-uniform costs. Their results are twofold. They
first present a randomized algorithm that achieves a competitive ratio of $O(\log \log k \gamma)$ where $\gamma$ is the ratio of the maximum to minimum cost that can be present in the input sequence. In addition to that, they also provide a 66.0832 - approximation algorithm for the uniform cost scenario of the online version of the RBM problem. This improves the constant factor approximation presented by Rabani and Elgrabli [7].

An interesting extension to the online RBM problem was proposed by Adamaszek et al. [13] which uses advice bits to choose the new active colour to be evicted. Use of advice bits for online algorithms was first proposed by Emek et al. [14]. Adamaszek et al [13] have used the same model as presented in [14] and applied it to the online version of the RBM problem. In the online RBM problem with advice, each item that enters the buffer also has a constant number of advice bits that gives the algorithm some information about the future or some information about an optimal solution used. In particular, the advice bits tell the algorithm if it should immediately output the items of colour $c$, if it should wait until all items of colour $c$ arrive in the buffer, or if it should only wait for a small amount of time and then start outputting items of colour $c$. They prove that using advice bits for reordering achieves a competitive ratio of $1 + \epsilon$ for any $\epsilon > 0$. They also prove that this ratio can be achieved by using $O(\log(\frac{1}{\epsilon}))$ bits of advice per request. In addition to this result, they also prove a lower bound of $\Omega(\log k)$ on the number of advice bits per request for an optimal offline algorithm.

Another variation for the online version of the RBM problem was proposed by Azar et al. [15]. In this problem, we have a service station that consists of $m$ servers and a reordering buffer of size $k$. Each server is configured to process items of a particular colour. When an item arrives, it is stored in the buffer. If there is a server configured for an item in the buffer, that item is evicted and sent to the server which then processes it. If not then one of the servers has to be reconfigured with a different colour so that the next item present in the buffer can be processed.
This incurs a cost. Hence in this case, the goal is to minimize the cost incurred when servers have to be reconfigured to process different colours. Azar et al. \cite{15} propose a randomized algorithm for this problem that achieves a competitive ratio of \( O(\sqrt{m \ln k}) \). In addition to this, they also present a deterministic algorithm that achieves a competitive ratio of \( O(\min(m^2 \ln k, mk)) \).

It can be observed that the paging problem, proposed by Sleator and Tarjan \cite{3} is a special case of this problem where there is just one slot in the buffer or \( k = 1 \). The online version of the RBM problem, first proposed by Räcke et al \cite{1}, is also a special case of this problem where we have a single server, that is \( m = 1 \).

### 2.2 Maximization Variant

A slight variation of the offline version of the RBM problem was considered by Kohrt and Pruhs \cite{16}. In their variation of the problem, the input sequence consists of items of different colours and a random access buffer was used to permute the input sequence to produce the output sequence. But instead of minimizing the number of switches in the output sequence, they considered the maximization objective where their goal was to maximize the number of switches that were eliminated from the input sequence. For example, if the input sequence had 10 items and 9 switches and after permuting the sequence, there were 5 switches, then 4 switches were eliminated from the input sequence. Their objective was to maximize this number. They presented the first polynomial time 20 - approximation algorithm for the maximization problem in the offline setting.

Following the work by Kohrt and Pruhs \cite{16}, Bar-Yehuda and Laserson \cite{17} propose a 9 - approximation algorithm for the maximization variant of the offline version of the sorting buffers problem. However, in their model, items can either be added to the buffer or directly to the service station. In their result, they first make the
following observations about the optimal solution: Firstly, if two items of the same colour appear consecutively in the input sequence, then there is an optimal solution for which the the same two items appear consecutively in the output sequence of the optimal solution. Secondly, there exists an optimal solution that preserves the order in which the items of that colour appear in the input sequence. Using these two observations, they say that each colour block of the same colour will be considered as a single item, where its size is equal to the number of items in that block. Also, they redefine their buffer size to be 1, and each item in the buffer now occupies the fraction $\frac{t}{k}$, where $t$ is the size of the item. Using these new definitions, they say that a pair of items $p(c(i), c(i+1))$ with colour $c$ is said to be a colour saving pair if the items $i$ and $i + 1$ appear consecutively.

Based on these definitions, they present a local-ratio schema technique for the maximization variant of the RBM problem. In the final stage, they apply this schema and propose a 9-approximation algorithm which repeatedly applies the schema on different kinds of colour saving pairs to maximize the colour savings in the output sequence.

This improves the 20-approximation of the previous algorithm. To the best of our knowledge, the 20 and the 9-approximation algorithms are the only two constant factor approximation algorithms known for the maximization variant of the problem.

### 2.3 Experimental Evaluations

Krokowski et al. [18], provide experimental results of the strategies described in [1] for rendering images with different colours and textures. In addition they also propose the Random Choice and Round Robin strategy where the former randomly selects an item from the buffer to evict and the latter uses a selection pointer and evicts the item pointed to by the selection pointer. Their experiments reveal that Bounded Waste,
Random Choice and Round Robin essentially achieve the same performance with Bounded Waste giving an additional 3% of reduced colour changes. They also found that this approach reduces the rendering time by 35% making it time efficient. Simple data structures make this strategy to be efficiently implemented in both hardware and software.

More extensive experimental results were presented by Englert et al. [19] in 2009. They compare the algorithms First In First Out (FIFO), Least Recently Used (LRU), Most Colour First (MCF), Maximum Adjusted Penalty (MAP), Random Choice (RC) and the Round Robin (RR) algorithms. It is to be noted that because of the computational complexity of MAP, they have implemented a simpler version of the algorithm. As per their implementation, the penalty of each item in the buffer is set to the count of items of a particular colour present in the buffer, while the original algorithm [4] computes the adjusted penalty based on the buffer size and the count of the items of each colour in the buffer. This simplification makes their algorithm, similar to the Bounded Waste (BW) algorithm presented in [1].

To evaluate the performance of the algorithms, they first introduce a technique to generate input sequences with a known optimum. If the number of colour blocks present in the input sequence is $m$, and all blocks have different colours, then they propose a technique to generate input sequences for which an optimal offline algorithm can achieve $m - 1$ colour changes. Following which, they go on to experimentally evaluate the algorithms for random input sequences against uniform costs. In all cases, their results have been averaged over 50 runs. For random sequences where the colour blocks have constant lengths, their results indicate that the MCF strategy is optimal for buffer sizes greater than 49 and the MAP strategy is optimal for buffer sizes greater than 317. For random sequences where the lengths of the colour blocks are uniformly chosen, their results indicate that the competitive ratios of FIFO, LRU and MCF increase as the buffer sizes increase.
While their experimental results are more extensive than those presented in \[18\], it still suffers from two major drawbacks. Firstly, they only consider random input sequences for their experiments and exclude input sequences that display specific patterns. Secondly, their experiments are limited to uniform costs and no attempts have been made to compare the algorithms for different kinds of cost functions.

In the next chapter we present the details of the algorithms implemented in this thesis along with their pseudocodes and a brief insight into their analysis.
Chapter 3

Algorithms for Reordering Buffer Management

In this chapter, we provide further details of the algorithms implemented for this thesis. In Section 3.1 we define the mathematical notation used throughout the thesis. In Section 3.2 we give the generic methods used by most of the reordering buffer management (RBM) algorithms. In Section 3.3 we present the pseudocode for the different algorithms implemented in this thesis. In Section 3.4 we present our new algorithm, Accelerated Threshold for the RBM Problem.

3.1 Preliminaries

Our model consists of a server and a reordering buffer $B$. The server receives the input sequence $I$ and uses the buffer $B$ to permute it. Our input sequence has $N$ items and we denote the sequence as $I = 1 \ldots N$. Each item in $I$ is characterized by an attribute which is used for reordering. For our model we assume that the attribute to be processed is the colour of the item. In our notation, $C$ denotes the number of colours that can be present in the input sequence. We use the notation $B$ to denote the random access buffer which is a linked list of size $k$ ($k < N$). We use the notation
$i \in I$ to denote the \textit{current input item}, which is the first item in the input sequence that has not yet been processed. The colour of the current input item is denoted by $c(i)$. This item is $i$ is added to the buffer $B$ after which it will be processed. In each step of the algorithm, the current input item $i$ is added to the end of the buffer $B$. The addition of this item triggers the removal of an item from the buffer $B$, and this item is added to the output sequence. The reordering buffer $B$ generates the output sequence by adding the items removed at each step of the RBM algorithm. This sequence is a permutation of the input sequence denoted as $O = \pi\{1 \ldots N\}$. Note that the $B$ is initially empty and must be emptied after the algorithm has terminated. The colour of this item, which is currently assigned to the output sequence is called the \textit{current active colour}. We use the notation $c_{\text{active}}$ to denote the current active colour. The current active colour, $c_{\text{active}}$, changes every time we select a new colour to be assigned to the output sequence.

A continual sub-sequence of items with the same colour in the output sequence is called a \textit{colour block}. A \textit{colour change} occurs when there is a colour switch between two sub-sequences of different colours. As long as we are adding items of the same colour to the output sequence, no cost is incurred. A cost is incurred when we switch from one colour to another. Depending on the RBM algorithm under consideration, this cost may be uniform or non-uniform. In the case of uniform cost algorithms, the same cost is incurred when we switch from one colour to another. While in the case of non-uniform cost algorithms, the cost incurred depends on the colour to which we switch. We use the notation $C_A$ to denote the sum total of the costs incurred during one run of the algorithm, where $A$ denotes the algorithm in consideration.

RBM algorithms can either be offline or online. An offline algorithm is one in which the input sequence is known in advance. That is the choice of the current active colour can be made based on looking at the items in the input sequence. An online algorithm is one in which the input sequence is not known in advance. In
other words, the only items that we can see are the items that have entered the buffer. Each item in the input sequence is revealed only when it enters the buffer. And our selection of the new active colour is based on this piece-wise knowledge of the input sequence.

In real world applications, the input sequence always comes in continually making it almost impossible to know the entire input sequence in advance. Hence real world applications correspond to the online version of the RBM problem. Hence we only consider online algorithms for our experiments.

Online algorithms are evaluated using competitive analysis where every online algorithm is compared against an optimal offline algorithm. Let $C_{OPT}$ denote the cost of an optimal offline algorithm, then an online algorithm is said to be $c$-competitive if it produces a cost which is at most of $c \cdot C_{OPT}$, that is $C_A \leq c \cdot C_{OPT}$.

RBM algorithms are evaluated using lazy strategies. A lazy algorithm satisfies the following two characteristics:

- if there is an item in the buffer with the current output colour, then the lazy algorithm does not make a colour change
- if there are vacant slots in the buffer, then the lazy algorithm does not remove items from the buffer

### 3.2 Basic Structure and Generic Methods

In this section we present the basic structure and the common methods used by most of the reordering buffer management algorithms presented in this thesis. Unless otherwise specified, the algorithms use the structure as presented in this section. The `add` and `remove` methods are also common to most of our algorithms. These are also presented in this section.
The pseudo-code below presents the basic structure for most of our reordering buffer management algorithms:

**Algorithm 1: Basic Reordering Buffer Management Structure**

**Input:** Input stream \( I = 1 \ldots N \)

Buffer \( B \) of size \( k \)

**Output:** Output Stream \( O = \pi\{1 \ldots N\} \)

1. Initialize the buffer, \( B \leftarrow \text{NIL} \)
2. while \( |B| \leq k \) do
3.     add \((i)\); 
4.     Initialize \( c_{\text{active}} \leftarrow -1 \); 
5.     for \( i \leftarrow k + 1 \) to \( N \) do
6.         if \( c_{\text{active}} \notin B \) then
7.             \( c_{\text{active}} \leftarrow \text{ChooseNewActiveColour}(B) \); 
8.             remove \((c_{\text{active}})\); 
9.             add \((i)\); 
10.    while \( |B| \neq \text{NIL} \) do
11.        if \( c_{\text{active}} \notin B \) then
12.           \( c_{\text{active}} \leftarrow \text{ChooseNewActiveColour}(B) \); 
13.           remove \((c_{\text{active}})\); 

The pseudo-codes for the \textit{add}, \textit{remove} and the \textit{ChooseNewActiveColour} methods are as given below. The details of the \textit{ChooseNewActiveColour} method depends on the algorithm in question.

**Function add\((i)\)**

**Input:** \( i \), the input item to be added

1. add \( i \) to the end of the buffer ;

**Function remove\((c_{\text{active}})\)**

**Input:** colour \( c_{\text{active}} \)

1. remove the first item of colour \( c_{\text{active}} \) from the buffer ;
Function ChooseNewActiveColour(B)

Input: Buffer B of size k
Output: New active colour c_active

1 algorithm specific action ;
2 return c_active ;

3.3 Pseudocodes for Implemented Algorithms

This section focuses on the details of all the algorithms implemented in this thesis. We first describe our algorithm, present the pseudocode and give a brief insight into the analysis of the algorithm. The Bounded Waste (BW) Algorithm is described in Section 3.3.1. Random Choice (RC) is described in Section 3.3.2. Round Robin (RR) is described in Section 3.3.3. Next we move on to describing the Non-Uniform cost algorithms starting with Maximum Adjusted Penalty (MAP) in Section 3.3.4 and Threshold or Lowest Cost (TLC) in Section 3.3.5. In Section 3.4 we describe the new algorithm Accelerated Threshold (Acc-T) that we propose in this thesis. Table 1 presents a summary of all the algorithms presented in this thesis.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Cost Function</th>
<th>Competitive Ratio</th>
<th>Thesis Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bounded Waste</td>
<td>Uniform</td>
<td>$O(\log^2 k)$</td>
<td>3.3.1</td>
</tr>
<tr>
<td>Maximum Adjusted Penalty</td>
<td>Non-Uniform</td>
<td>$O(\log k)$</td>
<td>3.3.4</td>
</tr>
<tr>
<td>Random Choice</td>
<td>Uniform</td>
<td>-</td>
<td>3.3.2</td>
</tr>
<tr>
<td>Round Robin</td>
<td>Uniform</td>
<td>-</td>
<td>3.3.3</td>
</tr>
<tr>
<td>Threshold or Lowest Cost</td>
<td>Non-Uniform</td>
<td>$O(\frac{\log k}{\log \log k})$</td>
<td>3.3.5</td>
</tr>
<tr>
<td>Accelerated Threshold</td>
<td>Non-Uniform</td>
<td>-</td>
<td>3.4.2</td>
</tr>
</tbody>
</table>

Table 1: Summary of Algorithms Implemented
3.3.1 Bounded Waste (BW)

Bounded Waste (BW) [1], was the first proposed algorithm for the RBM problem. In this algorithm, items are input into the buffer for the first $k$ steps. Following this, the algorithm has to choose a new active colour to be evicted from the buffer and assigned to the output sequence. BW uses the concept of penalty to choose a new active colour; penalty is defined as the number of occurrences of a particular colour $c$ in the buffer ($B$) at that step. All colours in the buffer are assigned penalties based on the number of occurrences of items of that colour present in the buffer. Initially the penalties of all colours are zero. When a new active colour is to be chosen, the algorithm assigns penalties to each colour and the colour with the maximum penalty is chosen to be the new active colour. All items of this colour are evicted and assigned to the output sequence in the same step. We use the notation $P_c$ to denote the penalty assigned to colour $c$ and $c_{\text{max}}(P_c)$ to denote the colour with the maximum penalty. The pseudocode for the algorithm is as presented in Algorithm 2. BW uses the same add method as listed in Section 1. However, the remove method differs from the basic method in that all items of the current active colour $c_{\text{active}}$ are evicted in the same step as opposed to one item per step.

Räcke et al. [1] show that strategies like First in First Out (FIFO), Least Recently Used (LRU) and Largest Colour First (LCF) are unsuitable for the RBM problem as they evict items too early (FIFO and LRU) or store items indefinitely in the buffer (LCF). The prove that the competitive ratio of FIFO and LRU is $\Omega(\sqrt{k})$, while LCF has a competitive ratio of $\Omega(k)$.

BW is evaluated using competitive analysis. As a first step, Räcke et al. [1] assume that an optimal offline algorithm and the input sequence $I$ are fixed. Hence the number of colour changes made by BW and the optimal offline strategy are also fixed.
**Algorithm 2: Bounded Waste (BW)**

**Input**: Input stream $I = 1 \ldots N$

Buffer $B$ of size $k$

**Output**: Output Stream $O = \pi \{1 \ldots N\}$

1. Initialize the buffer, $B \leftarrow \text{NIL}$;
2. while $|B| \leq k$ do
   3. add $(i)$;
   4. Initialize $c_{\text{active}} \leftarrow -1$;
   5. while $|B| \leq k$ do
      6. if $c_{\text{active}} \notin B$ then
         7. $c_{\text{active}} \leftarrow \text{BWChooseNewActiveColour}(B)$;
         8. $\text{BWremove}(c_{\text{active}})$;
         9. add $(i)$;
   10. while $|B| \neq \text{NIL}$ do
      11. if $c_{\text{active}} \notin B$ then
         12. $c_{\text{active}} \leftarrow \text{BWChooseNewActiveColour}(B)$;
         13. $\text{BWremove}(c_{\text{active}})$;

**Function BWremove($c_{\text{active}}$)**

**Input**: colour $c_{\text{active}}$

1. remove all the items of colour $c_{\text{active}}$ from the buffer;

**Function BWChooseNewActiveColour($B$)**

**Input**: Buffer $B$ of size $k$

Penalty $P_c$ for each item in $B$

**Output**: New active colour $c_{\text{active}}$

1. foreach colour $c$ in the buffer do
   2. $P_c \leftarrow$ the number of occurrences of colour $c$
   3. $c_{\text{active}} \leftarrow c_{\text{max}}(P_c)$;
   4. $P_{c_{\text{active}}} \leftarrow 0$;
   5. return $c_{\text{active}}$;

They compare the performance of the fixed optimal offline strategy and BW by introducing the notion of waste $w$. At any colour change $c$, the waste $w$ is defined as the number of items of colour $c$ present in the buffer at that particular step. If
we use the notation $W_{OPT}^c$ and $W_{BW}^c$ to denote the waste for colour $c$ for the optimal offline and BW algorithms, and the notation $C_{OPT}$ and $C_{BW}$ to denote the number of colour blocks in the output sequence in the optimal offline and BW algorithms, then $\sum_{\text{colour}=c} W_{OPT}^c \leq k \cdot C_{BW}$ and $\sum_{\text{colour}=c} W_{BW}^c \geq k \cdot C_{BW} - k^2$. These inequalities hold since both algorithms follow the lazy strategy described in Section 3.1. Then the difference between the waste for colour $c$ is defined as $\Delta^c = W_{OPT}^c - W_{BW}^c$. They use this and prove that $\sum_{\text{colour}=c} \Delta^c \leq k^2$.

Their main lemma is to prove that $\Delta^c \geq W_{BW}^c - C_{OPT}^c \times O(k \log^2 k)$. The proof of this lemma involves proving that for any colour $c$, the penalty $P_c$ is at most $O(k \log k)$, which is in turn proved by using a potential function for all colours present the buffer at that particular time. This is referred to as the “Bounded Waste” lemma. Finally, substituting the value of $\Delta^c$ in the inequality $\sum_{\text{colour}=c} \Delta^c \leq k^2$ gives a competitive ratio of $O(\log^2 k)$ for the BW algorithm. The reader is referred to [1] for a detailed analysis of their proof.

3.3.2 Random Choice (RC)

The Random Choice algorithm (RC) [19], is a simple strategy that randomly selects a colour from the buffer as the new active colour. Items are added into the buffer for the first $k$ steps, until the buffer is full. When the buffer is full, the RC strategy selects the new active colour randomly with equal probability from the different colours that are currently present in the buffer. The RC strategy can also be seen as a randomized variation of the BW strategy. The pseudocode for the RC algorithm is as presented in Algorithm 3. The RC algorithm uses the same add and remove methods described in Section 3.2.

There have only been experimental evaluations of Random Choice [19]. As far as we are aware, no attempts have been made to analyze the theoretical efficiency or prove the competitive ratio of this algorithm.
Algorithm 3: Random Choice (RC)

**Input:** Input stream $I = 1 \ldots N$
Buffer $B$ of size $k$

**Output:** Output Stream $O = \pi\{1 \ldots N\}$

1. Initialize the buffer, $B \leftarrow \text{NIL}$;
2. while $|B| \leq k$ do
3.   add $(i)$;
4.   Initialize $c_{\text{active}} \leftarrow -1$;
5.   for $i \leftarrow k + 1$ to $N$ do
6.     if $c_{\text{active}} \notin B$ then
7.       $c_{\text{active}} \leftarrow \text{RCChooseNewActiveColour}(B)$;
8.       remove $(c_{\text{active}})$;
9.       add $(i)$;
10.  while $|B| \neq \text{NIL}$ do
11.    if $c_{\text{active}} \notin B$ then
12.       $c_{\text{active}} \leftarrow \text{RCChooseNewActiveColour}(B)$;
13.       remove $(c_{\text{active}})$;

**Function** RCChooseNewActiveColour($B$)

**Input:** Buffer $B$ of size $k$

**Output:** New active colour $c_{\text{active}}$

1. $c_{\text{active}} \leftarrow$ randomly select a colour from the colours present in the buffer ;
2. return $c_{\text{active}}$ ;

3.3.3 Round Robin (RR)

The Round Robin (RR) algorithm [19], uses a selection pointer to select the new active colour. As in the case of RC, we fill the buffer for the first $k$ steps. We initialize our selection pointer to point to the first item in the buffer. When a new active colour is to be chosen, it returns the colour pointed to by the selection pointer and then advances the selection pointer one step right to the next item in the buffer. All items of this colour are removed until there are no more items of this colour in the buffer. When the selection pointer reaches the end of the buffer, the selection pointer is reset to point to the first item in the buffer. We use the notation $p$ to denote the
selection pointer and the item pointed to by the selection pointer is denoted by \(i_p\).

The pseudo-code for the RR strategy is presented in Algorithm 4. The RR algorithm uses the same \texttt{add} and \texttt{remove} methods described in Section 3.2.

**Algorithm 4: Round Robin (RR)**

**Input:** Input stream \(I = 1 \ldots N\)
- Buffer \(B\) of size \(k\)
- Selection pointer \(p\)

**Output:** Output Stream \(O = \pi\{1 \ldots N\}\)

1. Initialize the buffer, \(B \leftarrow \text{NIL}\); 
2. Initialize the selection pointer, \(p \leftarrow 1\); 
3. while \(|B| \leq k\) do 
   4. \texttt{add}(i); 
   5. Initialize \(c_{active} \leftarrow -1\); 
   6. for \(i \leftarrow k + 1\) to \(N\) do 
      7. if \(c_{active} \notin B\) then 
         8. \(c_{active} \leftarrow \text{RRChooseNewActiveColour}(B)\); 
         9. \texttt{remove}(c_{active}); 
      10. \texttt{add}(i); 
6. \textbf{end while} 

11. while \(|B| \neq \text{NIL}\) do 
12. if \(c_{active} \notin B\) then 
13. \(c_{active} \leftarrow \text{RRChooseNewActiveColour}(B)\); 
14. \texttt{remove}(c_{active}); 
15. \textbf{end while}

**Function ChooseNewActiveColour(B)**

**Input:** Buffer \(B\) of size \(k\)
- selection pointer \(p\)

**Output:** New active colour \(c_{active}\)

1. \(c_{active} \leftarrow c(i_p)\) ; 
2. \(p \leftarrow (p + 1) \mod k\) ; 
3. \textbf{return} \(c_{active}\); 

Round Robin has only been evaluated experimentally [19], and to the best of our knowledge, there have been no theoretical analysis proving the competitive ratio of this algorithm.
3.3.4 Maximum Adjusted Penalty (MAP)

The Maximum Adjusted Penalty (MAP) algorithm [4], improvises on the Bounded Waste algorithm, to achieve a better competitive ratio of $O(\log k)$. It is designed to work for scenarios where the cost is non-uniform, where each colour is associated with a cost of $w(c)$.

Like with BW, MAP fills the buffer for the first $k$ steps and then chooses a colour to be evicted from the buffer. Once a colour has been chosen, it continues to remove all items of that colour until there are no more items of that colour in the buffer. As with BW, in order to choose a new active colour, MAP uses the notion of penalty. Each colour is assigned a penalty and MAP compares these penalties and costs of the colour to choose a new active colour. We use the notation $P_c$ to denote the penalty of colour $c$. When MAP has to select a new active colour, it chooses a colour $c$ such that $P_c - k \cdot w(c) \geq P_{c'} - k \cdot w(c')$ for every other colour $c'$. After returning $c$ to be the new active colour, $c_{active}$, the penalty $P_{c_{active}}$ is reset to zero and the penalties of all other colours in the buffer are increased by $n(c) \cdot w(c)$, where $n(c)$ denotes the number of items of colour $c$ in the buffer. The pseudocode for the MAP algorithm is presented in algorithm [5]

Englert and Westermann [4], prove that with a buffer size of $k$, we can achieve a reduction in cost by at most $2k - 1$. This basic upper bound holds good for both online and offline algorithms, across all cost functions. In addition to that, they also prove a competitive ratio of $k$ for the BW [1] for non-uniform costs.

Their analysis of the MAP algorithm consists of two parts. Firstly, they prove that with a buffer of size $k$, MAP is 4 - competitive against an optimal offline algorithm with a buffer of size $\frac{k}{4}$. From this, MAP has a performance that is 4 times worse than that achieved by an optimal offline algorithm with buffer size $\frac{k}{4}$. Secondly, they prove that an optimal offline algorithm with a buffer of size $\frac{k}{4}$ is $O(\log k)$ -
Algorithm 5: Maximum Adjusted Penalty (MAP)

Input: Input stream $I = 1 \ldots N$
Buffer $B$ of size $k$

Output: Output Stream $O = \pi \{1 \ldots N\}$

1 Initialize the buffer, $B \leftarrow \text{NIL}$
2 while $|B| \leq k$ do
3   add $(i)$
4   Initialize $c_{\text{active}} \leftarrow -1$
5   for $i \leftarrow k + 1$ to $N$ do
6     if $c_{\text{active}} \notin B$ then
7       $c_{\text{active}} \leftarrow \text{MAPChooseNewActiveColour}(B)$
8       remove $(c_{\text{active}})$
9       add $(i)$
10    while $|B| \neq \text{NIL}$ do
11       if $c_{\text{active}} \notin B$ then
12          $c_{\text{active}} \leftarrow \text{MAPChooseNewActiveColour}(B)$
13          remove $(c_{\text{active}})$
14    endwhile
15 endwhile

Function MAPChooseNewActiveColour($B$)

Input: Buffer $B$ of size $k$
Penalty $P_c$ for each colour $c$

Output: New active colour $c_{\text{active}}$

1 foreach colour $c'$ in $B$ do
2   foreach colour $c$ in $B$ do
3     if $P_c - k \cdot w(c) \geq P_{c'} - k \cdot w(c')$ then
4       $c_{\text{active}} \leftarrow c$
5       $P_c \leftarrow 0$
6   endforeach
7 endforeach
8 return $c_{\text{active}}$

competitive against an optimal offline algorithm with a buffer of size $k$. From this, the performance of the optimal offline algorithm with buffer size $\frac{b}{4}$ is $O(\log k)$ times worse than the performance achieved by the optimal offline algorithm with buffer size $k$. Hence from these two statements we can conclude that with a buffer of size
MAP has a performance that is $4 \cdot O(\log k)$ times worse than the performance achieved by an optimal offline algorithm with the same buffer size, which is the same as saying that MAP has a competitive ratio of $O(\log k)$ when compared against an optimal offline algorithm. The reader is referred to [4] for details of this proof.

### 3.3.5 Threshold or Lowest Cost (TLC)

The Threshold or Lowest Cost (TLC) [5], algorithm evicts one item at each step of the algorithm. For the first $k$ steps the input items are stored in the reordering buffer, and the elimination happens one element at each step from steps $k + 1, k + 2, \ldots, k + N$. The algorithm maintains a counter for each input item in the buffer. We use the notation $\phi_i$ to denote the counter for item $i$. Initially the counters for all items in the buffer are zero, and the values these counters are non-decreasing during the entire run of the algorithm. The counters are updated when we change the active colour to a colour with the lowest cost in the buffer. We use the notation $c_{\text{min}}$ to represent the colour with the lowest cost and the cost associated with that colour is denoted by $w(c_{\text{min}})$.

At each colour change, the algorithm selects a new active colour and removes one item of this colour from the buffer at each step until the buffer has no more items of this colour. At this point the algorithm selects a new active colour, and this is done as follows: if there is a colour $c$ in the buffer such that $\sum_{c(i) = c} \phi_i \geq w(c)$, then choose any such colour $c$ to be the new active colour. If there is no such colour $c$, then select the colour $c$ with the lowest cost $w(c_{\text{min}})$ and update the counters as follows: add the quantity $\frac{w(c_{\text{min}})}{k}$ to all the items in the buffer. The algorithm derives its name based on how the active colour is selected. At each active colour selection, we compare the values of the sum of values of the counters and check if it has exceeded the cost of the colour, i.e., $\sum_{c(i) = c} \phi_i \geq w(c)$. This inequality is referred to as the “threshold” for that colour. If such a colour is not present, then as described earlier we return the
colour with the lowest cost. Hence the algorithm has the name “Threshold or Lowest Cost”.

To make our implementation efficient, we start by initializing the new active colour $c_{active}$ to the colour with the lowest cost. So that in the end, if no colour had reached its threshold, the colour that will be returned will always be the colour with the lowest cost. The pseudocode for the TLC algorithm is presented in Algorithm 6. The TLC algorithm uses the same add and remove methods described in Section 3.2.

**Algorithm 6: TLC**

**Input:** Input stream $I = 1 \ldots N$
- Buffer $B$ of size $k$

**Output:** Output Stream $O = \pi\{1 \ldots N\}$

1. Initialize the buffer, $B \leftarrow NIL$
2. while $|B| \leq k$ do
   3. add $(i)$;
   4. Initialize $c_{active} \leftarrow -1$
   5. for $i \leftarrow k + 1$ to $N$ do
      6. if $c_{active} \notin B$ then
         7. $c_{active} \leftarrow $ TLCChooseNewActiveColour($B$);
      8. remove ($c_{active}$);
      9. add $(i)$;
   10. while $|B| \neq NIL$ do
       11. if $c_{active} \notin B$ then
          12. $c_{active} \leftarrow $ TLCChooseNewActiveColour($B$);
       13. remove ($c_{active}$);

The analysis for TLC follows an intricate linear programming relaxation scheme to achieve a competitive ratio of $O\left(\frac{\log k}{\log \log k}\right)$. In their analysis, Rabani and Elgrabli first prove two facts, one which proves an upper bound on the values of the counters the items can have in the buffer and the second fact which upper bounds the cost of TLC based on these counter values.

The first fact states that for a set $S^c$ of items of colour $c$ that are in the buffer at the same time, the sum of their counter values is at most two times the cost associated
**Function** TLCChooseNewActiveColour(B)

**Input:** Buffer $B$ of size $k$

**Output:** New active colour $c_{\text{active}}$

1. for every item $i \in B$ do
2. \[ S^c = \sum_{c(i)=c} \phi_i ; \]
3. \[ c_{\text{active}} \leftarrow c_{\text{min}} ; \]
4. for every item $i \in B$ do
5. \[ \sigma_c = S^c ; \]
6. if $\sigma_c \geq w(c)$ then
7. \[ c_{\text{active}} \leftarrow c ; \]
8. return $c_{\text{active}} ;$
9. add $\frac{w(c_{\text{min}})}{k}$ to the counter $\phi_i$ of every item in the buffer ;
10. return $c_{\text{active}} ;$

with that colour, $\sum_{i \in S^c} \phi_i < 2 \times w(c)$. If we use the notation $C_{\text{TLC}}$ to denote the cost incurred by TLC at the end of reordering, then the second fact states that the cost incurred by TLC is at most two times the sum of the counters of items in the buffer, i.e, $C_{\text{TLC}} \leq 2 \times \sum_{i=1}^{N} \phi_i$.

They then formulate an Integer Linear Program (ILP) for the offline variant of the Reordering Buffer Management problem and prove that the cost of the ILP is the same as that incurred by the optimal offline algorithm ($C_{\text{OPT}}$). They then propose the dual program for the relaxation of the ILP and use that to prove a competitive ratio of $O\left(\frac{\log k}{\log \log k}\right)$ for TLC. The reader is encouraged to refer to [5] for the details of this proof.

### 3.4 Accelerated Threshold (Acc-T)

In this section, we present our new algorithm, Accelerated Threshold, which is a variation of the Threshold or Lowest Cost algorithm presented in Section 3.3.5. In Section 3.4.1 we give the mathematical notation used for our algorithm and in Section
3.4.2 we give our algorithm and highlight its differences with TLC.

3.4.1 Preliminaries

To stay consistent with other algorithms, let $I = 1, 2, \ldots, N$ denote the input sequence, where $N$ is the size of the sequence. Let $B$ represent the reordering buffer and $k$ denote the buffer size. Our model uses $C = 1, 2, \ldots, c$ to represent the colours that can be present in the input sequence; the weight/cost for colour $c$ is represented by $w(c)$. Let $c_{\text{min}}$ denote the colour having the lowest cost. Let $O = \pi\{1 \ldots N\}$ represent our output sequence, which is a permutation of the input sequence $I$.

During the execution of the algorithm, $i \in I$ denotes the current input item, which is the first item of the input sequence that has not yet been processed. The colour of this item is denoted as $c(i)$, s.t. $c(i) \in C$. An item must be added to the buffer $B$ before it can be output. The colour of the item that was last assigned to the output sequence is called the current active colour, denoted by $c_{\text{active}}$. Since RBM algorithms use lazy strategies, we change the active colour only when we no longer have any item of the colour $c_{\text{active}}$ in the buffer.

Like with TLC, our algorithm maintains and updates a counter which we denote as $\Phi_i$, the counter for item $i$. This counter is non-decreasing in every step of the algorithm. On every step that requires a new active colour to be chosen, our algorithm inspects the counter values of each item in the buffer by colour and then selects a new active colour as described in Section 3.4.2.

3.4.2 Pseudocode for Accelerated Threshold

Our algorithm works as follows. For the first $k$ steps, we add one item per step to the buffer, and do not evict anything, to stay inline with the lazy strategy. Following this, the eviction process begins and we start evicting items from steps $k+1, k+2, \ldots, k+N$. 

Since we only evict one item per step of the algorithm, our buffer is emptied at step \( k + N \).

For all items in the buffer, the counter \( \Phi_i \) is set to 0 initially. On each iteration of the algorithm, we manipulate the values of these counters. The values of these counters play a crucial role in the selection of the new active colour.

Whenever a new active colour is to be selected, we increment the values of these counters in \( k \) sub-iterations. Let \( b \) denote the item that is currently being examined. At the end of each sub-iteration, we advance \( b \) to denote the next item in the buffer. We start by examining the counter values for the first item in the buffer; \( b \) now corresponds to the first item. Whenever we have to select a new active colour, we check if the sum of counters of all items of colour \( c(b) \) is greater than the cost associated with that colour, \( w(c(b)) \), i.e., \( \sum_{c(i)=c(b)} \Phi_i \geq w(c(b)) \). As in the case of TLC, we refer to this inequality as the “threshold” of that colour. If such a colour \( c(b) \) exists, we select colour \( c(b) \) to be the new active colour, \( c_{active} \), and evict one item of that colour. If the buffer has no such colour on the first sub-iteration, we increment the counters of all items in the buffer with the value \( \frac{w(c_{min})}{k} \) and advance \( b \) to be the next item in the buffer, moving on to the next sub-iteration. At this point, we do the “threshold” check again for the colour \( c(b) \), and increment the counter values if the item \( b \)'s counter value has not reached its threshold. We continue until some item \( b \) has reached its threshold or until we reach the end of the buffer at the end of \( k \) sub-iterations.

It is to be noted that at the end of \( k \) sub-iterations, regardless of any other colour \( c \) reaching its threshold, the colour \( c_{min} \) will definitely reach its threshold. If we have reached the end of the buffer, we have updated the values of all counters by \( \frac{w(c_{min})}{k} \times k \) which is the same as \( w(c_{min}) \) itself. Hence the threshold for \( c_{min} \) is always reached if we have completed \( k \) sub-iterations. Based on this, we return \( c_{min} \) as the new active colour if we have reached the end of \( k \) sub-iterations.

To do these \( k \) updates to the counters efficiently, we store the sums of these
counters in two arrays of size $C$. One each for the sum and count of the items of a particular colour $c$. We track the number of times for which our counters need to be updated (with the supplementary counter $j$) before the active colour is chosen. Using these aggregated values, we perform one update at the end of our active colour selection process making our algorithm time efficient. Doing this prevents us looping over the buffer to compute values of the sums, thereby reducing our running time to $O(k)$.

There are two main difference between TLC and our algorithm. Firstly, we update the values of the counters $\Phi_i$ of the items in the buffer in $k$ sub-iterations during one active colour selection, whereas TLC does only one such update per active colour selection. In other words, our counter values can be increased by up to $\frac{w(c_{\min})}{k} \times k$ during one new active colour selection, whereas TLC’s counter values can only increase by $\frac{w(c_{\min})}{k}$.

Secondly, TLC needs to return a default colour when it fails to find a colour that has reached its threshold. In this case, it returns the colour with the lowest cost, $c_{\min}$. The increased updates to counter values in Acc-T eliminates the need for a default colour, as we are assured that at least one colour has reached its threshold by the end of $k$ sub-iterations.

The pseudocode for our algorithm is presented in Algorithm 7. Our algorithm uses the same $\text{add}$ and $\text{remove}$ methods described in Section 3.2.

The analysis of TLC [5] uses a fairly elaborate linear programming relaxation scheme. So far, we have not been able to analyze Acc-T using other techniques, but as our algorithm is computationally very similar, it might be possible to use a modification of TLC’s LP scheme. Such an analysis is beyond the scope of this thesis.

In the next chapter, we analyze our algorithm experimentally and compare it with other algorithms implemented in this thesis.
Algorithm 7: Acc-T

Input: Input stream $I = 1 \ldots N$
Buffer $B$ of size $k$

Output: Output Stream $O = \pi \{1 \ldots N\}$

1. Initialize the buffer, $B \leftarrow \text{NIL}$
2. while $|B| \leq k$ do
3.   add $(i)$
4.   Initialize $c_{\text{active}} \leftarrow -1$
5.   for $i \leftarrow k+1$ to $N$ do
6.     if $c_{\text{active}} \notin B$ then
7.       $c_{\text{active}} \leftarrow \text{AccTActiveColour}(B)$
8.       remove ($c_{\text{active}}$)
9.       add $(i)$
10.  while $|B| \neq \text{NIL}$ do
11.    if $c_{\text{active}} \notin B$ then
12.       $c_{\text{active}} \leftarrow \text{AccTActiveColour}(B)$
13.       remove ($c_{\text{active}}$)

Function AccTActiveColour($B$)

Input: Buffer $B$ of size $k$
Output: New active colour $c_{\text{active}}$

1. for every item $b \in B$ do
2.   $S_c(b) = \sum_{c(i)=c(b)} \Phi_i$
3.   $C_c(b) = \text{number of items of colour } c(b) \text{ in } B$
4.   $c_{\text{active}} \leftarrow c_{\text{min}}$
5.   $j = 0$
6.   for every item $b \in B$ do
7.     $\sigma_c(b) = S_c(b) + (C_c(b) \times \frac{w(c_{\text{min}})}{k} \times j)$
8.     if $\sigma_c(b) \geq w(c(b))$ then
9.       $c_{\text{active}} \leftarrow c$
10.      break
11.    $j \leftarrow j + 1$
12.   add $\frac{w(c_{\text{min}})}{k} \times j$ to the counter $\Phi_i$ of every item in the buffer
13. return $c_{\text{active}}$
Chapter 4

Implementation and Experiments

In this chapter, we describe the implementation details of all the algorithms implemented in this thesis along with the experimental results obtained. In Section 4.1 we give detailed descriptions of all the data sets we have generated. Section 4.2 describes all the cost functions we have used for our experiments. Section 4.3 gives specific details about the programming language and underlying data structures used for our implementations. We describe our experimental set up in Section 4.4. Section 4.5 describes the impact of input sequence size on the algorithms’ performance. Section 4.6 presents our experimental results for the uniform cost algorithms and Section 4.7 presents our experimental results for the non-uniform cost algorithms. Finally, in Section 4.8 we compare our results to previously obtained experimental results for the RBM problem.

4.1 Data Sets

We generate input sequences of sizes 100, 1000, 10,000 and 100,000. Each input item in the sequence is assigned a colour within the range $1 \ldots C$. In order to recommend algorithms for various application scenarios we generated data sets that follow specific patterns as well as random data sets. We have performed experiments on the following
data sets for all our algorithms for different size combinations of the input parameters.

- **Alternation Sequence**: For the alternation sequence, we start by assigning the colour 1 to the first input item. From the second item, we successively assign colours in increments of 1 upto $C$ and repeat the pattern until the end of the input sequence. An example of the sequence for 3 colours and input sequence size 10 is as follows: 1, 2, 3, 1, 2, 3, 1, 2, 3, 1.

- **Delta Sequence**: In the delta sequence we generate input items that are always within a certain range. We use the notation $\Delta$ to denote this range. For this sequence, we generate the first item $c$ randomly with equal probability within the range $1 \ldots C$. Starting with the second item, we generate each item in the range of $c \pm \Delta$, where $\Delta$ can take the value 0. This ensures that all the items are within the range of $\Delta$ from the previous item. An example of the sequence for $\Delta$ value 2, 5 colours and input sequence size 10 is as follows: 2, 4, 5, 3, 1, 1, 3, 5, 4, 3.

- **Random Sequence**: The colours for the input items are generated randomly between the range $1 \ldots C$ with equal probability. The following is an example of a random sequence with 3 colours and input size 10 as follows: 3, 2, 3, 1, 2, 3, 2, 1, 1, 2.

- **Random Block Sequence**: In this case, we have a block sequence where the size of the blocks are randomly chosen with equal probability within the range $1 \ldots C$ and this size also corresponds to the colour assigned to each item in that particular block. An example of this sequence for input size 10 and 3 colours is as follows: 3, 3, 3, 1, 2, 2, 1, 1, 2, 2. For this example, we first generate the size 3 which also becomes the colour of the items in that block, so the first three items have colour 3. Then we generate the block size 1 which has the colour
1. Next we generate the block size 2 and the next two items have colour 2, we then generate two blocks each with size 1 and colour 1. Finally we generate the block size 2 which gets the colour 2 for the last block.

- **Sequential Block Sequence**: We use the notation $m$ to denote the block size. We assign a colour $c$, chosen from the range $1 \ldots C$ to all input items belonging to the same block. We have assigned colours to each block, starting from the first block, in an increasing order from 1 going up to $C$. We repeat the same pattern of colours for the entire input sequence. An example of the sequence for $m = 2$, 3 colours and input sequence size 10 is as follows: 1, 1, 2, 2, 3, 3, 1, 1, 2, 2.

### 4.2 Cost Functions

Since some of our algorithms are designed to handle non-uniform costs, we have designed the following cost functions and tested them with all the data sets listed in Section 4.1. Our cost functions are as defined below:

- **Cost-Equals-Colour cost function**: For this cost function, the cost assigned to the colour is the value of the colour itself, for example, an item with colour 3 has a cost of 3.

- **Cost-Equals-Quadratic-Colour cost function**: In this case, the cost assigned to the colour is equal to the square of the value of the colour. For example, the cost of colour 3 is set to be 9. The reason behind having such a cost function is to examine our algorithms’ performance when subject to input sequences where the cost of switching is very high and varies between very low and very high costs.

- **Random-Cost cost function**: In this case we randomly assign a cost to each colour between the range $1 \ldots R$. For our experiments we have set $R$ to be
5. Note that, with the Cost-Equals-Colour cost function and the Cost-Equals-Quadratic-Colour cost function, the cost range of costs that an item can have can be very large depending on the number of colours. We have used the Random-Cost cost function to compare our algorithms’ performance when the range of costs relatively small.

• **Colour-Difference cost function**: This is the only cost function that takes into account the colour that we are presently at, and the colour to which we would like to switch to. In other words, it is our only variable cost function. We call these two colours the *from colour* and *to colour*. Here the cost of switching from the *from colour* to the *to colour* is defined as the absolute difference between the two colours. This cost function is especially applicable in scenarios where the cost of switching depends on the two colours in question. An application of this cost function can be seen in the disk scheduling problem where switching is synonymous to moving the disk head to point to a particular disk block. In this case we want to minimize the distance between moving the head too far from the current point.

• **Uniform-Cost cost function**: In this case the cost assigned to every colour is the uniform. In other words, the same cost is occurred when we switch from one colour to the next.

### 4.3 Implementation Details

In this section we describe the details about our implementations and give a basic overview of our approach to the different algorithms we have implemented. Table 2 gives an overview of all the input sequence sizes, buffer sizes and number of colours used for our experiments. We have performed experiments with different data sets
and cost functions as described in Section 4.1 and Section 4.2.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Sizes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Sequence (N)</td>
<td>100 1,000 10,000 100,000</td>
</tr>
<tr>
<td>Buffer (k)</td>
<td>2 5 10 20 50 100 500</td>
</tr>
<tr>
<td>Num Colours (C)</td>
<td>2 5 10 20 50</td>
</tr>
</tbody>
</table>

**Table 2:** Summary of input parameters and their values for our experiments

We have used Java with the JavaSE-1.7 as our execution environment for all of our implementations. Our input sequences, output sequences and buffers are implemented as Java ArrayLists. We build our input sequences according to the data sets listed in 4.1 by adding an item at the end of the list representing the input sequence. Our algorithms also construct the output sequence in a similar manner by appending items to the end of the list representing the output sequence. Array lists allow for removing an item from the middle and hence our choice of data structure to model random access buffers.

Every item is designed to have the colour, the input time which we set when the input item is first created, the output time which specifies the time at which the item was evicted from the buffer and an optional counter which is specific to the algorithm. We begin our input time stamps from 1 and check to ensure that every item has an output time that is greater than the time at which it was input.

We define all the statistics that we would like to measure about our algorithms’ performance on the sequences, and compare the input and output sequences to decide the proportion of switches that have been reduced, the proportion of switching cost that has been reduced, the minimum, maximum and the average time that an item spends in the buffer.

Our basic outline for the reordering buffers happens in three stages: initialization, reordering and eviction. In the initialization phase, we fill the first k items of the
input sequence into the buffer. In the reordering phase we first pick a new colour
to evict from the different colours presently in the buffer, which we call as the new
active colour and evict one item of this colour in each time step. We then set the
output time on the evicted item, add the item to the output sequence and refill the
buffer. We continue to do this until all the input items been processed by the buffer.
In the eviction phase, we empty the buffer and start evicting one item at each step
until the buffer is empty. We also set output times and add the item to the output
sequence as in the reordering phase.

Except for the Bounded Waste algorithm described in Section 3.3.1, all our al-
gorithms have the same basic reordering outline as described above, where one item
is evicted in each time step making room for one item from the input sequence to
enter the buffer. The difference with Bounded Waste is that it evicts all items of the
chosen active colour in the same step, thereby making room for \( m \) \((m \leq k)\) items to
enter the buffer in each time step.

4.4 Experimental Setup

Section 4.3 describes the details of our implementation, our assumptions for the algo-
rithms and language and programming details. We have performed different exper-
iments on these algorithms for the all the data sets listed in Section 4.1 and all the
cost functions listed in Section 4.2 to compare algorithms and recommend different
algorithms for the various combinations of input sequence types and cost functions.
These results are listed in the following sections.

Experiments are performed once for each deterministic algorithm. For randomized
algorithms, random data sets and random cost functions, we repeat the experiments
100 times and average the results from all experiments.1

1We have used Excel 2010 to analyze all our results and used Excel Pivot Tables to compute
averages and plot charts for our experiments.
Table 3 lists all the input parameters we record during our experiments and the output parameters that we compute using the recorded input parameters.

<table>
<thead>
<tr>
<th>Type of Parameter</th>
<th>Values Recorded</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input parameters</td>
<td>The Algorithm being executed</td>
</tr>
<tr>
<td></td>
<td>Kind of Input Sequence Used</td>
</tr>
<tr>
<td></td>
<td>Cost Function</td>
</tr>
<tr>
<td></td>
<td>Input Sequence Size</td>
</tr>
<tr>
<td></td>
<td>Buffer Size</td>
</tr>
<tr>
<td></td>
<td>Number of Colours</td>
</tr>
<tr>
<td></td>
<td>Count of Switches in the Input Sequence</td>
</tr>
<tr>
<td></td>
<td>Count of Switches in the Output Sequence</td>
</tr>
<tr>
<td></td>
<td>Input Sequence Cost</td>
</tr>
<tr>
<td></td>
<td>Output Sequence Cost</td>
</tr>
<tr>
<td>Output parameters</td>
<td>Output/Input Switch Ratio</td>
</tr>
<tr>
<td></td>
<td>Output/Input Cost Ratio</td>
</tr>
</tbody>
</table>

Table 3: Input and Output Parameters used for our experiments

We define a switch as a colour change that occurs between one item and the next item for both the input and output sequences. Hence the Count of Switches in the input and output sequences are the total number of colour changes between the items in that sequence.

Whenever a switch occurs in the input or output sequence, we incur a cost associated with the colour that we have switched to. We use the term Input Sequence Cost to denote the sum total of all the costs that are incurred while switching to different colours in the input sequence. Likewise, the term Output Sequence Cost denotes the sum total of all the costs incurred while switching to different colours in the output sequence.

In the rest of this thesis, we refer to the Output/Input Switch Ratio as just the
“switch” ratio and the Output/Input Cost Ratio as just the “cost” ratio to enhance readability.

We use these ratios to plot our graphs since it gives us an insight into how each algorithm performs under different data sets and cost function combinations. The ratios give us some insight into the percentage of how many switches were eliminated or how much of the cost was reduced. For example, let us assume that our input sequence has a size of 10, and has 8 switches in the input and 3 switches in the output, our switch ratio is $3/8 = 0.375$. This indicates that the output sequence has 37.5% of the switches present in the input sequence. We use similar ratios to analyze the cost as well. This indicates that it is desirable to achieve low switch and cost ratios for the algorithms.

In the next subsections, we describe our results as observed for different input sequence sizes, for uniform cost and for non-uniform costs across all the input sequence types and cost function combinations.

### 4.5 Impact of Input Sequence Size

Our experimental results reveal that the input sequence size does not play a significant role in the way our algorithms perform with different buffer sizes, varied number of colours or even different data sets and cost functions. This result is illustrated in Figure 2 where we can observe that our algorithms achieve almost identical ratios for all input sequence sizes.

From Figure 2 we can see that while different algorithms have different performance for different combinations of data sets and cost functions, the general trend suggests that they do not heavily depend on the size of the input sequence. We observe that for buffer size $k = 5$, all our algorithms have comparable switch ratios which are around 0.73 across all input sequence sizes. The same trend is observed for
other buffer sizes as illustrated in Figure 2. This has also been true in the case of all kinds of data sets when subjected to all different cost functions as well.

Hence we conclude that the input sequence size is not a significant factor when comparing algorithms’ performance and only use buffer sizes and number of colours to compare our algorithms’ performance henceforth in this thesis.

Figure 2: The outer values on the x-axis correspond to the input sequence sizes and the inner values on the x-axis correspond to the different buffer sizes. The values on the y-axis correspond to the output/input switch ratio achieved by the algorithms. From this figure we can conclude that the Input Sequence Size does not impact the performance of algorithms.
4.6 Experimental Results for Uniform Cost

In this section we present the uniform cost algorithms’ performance for different input sequence types. In the case of uniform cost algorithms, switch and cost ratios are identical since the same cost is incurred when we switch from one colour to the next. We present our results based on the switch ratios of the algorithms.

We have implemented the following uniform cost algorithms. See Chapter 3 for details.

- Bounded Waste (BW)
- Random Choice (RC)
- Round Robin (RR)

In the following subsections, we present the experimental results for these algorithms for different input sequence types and while varying the combinations of buffer sizes and number of colours. In each subsection, we first describe the results observed for that sequence type and present a figure to verify our results. In all our figures, blue represents the BW algorithm, red represents the RC algorithm, and green represents the RR algorithm. We refer to the algorithms using their abbreviations to enhance readability.

In Section 4.6.1 we compare these algorithms against Alternation Sequences. Section 4.6.2 compares the algorithms against Delta Sequences. Section 4.6.3 compares the algorithms against Random Sequences. Sections 4.6.4 and 4.6.5 compare these algorithms against Random Block and Sequential Block Sequences, respectively. Section 4.6.6 concludes our results obtained for the uniform cost algorithms.
4.6.1 Alternation Sequences

Our observations indicate that BW achieves the lowest switch ratio for larger buffer sizes (50, 100 and 500), when compared against its competitors while RC achieves the highest switch ratios.

For small buffer sizes (2, 5, 10 and 20), we observe that no algorithm is able to achieve low switch ratios, likely because the buffer is too small to produce long subsequences of the same colour. These results are illustrated in Figure 3.

Figure 3: The performance of uniform cost algorithms for the Alternation Sequence. The outer values on the x-axis correspond to buffer sizes and the inner values on the x-axis correspond to number of colours. The values on the y-axis correspond to the output/input switch ratio achieved by the algorithms. From this figure we can conclude that BW achieves lower switch ratios than RC and RR for for larger buffer sizes.
4.6.2 Delta Sequences

We observe lower switch ratios across all combinations of buffer sizes and number of colours for Delta Sequences, when compared to Alternation Sequences. A higher reduction in switch ratios is observed as buffer sizes increase.

![Delta - Uniform Cost - Switch Ratio](image)

**Figure 4:** The performance of uniform cost algorithms for Delta Sequences. The outer values on the x-axis correspond to buffer sizes and the inner values on the x-axis correspond to number of colours. The values on the y-axis correspond to the output/input switch ratio achieved by the algorithms. From this figure we can conclude that BW achieves the highest reduction in switch ratios.

Our observations indicate that BW has the highest reduction in switch ratios for all combinations of buffer sizes and number of colours, when compared to its competitors. As in the case of Alternation Sequences, RR has marginal improvement over RC for larger buffer sizes. These results are illustrated in Figure 4.
4.6.3 Random Sequences

Random Sequences exhibit a similar trend to that observed with Delta Sequences, for all combinations of buffer sizes and number of colours. Staying inline with the trend, BW achieves the highest reduction in switch ratios, when compared to RC and RR. RC achieves the lowest reduction in switch ratios. These results are illustrated in Figure 5.

Figure 5: The performance of uniform cost algorithms for Random Sequences. The outer values on the x-axis correspond to buffer sizes and the inner values on the x-axis correspond to number of colours. The values on the y-axis correspond to the output/input switch ratio achieved by the algorithms. From this figure we can conclude that BW achieves lower switch ratios than RR and RC.
4.6.4 Random Block Sequences

For combinations involving large buffer sizes and small number of colours, BW achieves the highest reduction in switch ratio followed by RR and then RC. These results are illustrated in Figure 6.

No algorithm achieves a high reduction in switch ratios as the number of colours increase. This behaviour is expected; the block sizes increase as the number of colours increase, thereby making it impossible for our algorithms to achieve high reduction in switch ratios.

![Random Block - Uniform Cost - Switch Ratio](image)

**Figure 6:** The performance of uniform cost algorithms for Random Block Sequences. The outer values on the x-axis correspond to buffer sizes and the inner values on the x-axis correspond to number of colours. The values on the y-axis correspond to the output/input switch ratio achieved by the algorithms. From this figure we can conclude that the BW achieves the lowest switch ratio for combinations involving large buffer sizes and small number of colours.
4.6.5 Sequential Block Sequences

Our observations indicate that BW achieves a lower switch ratio than RC and RR only when the buffer size is at least five times greater than the number of colours \((k \geq 5 \cdot C)\). RR has a lower switch ratio than RC and BW in most other cases. These results are illustrated in Figure 7.

**Figure 7:** The performance of uniform cost algorithms for Sequential Block Sequences. The outer values on the x-axis correspond to buffer sizes and the inner values on the x-axis correspond to number of colours. The values on the y-axis correspond to the output/input switch ratio achieved by the algorithms. From this figure we can conclude that BW has lower switch ratios than RC and RR only when \(k \geq 5 \cdot C\).
4.6.6 Conclusion

Based on the experimental results presented in the previous subsections, we can conclude that BW achieves the highest reduction in switch ratios for all input sequence types across all combinations of large buffer sizes (50, 100 and 500) and number of colours. Intuitively, we can conclude that no algorithm can achieve a high reduction in switch ratios, for small buffer sizes (2, 5, 10 and 20), because the buffer is too small to allow long subsequences of items of the same colour. This is bad because we would like to achieve maximal amounts of reduction in switch ratios. Hence we only consider large buffer sizes when summarizing our results.

The summary of our results for large buffer sizes are presented in Table 4. The first line compares the performance of BW and RC while the second line compares the performance of BW and RR. In each case, the percentage gives the range by which BW outperforms the other algorithm. We have rounded our percentage values to the nearest decimal for readability.

<table>
<thead>
<tr>
<th></th>
<th>Alternation</th>
<th>Delta</th>
<th>Random</th>
<th>Random Block</th>
<th>Sequential Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC-BW</td>
<td>BW</td>
<td>BW</td>
<td>BW</td>
<td>BW</td>
<td>BW</td>
</tr>
<tr>
<td></td>
<td>1% - 6%</td>
<td>1% - 4%</td>
<td>1% - 6%</td>
<td>5% - 6%</td>
<td>2% - 4%</td>
</tr>
<tr>
<td>RR-BW</td>
<td>BW</td>
<td>BW</td>
<td>BW</td>
<td>BW</td>
<td>BW</td>
</tr>
<tr>
<td></td>
<td>0% - 2%</td>
<td>0% - 2%</td>
<td>0% - 3%</td>
<td>2% - 3%</td>
<td>0% - 1%</td>
</tr>
</tbody>
</table>

Table 4: Summary of the uniform cost algorithms (BW-RR-RC) with the percentage improvements with respect to switches. The first row gives the name of the algorithm that has the highest reduction in switch ratio compared to its competitor. The second row gives the range of percentage improvement over its competitor.
4.7 Experimental Results for Non-Uniform Cost

In this section, we present the non-uniform cost algorithms’ performance against different input sequences for different combinations of cost functions, buffer sizes and number of colours.

We have implemented the following non-uniform cost algorithms. Refer to Chapter 3 for details.

- Maximum Adjusted Penalty (MAP)
- Threshold or Lowest Cost (TLC)
- Accelerated Threshold (Acc-T)

In the following subsections, we compare these algorithms against different input sequence types starting with Alternation Sequences in Section 4.7.1, Delta Sequences in Section 4.7.2, Random Sequences in Section 4.7.3, Random Block Sequences in 4.7.4 and Sequential Block Sequences in 4.7.5. Section 4.7.6 presents our conclusions regarding non-uniform cost algorithms.

For each of these input sequence types, we compare the algorithms using small buffer sizes (2, 5, 10 and 20), medium buffer sizes (50, 100 and 500) and large buffer sizes (1000, 2000, 5000 and 10,000).

Because of the computational complexity of MAP, we have restricted our large buffer size experiments to Acc-T and TLC. For large data sets, we have generated input sequences with size 1,000,000. Number of colours have also been increased to 100, 500 and 1000. Since the algorithms are fairly high on wall time, we have restricted the number of times we execute our algorithms for random sequences and random cost functions, to 30 times each and averaged the results.
For each buffer size, we in turn analyze the algorithms against different cost functions: Cost-Equals-Colour cost function, Cost-Equals-Quadratic-Colour cost function, Random-Cost cost function, Colour-Difference cost function and Uniform-Cost cost function. Relevant figures are presented to demonstrate our results. In all figures, brown represents MAP, orange represents TLC, and purple represents Acc-T.

4.7.1 Alternation Sequences

This section presents experimental results for non-uniform cost algorithms against Alternation Sequences, while varying the combinations of cost functions, buffer sizes and number of colours.

**Small Buffer Sizes:** Our observations indicate little or no reordering for very small buffer sizes (2 and 5) when comparing switch ratios against the Cost-Equals-Colour cost function. We observe that TLC achieves a higher reduction in switch ratios than its competitors when the number of colours is greater than two times the buffer size \(C \geq 2 \cdot k\). In other cases, Acc-T achieves a higher reduction in switch ratios than MAP and TLC when the number of colours is at most the buffer size \(C \leq k\).

Acc-T achieves higher reduction in cost ratios than its competitors when the number of colours is at most the buffer size \(C \leq k\). As with switch ratios, TLC achieves a higher reduction in cost ratios when the number of colours is at least two times the buffer size \(C \geq 2 \cdot k\). No algorithm achieves low cost ratios when the number of colours is significantly larger than the buffer size.

The Cost-Equals-Quadratic-Colour cost function exhibits a similar trend to that observed with the Cost-Equals-Colour cost function. Our observations indicate that Acc-T achieves a higher reduction in switch ratios than MAP and TLC when the number of colours is at most the buffer size \(C \leq k\). However, an exception occurs when \(k = 2\) and \(C = 5\); In this case, Acc-T achieves a lower switch ratio than MAP and TLC, even though the number of colours is greater than the buffer size. TLC
Figure 8: The performance of non-uniform cost algorithms against Alternation Sequences with the Cost-Equals-Quadratic-Colour cost function. The outer values on the x-axis correspond to buffer sizes and the inner values on the x-axis correspond to number of colours. The values on the y-axis correspond to the output/input switch ratio achieved by the algorithms. From this figure we can conclude that Acc-T achieves a higher reduction in switch ratios than its competitors when $C \leq k$.

achieves a higher reduction in switch ratios when the number of colours is at most two times the size of the buffer $C \geq 2 \cdot k$). These results are illustrated in Figure 8.

The trend continues when comparing cost ratios, with Acc-T having higher reduction in cost ratios when the number of colours is at most the size of the buffer ($C \leq k$), and TLC having higher reduction in cost ratios when the number of colours is at least two times greater than the buffer size ($C \geq 2 \cdot k$). In most cases, MAP has switch and cost ratios comparable to Acc-T.

A similar trend as that observed with the Cost-Equals-Colour cost function and
Cost-Equals-Quadratic-Colour cost function is also observed when comparing switch and cost ratios against the Random-Cost cost function. Acc-T achieves a higher reduction in switch and cost ratios than MAP and TLC when the number of colours is at most buffer size ($C \leq k$). MAP achieves a slightly lower switch and cost ratio than TLC in these cases. TLC achieves a higher reduction in switch and cost ratios than its competitors when the number of colours is at least two times the buffer size ($C \geq 2 \cdot k$). MAP and Acc-T have comparable switch and cost ratios in these cases.

**Figure 9:** The performance of non-uniform cost algorithms against Alternation Sequence with the Colour-Difference cost function. The outer values on the x-axis correspond to the buffer sizes and the inner values on the x-axis correspond to the number of colours. The values on the y-axis correspond to the output/input switch ratio achieved by the algorithms. From this figure we can conclude that Acc-T achieves a lower switch ratio than MAP and TLC when $C \leq k$.

Acc-T achieves a higher reduction in switch ratios than MAP and TLC when
the number of colours is at most the buffer size \( C \leq k \), while comparing switch ratios against the Colour-Difference cost function. It is interesting to note that MAP and TLC have identical switch ratios in this case as illustrated in Figure 9. When comparing cost ratios, we observe that the results are reversed. Acc-T has lower reduction in cost ratios (i.e., is less performant), when the number of colours is at most the buffer size \( C \leq k \), with TLC and MAP having identical cost ratios.

For the Uniform-Cost cost function, our observations indicate that all algorithms achieve comparable switch ratios, across all combinations of buffer sizes and number of colours.

**Medium Buffer Sizes:** Our observations indicate that Acc-T achieves higher reduction in switch ratios over all combinations of buffer sizes and number of colours for the Cost-Equals-Colour cost function. Similar results hold when comparing cost ratios. We also observe that Acc-T and MAP have comparable cost ratios, while TLC achieves the lowest reduction in cost ratios.

The Cost-Equals-Quadratic-Colour cost function exhibits a similar trend to that observed with the Cost-Equals-Colour cost function, with Acc-T achieving a significantly higher reduction in switch ratios than MAP and TLC (see Figure 10). Our observations indicate that Acc-T has a higher reduction in cost ratios than TLC across all combinations of buffer sizes and number of colours, while comparing cost ratios. An exception occurs when the buffer size and the number of colours are both 50; In this case, TLC has the lowest reduction in cost ratios. The cost ratios of MAP are comparable to Acc-T for all combinations of buffer sizes and number of colours.

Results observed for the Random-Cost cost function are consistent with those observed for the Cost-Equals-Colour cost function and the Cost-Equals-Quadratic-Colour cost function. Acc-T has a significantly higher reduction in switch ratios, when compared to MAP and TLC, across all combinations of buffer sizes and number of colours. The trend is also consistent when comparing cost ratios; we observe that
Figure 10: The performance of non-uniform cost algorithms against Alternation Sequences with the Cost-Equals-Quadratic-Colour cost function. The outer values on the x-axis correspond to buffer sizes and the inner values on the x-axis correspond to number of colours. The values on the y-axis correspond to the output/input switch ratios achieved by the algorithms. From this figure we can conclude that Acc-T achieves significantly higher reduction in switch ratios than MAP and TLC.

Acc-T has higher reduction in cost ratios than TLC and MAP across all combinations of buffer sizes and number of colours. An exception occurs when the buffer size and number of colours are both 50; In this case, Acc-T has a lower reduction in cost ratio than TLC and MAP.

Acc-T has a higher reduction in switch ratios, when compared to TLC and MAP, across all combinations of buffer sizes and number of colours, for the Colour-Difference cost function. However, the reduction in cost ratios for Acc-T is significantly lower than that for TLC and MAP, as illustrated in Figure. [11] It is also interesting to
Figure 11: The performance of non-uniform cost algorithms against Alternation Sequences with the Colour-Difference cost function. The outer values on the x-axis correspond to buffer sizes and the inner values on the x-axis correspond to number of colours. The values on the y-axis correspond to output/input cost ratios achieved by the algorithms. From this figure we can conclude that Acc-T has lower reduction in cost ratios when compared to its competitors.

note that TLC and MAP have identical switch and cost ratios in this cases.

For the Uniform-Cost cost function, our observations indicate that all algorithms have comparable switch ratios, across all combinations of buffer sizes and number of colours.

Large Buffer Sizes: Our observations indicate that Acc-T achieves a significantly higher reduction in switch and cost ratios than TLC across all combinations of buffer sizes and number of colours, for the Cost-Equals-Colour cost function. The highest reduction in switch and cost ratios for Acc-T are observed when buffer sizes are
sufficiently larger than number of colours. These results are illustrated in Figure 12. As mentioned earlier, we have not considered MAP for large buffer size analysis because of its computational complexity and high wall time.

**Figure 12:** The performance of non-uniform cost algorithms TLC and Acc-T against Alteration Sequences for the Cost-Equals-Colour cost function. The outer values on the x-axis correspond to buffer sizes and the inner values on the x-axis correspond to number of colours. The values on the y-axis correspond to output/input cost ratios achieved by the algorithms. From this figure we can conclude that Acc-T has the higher reduction in cost ratios than TLC regardless of the buffer sizes and number of colours.

Acc-T has a significantly higher reduction in switch ratios, across all combinations of buffer sizes and number of colours, while comparing switch ratios against the Cost-Equals-Quadratic-Colour cost function. Like with the Cost-Equals-Colour cost function, the best switch ratios are obtained when the number of colours is one-tenth the buffer sizes \((k \geq 10 \cdot C)\). Overall, we observe a very high reduction in switch ratios...
for Acc-T when compared to TLC. Acc-T achieves a higher reduction in cost ratios than TLC when buffer sizes are at least two times the number of colours \( k \geq 2 \cdot C \).

Our observations for the Random-Cost cost function are inline with the observations for the Cost-Equals-Quadratic-Colour cost function, where we find that Acc-T has a significantly higher reduction in switch ratios across all combinations of buffer sizes and number of colours. Acc-T achieves a higher reduction in cost ratios than TLC, when buffer sizes are at least two times the number of colours \( k \geq 2 \cdot C \). This is illustrated in Figure 13.
Consistent with other cost functions, our observations indicate Acc-T has a significantly higher reduction in switch ratios than TLC, for all combinations of buffer sizes and number of colours, when using the Colour-Difference cost function. However, as seen with small buffer sizes, Acc-T does not achieve high reduction in cost ratios when compared to TLC for this cost function.

For the Uniform-Cost cost function, both algorithms have comparable switch ratios across all combinations of buffer sizes and number of colours.

4.7.2 Delta Sequences

This section presents experimental results for non-uniform cost algorithms against Delta Sequences, while varying the combinations of cost functions, buffer sizes and number of colours.

**Small Buffer Sizes**: Our observations reveal that Acc-T achieves a slightly higher reduction in switch ratios when compared to MAP and TLC, against the Cost-Equals-Colour cost function. TLC has the lowest reduction in switch ratios while MAP achieves slightly higher reduction in switch ratios than that of TLC. These results are illustrated in Figure 14. Similar results are observed while comparing cost ratios. Acc-T has a slightly higher reduction in cost ratios than its competitors while TLC has the lowest reduction in cost ratios.

The Cost-Equals-Quadratic-Colour cost function exhibits a similar trend to that observed with the Cost-Equals-Colour cost function, where Acc-T achieves a higher reduction in switch ratios when compared with TLC and MAP, across all combinations of buffer sizes and number of colours. TLC and MAP have comparable switch ratios. These results are illustrated in Figure 15. Acc-T and MAP achieve slightly higher reduction in cost ratios than TLC across all combinations of buffer sizes and number of colours, when the cost ratios are being compared.

This trend is also consistent with the Random-Cost cost function, where Acc-T
Figure 14: The performance of non-uniform cost algorithms against Delta Sequences for the Cost-Equals-Colour cost function. The outer values on the x-axis correspond to buffer sizes and the inner values on the x-axis correspond to number of colours. The values on the y-axis correspond to output/input switch ratios achieved by the algorithms. From this figure we can conclude that Acc-T achieves higher reduction in switch ratios than its competitors.

achieves the highest reduction in switch ratios compared to MAP and TLC, for all combinations of buffer sizes and number of colours. TLC has the lowest reduction in switch ratios while the switch ratio reduction of MAP is in between that of TLC and Acc-T. All algorithms achieve comparable cost ratios, with Acc-T having marginally higher reduction in cost ratios when compared to TLC and MAP in most cases.

Our observations reveal that Acc-T achieves the highest reduction in switch ratios among all algorithms across all combinations of buffer sizes and number of colours for the Colour-Difference cost function, while MAP and TLC have comparable switch
Figure 15: The performance of non-uniform cost algorithms against Delta Sequences with the Cost-Equals-Quadratic-Colour cost function. The outer values on the x-axis correspond to buffer sizes and the inner values on the x-axis correspond to number of colours. The values on the y-axis correspond to output/input switch ratios achieved by the algorithms. From this figure we can conclude that Acc-T achieves higher reduction in switch ratios than MAP and TLC.

A stark difference is observed when we compare cost ratios for the Colour-Difference cost function, where we observe that Acc-T has a significantly lower reduction in cost ratios when compared to TLC and MAP. It is also interesting to note that TLC and MAP have almost identical cost ratios for all combinations of buffer sizes and number of colours.

For the Uniform-Cost cost function, all algorithms achieve comparable switch ratios for all combinations of buffer sizes and number of colours.
Medium Buffer Sizes: Our observations indicate that Acc-T achieves a significantly higher reduction in switch ratios when compared with MAP and TLC across all combinations of buffer sizes and number of colours, against the Cost-Equals-Colour cost function. This result also holds for cost ratio comparison, where we observe that Acc-T achieves a slightly higher reduction in cost ratios than MAP and TLC. An exception occurs when $k = 50$ and $C = 50$, where MAP has a slightly higher reduction in cost ratio than Acc-T.

Acc-T has the highest reduction in switch ratios when compared to MAP and TLC across all combinations of buffer sizes and number of colours, for the Cost-Equals-Quadratic-Colour cost function. When cost ratios are compared, Acc-T achieves a slightly higher reduction in cost ratios than TLC and MAP, for most cases. However, when the buffer size is 50 and number of colours is 50 ($k = 50, C = 50$), MAP achieves a higher reduction in cost ratio than TLC and Acc-T. Another case is when the buffer size is 100 and number of colours is 50 ($k = 100, C = 50$), TLC has a lower cost ratio than MAP and Acc-T.

The Random-Cost cost function exhibits the same trend as that observed for the Cost-Equals-Quadratic-Colour cost function, where Acc-T achieves a significantly higher reduction in switch ratios when compared to TLC and MAP for all combinations of buffer sizes and number of colours, as illustrated in Figure 16. While the performance is significantly better when comparing switch ratios, it is only slightly better when cost ratios are being compared, where we observe that Acc-T has slightly higher reduction in cost ratios when compared to TLC and MAP, across all combinations of buffer sizes and number of colours.

We observe that Acc-T achieves a higher reduction in switch ratios when compared to TLC and MAP across all combinations of buffer sizes and number of colours, for the Colour-Difference cost function. However, Acc-T has the lowest reduction in cost ratios when compared to TLC and MAP across all combinations of buffer sizes and
Figure 16: The performance of non-uniform cost algorithms against Delta Sequences with the Random-Cost cost function. The outer values on the x-axis correspond to buffer sizes and the inner values on the x-axis correspond to number of colours. The values on the y-axis correspond to output/input switch ratios achieved by the algorithms. From this figure we can conclude that Acc-T has significantly higher reduction in switch ratios than MAP and TLC.

Our observations reveal that the cost ratios for Acc-T are about two times the cost ratios achieved by MAP and TLC, as illustrated in Figure. While the cost ratios are higher than those for MAP and TLC, it is to be noted that Acc-T achieves reasonably high reduction in cost ratios, with the cost ratio being less than 0.45 in the worst case. This indicates that more than half the cost of the input sequence has been eliminated. This can be inferred from Figure.

For the Uniform-Cost cost function, all algorithms achieve a high reduction in switch ratios. An exception occurs when the buffer size is 50 and number of colours
is 50 ($k = 50, C = 50$), in which case MAP has a slightly lower reduction in switch ratio than TLC and Acc-T.

**Large Buffer Sizes:** Recollect that we do not perform experiments for MAP because of high wall time. Our observations indicate that Acc-T has a higher reduction in switch ratios than TLC across all combinations of buffer sizes and number of colours, for the Cost-Equals-Colour cost function. Likewise, this result also holds when comparing cost ratios, as we find that the Acc-T has a higher reduction in cost ratios than TLC across all combinations of buffer sizes and number of colours.
Figure 18: The performance of the algorithms TLC and Acc-T against Delta Sequences with the Cost-Equals-Quadratic-Colour cost function. The outer values on the x-axis correspond to buffer sizes and the inner values on the x-axis correspond to number of colours. The values on the y-axis correspond to output/input switch ratios achieved by the algorithms. From this figure we can conclude that Acc-T has significantly higher reduction in switch ratios than TLC, across all combinations of buffer sizes and number of colours.

Similar observations hold good for the Cost-Equals-Quadratic-Colour cost function, where we observe that Acc-T has a significantly higher reduction in switch ratios than TLC across all combinations of buffer sizes and number of colours, as illustrated in Figure 18. The trend also holds good for cost ratio comparisons, where we observe that Acc-T achieves a higher reduction in cost ratios than TLC, across combinations of buffer sizes and number of colours.

As seen in the case of the previous two cost functions, Acc-T has higher reductions in switch and cost ratios than TLC across all combinations of buffer sizes and number
Figure 19: The performance of Acc-T and TLC Delta Sequences with the Random-Cost cost function. The outer values on the x-axis correspond to buffer sizes and the inner values on the x-axis correspond to number of colours. The values on the y-axis correspond to the output/input switch ratios achieved by the algorithms. From this figure we can conclude that Acc-T has higher reduction in cost ratios than TLC, across all combinations of buffer sizes and number of colours.

Consistent with small and medium buffer sizes, Acc-T achieves a higher reduction in switch ratios than TLC against the Colour-Difference cost function. However, the reduction in cost ratios are lower when compared to that of TLC. It is to be noted that this difference is marginal; overall, both algorithms achieve a very high reduction in cost ratios across all combinations of buffer sizes and number of colours.

For the Uniform-Cost cost function, both algorithms achieve comparable switch ratios across all combinations of buffer sizes and number of colours.
4.7.3 Random Sequences

This section presents experimental results for non-uniform cost algorithms against Random Sequences, while varying the combinations of cost functions, buffer sizes and number of colours.

**Small Buffer Sizes:** Our observations indicate that that Acc-T achieves a higher reduction in switch ratios than MAP and TLC across all combinations of buffer sizes and number of colour, for the Cost-Equals-Colour cost function. However, when comparing cost ratios, we observe that Acc-T has marginally lower reduction in cost ratios than TLC and MAP when the number of colours is more than two times the buffer size \( C \geq 2 \cdot k \). For all other cases, Acc-T has marginally higher reduction in cost ratios when compared to TLC and MAP.

The Cost-Equals-Quadratic-Colour cost function exhibits a similar trend to that observed with the Cost-Equals-Colour cost function, where Acc-T achieves a higher reduction in switch ratios when compared to TLC and MAP across all combinations of buffer sizes and number of colours, but has a lower reduction in cost ratios when compared to its competitors, Specifically, we observe that Acc-T has a lower reduction cost ratios than TLC and MAP for all combinations where the number of colours is greater than the buffer size \( C \geq k \). For all other cases, Acc-T has a marginally higher reduction in cost ratios.

Consistent with the Cost-Equals-Colour cost function and the Cost-Equals-Quadratic-Colour cost function, Acc-T achieves a higher reduction in switch ratios when compared to TLC and MAP across all combinations of buffer sizes and number of colours, even for the Random-Cost cost function, as illustrated in Figure 20. Our results are also consistent when comparing cost ratios, where we observe that Acc-T achieves marginally lower reduction in cost ratios when the number of colours is greater than the buffer size \( C \geq k \). TLC and MAP have comparable cost ratios.
with TLC having marginally lower cost ratios when the number of colours is at least two times the buffer size \((C \geq 2 \cdot k)\).

![Graph](image)

**Figure 20:** The performance of non-uniform algorithms against Random Sequences with the Random-Cost cost function. The outer values on the x-axis correspond to buffer sizes and the inner values on the x-axis correspond to number of colours. The values on the y-axis correspond to the output/input switch ratios achieved by the algorithms. From this figure we can conclude that Acc-T achieves a higher reduction in switch ratios than MAP and TLC.

Our results for the Colour-Difference cost function are consistent with those observed for Alternation and Delta Sequences, where we observe that Acc-T has a higher reduction in switch ratios when compared to TLC and MAP, across all combinations of buffer sizes and number of colours. However, when comparing cost ratios, we observe that Acc-T has a significantly lower reduction in cost ratios than TLC and MAP across all combinations of buffer sizes and number of colours as illustrated in Figure
TLC and MAP achieve comparable cost ratios for this cost function.

**Figure 21:** The performance of non-uniform cost algorithms against Random Sequences with the Colour-Difference cost function. The outer values on the x-axis correspond to buffer sizes and the inner values on the x-axis correspond to number of colours. The values on the y-axis correspond to the output/input cost ratios achieved by the algorithms. From this figure we can conclude that Acc-T has a significantly lower reduction in cost ratios than TLC and MAP.

For the Uniform-Cost cost function, we observe that all algorithms achieve comparable switch ratios, across all combinations of buffer sizes and number of colours.

**Medium Buffer Sizes:** For the Cost-Equals-Colour cost function and the Cost-Equals-Quadratic-Colour cost function, our observations indicate that Acc-T achieves a significantly higher reduction in switch ratios than TLC and MAP, across all combinations of buffer sizes and number of colours. When comparing cost ratios, we observe that Acc-T has a slightly higher reduction in cost ratios than TLC and MAP, across
all combinations of buffer sizes and number of colours, for both these cost functions. These results are illustrated in Figure 22 and Figure 23.

![Random - Cost Equals Colour - Medium Buffers - Switch Ratio](image)

**Figure 22:** The performance of non-uniform cost algorithms against Random Sequences with the Cost-Equals-Colour cost function. The outer values on the x-axis correspond to buffer sizes and the inner values on the x-axis correspond to number of colours. The values on the y-axis correspond to output/input switch ratios achieved by the algorithms. From this figure we can conclude that Acc-T achieves a significantly higher reduction in switch ratios than TLC and MAP.

This trend where Acc-T achieves a significantly higher reduction in switch ratios than MAP and TLC, and slightly higher reduction in cost ratios when compared to MAP and TLC, across all combinations of buffer sizes and number of colours, extends to the Random-Cost cost function as well. Similar figures can be plotted to reveal this behaviour.

Results observed for the Colour-Difference cost function are consistent with those
Figure 23: The performance of non-uniform cost algorithms against Random Sequences with the Cost-Equals-Quadratic-Colour cost function. The outer values on the x-axis correspond to buffer sizes and inner values on the x-axis correspond to number of colours. The values on the y-axis correspond to output/input cost ratios achieved by the algorithms. From this figure we can conclude that Acc-T achieves a slightly higher reduction in cost ratios than TLC and MAP.

For small buffer sizes, where Acc-T achieves a higher reduction in switch ratios than MAP and TLC across all combinations of buffer sizes and number of colours. However, our observations also indicate that Acc-T does not achieve a high reduction in cost ratios when compared to its competitors. TLC achieves the highest reduction in cost ratios while MAP has a performance in between the two.

For the Uniform-Cost cost function, we observe that all algorithms have comparable reduction in switch ratios, with MAP achieving marginally higher reduction in switch ratios than TLC and Acc-T in some cases.
**Figure 24:** The performance of algorithms TLC and Acc-T against Random Sequences with the Cost-Equals-Colour cost function. The outer values on the x-axis correspond to buffer sizes and the inner values on the x-axis correspond to number of colours. The values on the y-axis correspond to the output/input switch ratios achieved by the algorithms. From this figure we can conclude that Acc-T achieves a higher reduction in switch ratios than TLC for all combinations of buffer sizes and number of colours.

**Large Buffer Sizes:** Recall that we do not conduct experiments for MAP because of its high wall time. For the Cost-Equals-Colour cost function, our observations indicate that Acc-T achieves a higher reduction in switch and cost ratios than TLC, across all combinations of buffer sizes and number of colours. Our best results for switch ratio reduction are observed when buffer sizes are at least 10 times the number of colours ($k \geq 10 \cdot C$). These results are illustrated in Figure 24.

The Cost-Equals-Quadratic-Colour cost function exhibits the same trend to that observed with the Cost-Equals-Colour cost function, where we observe that Acc-T
achieves a significantly higher reduction in switch ratios across all combinations of buffer sizes and number of colours. Likewise, Acc-T achieves a higher reduction in cost ratios than TLC, regardless of the combinations of buffer sizes and number of colours.

![Random - Colour Difference - Large Buffers - Cost Ratio](image.png)

**Figure 25:** The performance of the TLC and Acc-T for against Random Sequences with the Colour-Difference cost function. The outer values on the x-axis correspond to buffer sizes and the inner values on the x-axis correspond to number of colours. The values on the y-axis correspond to output/input cost ratios achieved by the algorithms. From this figure we can conclude that both algorithms have very high reduction in cost ratios.

For the Random-Cost cost function, we observe that Acc-T achieves a significantly higher reduction in switch ratios than TLC for all combinations of buffer sizes and number of colours. However, when comparing cost ratios, we observe that Acc-T has a higher reduction in cost ratios than TLC when the buffer size is greater than two
times the number of colours \( (k \geq 2 \cdot C) \).

As observed with the other cost functions, Acc-T achieves a higher reduction in switch ratios than TLC across all combinations of buffer sizes and number of colours, even for the Colour-Difference cost function. When comparing cost ratios, both algorithms do exceptionally well with cost ratios being well below 0.04 in the worst case indicating that about 96% of the cost of the input sequences have been reduced. However, Acc-T has a lower reduction in cost ratios compared to TLC as illustrated in Figure 25.

For the Uniform-Cost cost function, our observations reveal that Acc-T has a marginally lower reduction in switch ratios for smaller buffer sizes, i.e, Acc-T achieves a higher reduction in switch ratios than TLC only when the buffer size is at least two times the number of colours \( (k \geq 2 \cdot C) \).

4.7.4 Random Block Sequences

This section presents experimental results for non-uniform cost algorithms against Random Block Sequences, while varying the combinations of cost functions, buffer sizes and number of colours.

**Small Buffer Sizes:** Our observations indicate that Acc-T achieves a higher reduction in switch ratios than its competitors when the number of colours is small, for the Cost-Equals-Colour cost function. MAP and TLC have comparable performance. These are illustrated in Figure 26. All algorithms achieve comparable cost ratios across all combinations of buffer sizes and number of colours. Since the input sequences come in variable sized blocks, all algorithms can achieve a high reduction in cost ratios only when the size of the blocks are smaller than the buffer sizes. This can happen only when the number of colours are small since the block sizes increase with the increase in the number of colours. This is reflected in the results obtained as we observe switch ratios closer to 1.0 for large number of colours.
Figure 26: The performance of non-uniform cost algorithms against Random Block Sequences with the Cost-Equals-Colour cost function. The outer values on the x-axis correspond to buffer sizes and the inner values on the x-axis correspond to number of colours. The values on the y-axis correspond to output/input switch ratios achieved by the algorithms. From this figure we can conclude that Acc-T achieves marginally higher reduction in switch ratios than its competitors for small number of colours.

The Cost-Equals-Quadratic-Colour cost function exhibits a trend similar to that observed with the Cost-Equals-Colour cost function, where Acc-T achieves a higher reduction in switch ratios than TLC and MAP across all combinations of buffer sizes and number of colours. When comparing cost ratios, we observe that all algorithms achieve comparable ratios with TLC achieving marginally higher reduction in cost ratios than MAP and Acc-T in most cases.

For the Random-Cost cost function, our observations indicate that all algorithms achieve comparable reduction in switch and cost ratios across all combinations of
buffer sizes and number of colours as illustrated in Figure 27.

Figure 27: The performance of non-uniform cost algorithms against Random Block Sequences with the Random-Cost cost function. The outer values on the x-axis correspond to buffer sizes and the inner values on the x-axis correspond to number of colours. The values on the y-axis correspond to output/input cost ratios achieved by the algorithms. From this figure we can conclude that all the algorithms have comparable reduction in cost ratios.

As observed with Cost-Equals-Colour cost function, Acc-T achieves a slightly higher reduction in switch ratios than its competitors when the number of colours is small, but otherwise all algorithms achieve comparable reduction in switch ratios, for the Colour-Difference cost function. However, when comparing cost ratios, Acc-T has a lower reduction in cost ratios than TLC and MAP for all combinations of buffer sizes and number of colours. MAP and TLC have comparable reduction in cost ratios.

For the Uniform-Cost cost function, all algorithms have comparable reduction in
switch ratios across all combinations of buffer sizes and number of colours.

**Medium Buffer Sizes:** Our observations indicate that all algorithms achieve high reduction in switch and cost ratios only when the number of colours is small. This is not surprising since the block sizes increase as the number of colours increase leaving little scope for permutation.

For the Cost-Equals-Colour cost function, we observe that Acc-T achieves a higher reduction in switch and cost ratios when compared to TLC and MAP across all combinations of buffer sizes and number of colours. While Acc-T does achieve a higher reduction in cost ratios, the reduction is not as significant as those observed with switch ratios.

The Cost-Equals-Quadratic-Colour cost function exhibits the same trend to that observed with the Cost-Equals-Colour cost function where Acc-T achieves a higher reduction in switch ratios when compared to its competitors across all combinations of buffer sizes and number of colours. However, the similarity is only limited to switch ratio comparison. When comparing the cost ratios our observations indicate that all algorithms achieve comparable cost ratios as illustrated in Figure 28.

For the Random-Cost cost function, our observations reveal that Acc-T achieves a higher reduction in switch ratios than MAP and TLC across all combinations of buffer sizes and number of colours. When comparing cost ratios, we observe that Acc-T has lower reduction in cost ratios when compared to its competitors, as the number of colours increases.

When comparing switch ratios for the Colour-Difference cost function, we observe that all algorithms achieve comparable reduction in switch ratios. However, cost ratio comparisons reveal that Acc-T has significantly lower reduction in cost ratios than TLC and MAP, with the cost ratios being more than two times the cost ratios observed for TLC and MAP as illustrated in Figure 29.

For the Uniform-Cost cost function, all algorithms have comparable reduction in
**Figure 28:** The performance of non-uniform cost algorithms against Random Block Sequences with the Cost-Equals-Quadratic-Colour cost function. The outer values on the x-axis correspond to buffer sizes and the inner values on the x-axis correspond to number of colours. The values on the y-axis correspond to the output/input cost ratios achieved by the algorithms. From this figure we can conclude that all algorithms have comparable cost ratios.

switch ratios across all combinations of buffer sizes and number of colours.

**Large Buffer Sizes:** Recall that we do not perform experiments for MAP because of its computational complexity. When we compare switch ratios of TLC and Acc-T against the Cost-Equals-Colour cost function, our observations indicate that Acc-T has a significantly higher reduction in switch ratios than TLC, when the number of colours is a small fraction of the buffer size. For example, when the buffer size is 10 times the number of colours \((k = 10 \cdot C)\), we observe that Acc-T has 82% of the switches in the input sequence where as TLC has about 88% of switches in the input
**Figure 29:** The performance of non-uniform cost algorithms against Random Block Sequences with the Colour-Difference cost function. The outer values on the x-axis correspond to buffer sizes and the inner values on the x-axis correspond to number of colours. The values on the y-axis correspond to the output/input cost ratios achieved by the algorithms. From this figure we can conclude that the cost ratio for Acc-T is more than two times than that for TLC and MAP.

sequence. This is not a very significant difference, but when buffer size is a 100 times the number of colours \((k = 100 \cdot C)\), Acc-T has 23.5% of the switches present in the input sequence while TLC has 47.4% of the original switches. This is a significant result. However, when we compare cost ratios of these two algorithms, we observe that Acc-T only achieves a slightly higher reduction in cost ratios than TLC in most cases, the best result being, for buffer size 10000 and 100 colours.

The Cost-Equals-Quadratic-Colour cost function exhibits a similar trend to that observed with the Cost-Equals-Colour cost function, where the reduction in switch...
Figure 30: The performance of non-uniform cost algorithms TLC and Acc-T against Random Block Sequences with the Cost-Equals-Quadratic-Colour cost function. The outer values on the x-axis correspond to buffer sizes and the inner values on the x-axis correspond to number of colours. The values on the y-axis correspond to output/input switch ratios achieved by the algorithms. From this figure we can conclude that Acc-T has a significantly higher reduction in switch ratios than TLC as the buffer size increases.
Figure 31: The performance of non-uniform cost algorithms TLC and Acc-T Random Block Sequences with the Random-Cost cost function. The outer values on the x-axis correspond to buffer sizes and the inner values on the x-axis correspond to number of colours. The values on the y-axis correspond to the output/input switch ratios achieved by the algorithms. From this figure we can conclude that Acc-T achieves a higher reduction in switch ratios for $C = 100$ and gets marginally lower reduction than TLC as the number of colours increase.

significantly higher reduction in switch ratio for 100 colours across all buffer sizes. However, while the reduction in switch ratios was always higher for Acc-T in all other input sequence types, the reduction in switch ratios gets marginally lower than TLC as the number of colours increases. This behaviour is consistent with all buffer sizes as illustrated in Figure 31. As to cost ratio comparison, we observe that Acc-T has a slightly lower reduction in cost ratios than TLC across all combinations of buffer sizes and number of colours.

For the Colour-Difference cost function, our observations indicate that Acc-T
achieves a slightly higher reduction in switch ratios for 100 colours; as the number of colours increases, both the algorithms have comparable reduction in switch ratios. However, when cost ratios are compared, TLC outperforms Acc-T by a very large margin as the reduction in cost ratios for Acc-T are extremely low compared to TLC.

For the Uniform-Cost cost function, both algorithms achieve comparable reduction in switch ratios across all combinations of buffer sizes and number of colours.

4.7.5 Sequential Block Sequences

This section presents experimental results for non-uniform cost algorithms against Sequential Block Sequences, while varying the combinations of cost functions, buffer sizes and number of colours.

Small Buffer Sizes: For the Cost-Equals-Colour cost function, no algorithm achieves any permutation very small buffer sizes (2 and 5) indicating that the output sequence is identical to the input sequence. This behaviour is not surprising since our block sizes are 5; the buffer would only have items of a single colour making it impossible to achieve any reordering. For buffer size 10, we observe that TLC achieves higher reduction in switch and cost ratios than MAP and Acc-T when the number of colours is 5, while Acc-T achieves a higher reduction in switch and cost ratio than MAP and TLC for buffer size 20 and 5 colours. In all other cases, we do not see any reordering as the block sized nature of the input sequence combined with the increase in colours does not permit any reordering.

As observed with the Cost-Equals-Colour cost function, the Cost- Equals-Quadratic-Colour cost function also witnesses no permutations for very small buffer sizes (2 and 5). However, for buffer sizes 10 and 20, all algorithms achieve reordering ratios close to 1.0 indicating that permutations are minimal. For buffer size 20, TLC achieves the highest reduction in switch and cost ratios as long as the number of colours is at most the buffer size \( C \leq k \). It is interesting to note that, while TLC
Figure 32: The performance of non-uniform cost algorithms against Sequential Block Sequences with the Cost-Equals-Quadratic-Colour cost function. The outer values on the x-axis correspond to buffer sizes and the inner values on the x-axis correspond to number of colours. The values on the y-axis correspond to output/input cost ratios achieved by the algorithms. From this figure we observe that Acc-T does not permute the input sequence for $k = 10, 20$ and $C = 5, 10$, while its competitors achieve some reordering.

When comparing switch ratios, we observe that the Random-Cost cost function follows a similar trend to that of the Cost-Equals-Quadratic-Colour cost function, where Acc-T does not permute the input sequences for the same combination of buffer sizes and number of colours ($k = 10, 20$ and $C = 5, 10$). Consistent with the Cost-Equals-Quadratic-Colour cost function, TLC and MAP do permute the input
sequence, with TLC having a lower switch ratio than MAP. All algorithms achieve comparable reductions in cost ratios, with TLC achieving the highest reduction in cost ratios for the combinations of buffer sizes and number of colours mentioned earlier.

Figure 33: The performance of non-uniform cost algorithms against Sequential Block Sequences with the Colour-Difference cost function. The outer values on the x-axis correspond to buffer sizes and the inner values on the x-axis correspond to number of colours. The values on the y-axis correspond to output/input switch ratios achieved by the algorithms. From this figure we observe that all algorithms have identical switch ratios; except for $k = 20$ and $C = 5$, where Acc-T achieves a higher reduction in switch ratio than TLC and MAP.

For the Colour-Difference cost function, our observations reveal that all algorithms achieve identical reduction in cost ratios, across all combinations of buffer sizes and number of colours. This trend continues when we compare switch ratios as well. However, an exception occurs when the buffer size is 20 and the number of colours is 5, ($k = 20$ and $C = 5$), in this case, Acc-T achieves a higher reduction in switch ratio.
than TLC and MAP as illustrated in Figure 33.

For the Uniform-Cost cost function, all algorithms achieve identical reduction in switch ratios. An exception occurs when $k = 20$ and $C = 5$, where MAP achieves a significantly higher reduction in switch ratio than TLC and Acc-T. In this case TLC and Acc-T fail to permute the input sequence. It is interesting to note that this is exactly the same buffer size/number of colours combination that had a non-identical performance in the Colour-Difference cost function.

**Medium Buffer Sizes**: For the Cost-Equals-Colour cost function, our observations indicate that Acc-T achieves a higher reduction in switch ratios than MAP and TLC when buffer sizes are at least two times the number of colours, $(k \geq 2 \cdot C)$. In all other cases, TLC achieves a higher reduction in switch ratios than Acc-T and MAP. However, when comparing cost ratios, Acc-T achieves a higher reduction in cost ratios than its competitors only when the number of colours is one-tenth the buffer sizes, $(k \geq 10 \cdot C)$.

With the Cost-Equals-Quadratic-Colour cost function, our observations indicate that, Acc-T a higher reduction in switch ratios than its competitors, when buffer sizes are 5 times greater than the number of colours, $(k \geq 5 \cdot C)$ as illustrated in Figure 34. When comparing cost ratios, Acc-T achieves a higher reduction in cost ratios than TLC and MAP, when the buffer sizes are 10 times greater than the number of colours, $(k \geq 10 \cdot C)$.

Our observations for the Random-Cost cost function are inline with those for the Cost-Equals-Quadratic-Colour cost function, where Acc-T achieves a higher reduction in switch ratios than MAP and TLC, when buffer sizes are at least 5 times the number of colours $(k \geq 5 \cdot C)$; while Acc-T achieves a higher reduction in cost ratios than its competitors, when buffer sizes are at least 10 times the number of colours $(k \geq 10 \cdot C)$.

For the Colour-Difference cost function, our observations indicate that Acc-T achieves a higher reduction in switch ratios than MAP and TLC when buffer sizes are
Figure 34: The performance of non-uniform cost algorithms against Sequential Block Sequences with the Cost-Equals-Quadratic-Colour cost function. The outer values on the x-axis correspond to buffer sizes and the inner values on the x-axis correspond to number of colours. The values on the y-axis correspond to output/input switch ratios achieved by the algorithms. From this figure we can conclude that Acc-T has achieves a higher reduction in switch ratios than MAP and TLC when $k \geq 5 \cdot C$.

at least 5 times greater than the number of colours. ($k \geq 5 \cdot C$) as illustrated in Figure 35. However, the exact opposite results are observed for cost ratio comparisons where Acc-T has a significantly lower reduction in cost ratios when $k \geq 5 \cdot C$.

For the Uniform-Cost cost function, all algorithms have comparable reduction in switch ratios across all combinations of buffer sizes and number of colours.

Large Buffer Sizes: For the Cost-Equals-Colour cost function, our observations indicate that Acc-T achieves a higher reduction in switch ratios than TLC when the buffer size is at least 10 times the number of colours ($k \geq 10 \cdot C$). For all other
Figure 35: The performance of non-uniform cost algorithms Sequential Block Sequences with the Colour-Difference cost function. The outer values on the x-axis correspond to buffer sizes and the inner values on the x-axis correspond to number of colours. The values on the y-axis correspond to output/input switch ratios achieved by the algorithms. From this figure we can conclude that Acc-T has a higher reduction in switch ratios than MAP and TLC when $k \geq 5 \cdot C$.

In most cases, TLC achieves a higher reduction in switch ratios than Acc-T. This result also holds good for cost ratio comparison, where we observe that Acc-T achieves a higher reduction in cost ratios than TLC when the buffer size is at least 10 times the number of colours ($k = 10 \cdot C$), while TLC has higher reduction in cost ratios than Acc-T for all other cases. This result is illustrated in Figure 36.

The Cost-Equals-Quadratic-Colour cost function exhibits the same trend to that observed with the Cost-Equals-Colour cost function, where Acc-T has higher reduction in switch and cost ratios than TLC, when the buffer size is at least 10 times the
Figure 36: The performance of TLC and Acc-T Sequential Block Sequences with the Cost-Equals-Colour cost function. The outer values on the x-axis correspond to buffer sizes and the inner values on the x-axis correspond to number of colours. The values on the y-axis correspond to the output/input cost ratios achieved by the algorithms. From this figure we can conclude that Acc-T has a higher reduction in cost ratio than TLC when $k \geq 10 \cdot C$.

number of colours ($k \geq 10 \cdot C$), while TLC has a higher reduction in switch and cost ratios for all other cases.

Our observations for the Random-Cost cost function are inline with those observed for the Cost-Equals-Colour cost function and the Cost-Equals-Quadratic-Colour cost function, where Acc-T achieves a higher reduction in switch and cost ratios than TLC, when the buffer size is at least 10 times the number of colours ($k \geq 10 \cdot C$) as illustrated in Figure 37.
Figure 37: The performance of TLC and Acc-T against Sequential Block Sequences with the Random-Cost cost function. The outer values on the x-axis correspond to buffer sizes and the inner values on the x-axis correspond to number of colours. The values on the y-axis correspond to the output/input cost ratios achieved by the algorithms. From this figure we can conclude that Acc-T achieves a significantly higher reduction in switch ratios when $k \geq 10 \cdot C$.

When comparing switch ratios for the Colour-Difference cost function, our observations indicate that Acc-T achieves a higher reduction in switch ratios than TLC when the buffer size is at least 10 times the number of colours ($k \geq 10 \cdot C$). However, when comparing cost ratios, we observe that Acc-T does not achieve high reduction in cost ratios across all combinations of buffer sizes and number of colours.

For the Uniform-Cost cost function, both algorithms have comparable reduction in switch ratios across all combinations of buffer sizes and number of colours.
4.7.6 Conclusion

In this section we present a summary of results obtained for non-uniform cost algorithms across variations of input sequence types, cost functions, buffer sizes and number of colours.

**Small Buffer Sizes:** Our results from Sections 4.7.1 through 4.7.5 indicate little reordering for small buffer sizes (2, 5, 10 and 20). This trend has been consistent with all input sequence types and cost functions. Intuitively, we can conclude that any reduction in switch and cost ratios will be minimal for small buffers since the limited size of the buffer cannot produce long subsequences of items of the same colour.

**Medium Buffer Sizes:** Table 5 summarizes our observations when comparing switch ratios of all non-uniform cost algorithms. The first line of each entry in the table gives the name of the algorithm that achieves a higher reduction in switch ratios when compared to its competitors, while the second line gives the percentage range by which it performs better.

For each buffer size, the values are averaged over different combinations of number of colours (2, 5, 10, 20 and 50) and rounded to the nearest whole number to enhance readability. The percentage improvements are expressed as a range where this range corresponds to the percentage improvements observed across different buffer sizes.

From Table 5, Acc-T achieves a higher reduction in switch ratios for almost all combinations of input sequences and cost functions. An exception occurs when considering Random Sequences and the Uniform-Cost cost function, where MAP achieves a slightly higher reduction in switch ratios. The highest reduction in switch ratios for Acc-T is observed with the Cost-Equals-Quadratic-Colour cost function. The Cost-Equals-Colour cost function achieves the next best performance, followed by the Random-Cost and the Colour-Difference cost functions. Overall, we conclude
Table 5: Summary of non-uniform cost algorithms with respect to the reductions in switch ratios. The first line in each row gives the name of the algorithm that achieves a higher reduction in switch ratios when compared to its competitors, while the second line gives the percentage range by which it performs better.

Based on results presented in Table 6, Acc-T has a slightly higher reduction in cost ratios than its competitors against the Cost-Equals-Colour, Cost-Equals-Quadratic-Colour and the Random-Cost cost functions. An exception occurs when considering Delta Sequences and the Cost-Equals-Colour cost function, where MAP has a better percentage range. However, it may be noted that the computational complexity of MAP prevents practical implementations. Acc-T has a slightly higher reduction in
Table 6: Summary of non-uniform cost algorithms with respect to the reductions in cost ratios. The first line in each row gives the name of the algorithm that achieves a higher reduction in cost ratios when compared to its competitors, while the second line gives the percentage range by which it performs better.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Cost Equals Colour</th>
<th>Cost Equals Quadratic</th>
<th>Random</th>
<th>Colour Difference</th>
<th>Uniform</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alternation</td>
<td>Acc-T</td>
<td>Acc-T</td>
<td>Acc-T</td>
<td>TLC</td>
<td>comparable</td>
</tr>
<tr>
<td></td>
<td>0% - 1%</td>
<td>0% - 1%</td>
<td>0% - 1%</td>
<td>0% - 6%</td>
<td>-</td>
</tr>
<tr>
<td>Delta</td>
<td>MAP</td>
<td>Acc-T</td>
<td>MAP</td>
<td>TLC</td>
<td>comparable</td>
</tr>
<tr>
<td></td>
<td>0% - 1%</td>
<td>0% - 2%</td>
<td>0% - 1%</td>
<td>2% - 7%</td>
<td>-</td>
</tr>
<tr>
<td>Random</td>
<td>Acc-T</td>
<td>Acc-T</td>
<td>Acc-T</td>
<td>TLC</td>
<td>comparable</td>
</tr>
<tr>
<td></td>
<td>1% - 2%</td>
<td>0% - 1%</td>
<td>0% - 1%</td>
<td>0% - 4%</td>
<td>-</td>
</tr>
<tr>
<td>RandomBlock</td>
<td>Acc-T</td>
<td>Acc-T</td>
<td>Acc-T</td>
<td>TLC</td>
<td>comparable</td>
</tr>
<tr>
<td></td>
<td>2% - 4%</td>
<td>0% - 1%</td>
<td>0% - 1%</td>
<td>8% - 13%</td>
<td>-</td>
</tr>
<tr>
<td>SequentialBlock</td>
<td>TLC</td>
<td>TLC</td>
<td>TLC</td>
<td>TLC</td>
<td>comparable</td>
</tr>
<tr>
<td></td>
<td>0% - 6%</td>
<td>0% - 8%</td>
<td>0% - 4%</td>
<td>4% - 6%</td>
<td>-</td>
</tr>
</tbody>
</table>

Although we observe that Acc-T has slightly better cost ratios than its competitors, our best results are obtained with respect to switch ratios. Additionally, while the Cost-Equals-Quadratic-Colour cost function had the best performance with respect to switch ratios, we observe that the Cost-Equals-Colour cost function has the best performance, with respect to costs.

TLC has the best performance for the Colour-Difference cost function, regardless of the input sequence type or cost function, indicating that Acc-T is not suitable for variable cost functions.

Based on Table 6, we can conclude that Acc-T is generally a better suited algorithm for the Alternation, Delta, Random and Random Block Sequences against the
Cost-Equals-Colour, Cost-Equals-Quadratic-Colour and the Random-Cost cost functions. For the Sequential Block Sequence, TLC has the best performance regardless of the cost function used. Finally, for the Colour-Difference cost function, TLC has the best performance regardless of the input sequence type.

**Large Buffer Sizes:** Table 7 compares our results on the switch ratios for TLC and Acc-T for large data sets.

<table>
<thead>
<tr>
<th></th>
<th>Cost Equals Colour</th>
<th>Cost Equals Quadratic</th>
<th>Random Difference</th>
<th>Colour Difference</th>
<th>Uniform</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Alternation</strong></td>
<td>Acc-T</td>
<td>11% - 17%</td>
<td>Acc-T</td>
<td>5% - 12%</td>
<td>Acc-T</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14% - 21%</td>
<td></td>
<td>1% - 6%</td>
<td>comparable</td>
</tr>
<tr>
<td><strong>Delta</strong></td>
<td>Acc-T</td>
<td>3% - 5%</td>
<td>Acc-T</td>
<td>3% - 8%</td>
<td>Acc-T</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8% - 9%</td>
<td></td>
<td>1% - 2%</td>
<td>comparable</td>
</tr>
<tr>
<td><strong>Random</strong></td>
<td>Acc-T</td>
<td>11% - 17%</td>
<td>Acc-T</td>
<td>5% - 12%</td>
<td>Acc-T</td>
</tr>
<tr>
<td></td>
<td></td>
<td>17% - 21%</td>
<td></td>
<td>1% - 5%</td>
<td>comparable</td>
</tr>
<tr>
<td><strong>RandomBlock</strong></td>
<td>Acc-T</td>
<td>2% - 9%</td>
<td>Acc-T</td>
<td>0% - 3%</td>
<td>Acc-T</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2% - 11%</td>
<td></td>
<td>0% - 2%</td>
<td>comparable</td>
</tr>
</tbody>
</table>

**Table 7:** Summary of non-uniform cost algorithms with respect to the reductions in switch ratios for large buffer sizes. The first line in each row gives the name of the algorithm that achieves a higher reduction in switch ratios when compared to its competitors, while the second line gives the percentage range by which it performs better.

The values in Table 7 give the range of percentage values observed across different buffer sizes (1000, 2000, 5000 and 10,000). For each of these buffer sizes, the values are averaged over the different combinations of number of colours (100, 500 and 1000) and the nearest whole decimal number. Like in the previous case, the range corresponds to improvements observed across different buffer sizes.

From Table 7 we can unanimously conclude that Acc-T is the best algorithm for achieving high reduction in switch ratios for the Alternation, Delta, Random
and Random Block Sequences. Acc-T achieves the best performance with the Cost-Equals-Quadratic-Colour cost function, followed by the Cost-Equals-Colour and the Random-Cost cost functions. For all input sequence types, we observe that the Colour-Difference cost function has the lowest performance among all cost functions that we have tested.

For the Sequential Block Sequence, no algorithm achieves a better performance than its competitors across all combinations of buffer sizes and number of colours. Hence we have presented detailed results for the Sequential Block Sequence in Table 8. The values are averaged over combinations of different number of colour (100, 500 and 1000). Since they are presented for each buffer size individually, they correspond to a single number and not a range. We have rounded the values to the nearest whole number.

<table>
<thead>
<tr>
<th></th>
<th>1000</th>
<th>2000</th>
<th>5000</th>
<th>10000</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cost Equals Colour</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Acc-T</td>
<td>2%</td>
<td>9%</td>
<td>17%</td>
<td>18%</td>
</tr>
<tr>
<td>TLC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Cost Equals Quadratic</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Acc-T</td>
<td>1%</td>
<td>2%</td>
<td>16%</td>
<td>23%</td>
</tr>
<tr>
<td>TLC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Random</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLC</td>
<td>1%</td>
<td>13%</td>
<td>12%</td>
<td>12%</td>
</tr>
<tr>
<td><strong>Colour Difference</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Acc-T</td>
<td>2%</td>
<td>7%</td>
<td>6%</td>
<td>3%</td>
</tr>
<tr>
<td>TLC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Uniform</strong></td>
<td>comparable</td>
<td>comparable</td>
<td>comparable</td>
<td>comparable</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 8: Summary of non-uniform cost algorithms with respect to the reductions in switch ratios for large buffer sizes for the Sequential Block Sequence. The first line in each row gives the name of the algorithm that achieves a higher reduction in switch ratios when compared to its competitors, while the second line gives the percentage range by which it performs better.

From Table 8 we can conclude that Acc-T is better at achieving low switch ratios
than TLC against all combinations of cost functions and buffer sizes. An exception occurs while considering the buffer size 2000 against all combinations of colours (100, 500 and 1000), in which case TLC is better.

Table 9 summarizes our observations while comparing cost ratios for TLC and Acc-T for large data sets. The values in Table 9 gives the range of percentage values observed across different buffer sizes. For each buffer size, the values are averaged over the different number of colours (100, 500 and 1000) and rounded to the nearest whole number. As in the previous cases, the range corresponds to percentage improvements observed across different buffer sizes.

<table>
<thead>
<tr>
<th></th>
<th>Cost Equals Colour</th>
<th>Cost Equals Quadratic</th>
<th>Random</th>
<th>Colour Difference</th>
<th>Uniform</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Alternation</strong></td>
<td>Acc-T</td>
<td>Acc-T</td>
<td>Acc-T</td>
<td>TLC</td>
<td>comparable</td>
</tr>
<tr>
<td></td>
<td>2% - 4%</td>
<td>0% - 3%</td>
<td>0% - 2%</td>
<td>2% - 22%</td>
<td>-</td>
</tr>
<tr>
<td><strong>Delta</strong></td>
<td>Acc-T</td>
<td>Acc-T</td>
<td>Acc-T</td>
<td>TLC</td>
<td>comparable</td>
</tr>
<tr>
<td></td>
<td>0.5% - 1%</td>
<td>0.5% - 1%</td>
<td>1% - 2%</td>
<td>4% - 13%</td>
<td>-</td>
</tr>
<tr>
<td><strong>Random</strong></td>
<td>Acc-T</td>
<td>Acc-T</td>
<td>Acc-T</td>
<td>TLC</td>
<td>comparable</td>
</tr>
<tr>
<td></td>
<td>1% - 4%</td>
<td>1% - 3%</td>
<td>1% - 2%</td>
<td>0.5% - 2%</td>
<td>-</td>
</tr>
<tr>
<td><strong>Random Block</strong></td>
<td>Acc-T</td>
<td>Acc-T/TLC</td>
<td>TLC</td>
<td>TLC</td>
<td>comparable</td>
</tr>
<tr>
<td></td>
<td>0% - 2%</td>
<td>0% - 1%</td>
<td>3% - 4%</td>
<td>43% - 46%</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 9: Summary of non-uniform cost algorithms with respect to the reductions in switch ratios for large buffer. The first line in each row gives the name of the algorithm that achieves a higher reduction in switch ratios when compared to its competitors, while the second line gives the percentage range by which it performs better.

From Table 9, we can conclude that Acc-T is better for achieving low cost ratios for Alternation, Delta and Random Sequences against the Cost-Equals-Colour, Cost-Equals-Quadratic-Colour and Random-Cost cost functions. For the Colour-Difference cost function, we can conclude that TLC is the best algorithm for achieving low
cost ratios regardless of the input sequence type. Finally, for the Uniform-Cost cost function, both algorithms achieve comparable cost ratios across all combinations of input sequence types and cost functions.

For the Random Block Sequence, Acc-T achieves lower cost ratios for the Cost-Equals-Colour cost function, while TLC is better suited for the Random-Cost and Colour-Difference cost functions. For the Cost-Equals-Quadratic-Colour cost function, TLC achieves lower cost ratios for buffer sizes 1000 and 2000, while Acc-T achieves lower cost ratios for 5000 and 10000. This is represented by the Acc-T/TLC in the corresponding entry for Random Block Sequence in the Cost-Equals-Quadratic-Colour cost function column of Table 9.

<table>
<thead>
<tr>
<th></th>
<th>1000</th>
<th>2000</th>
<th>5000</th>
<th>10000</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cost Equals Colour</strong></td>
<td>TLC</td>
<td>TLC</td>
<td>Acc-T</td>
<td>Acc-T</td>
</tr>
<tr>
<td></td>
<td>8%</td>
<td>24%</td>
<td>1%</td>
<td>3%</td>
</tr>
<tr>
<td><strong>Cost Equals Quadratic</strong></td>
<td>TLC</td>
<td>TLC</td>
<td>TLC</td>
<td>Acc-T</td>
</tr>
<tr>
<td></td>
<td>14%</td>
<td>19%</td>
<td>1%</td>
<td>3%</td>
</tr>
<tr>
<td><strong>Random</strong></td>
<td>TLC</td>
<td>TLC</td>
<td>TLC</td>
<td>Acc-T</td>
</tr>
<tr>
<td></td>
<td>9%</td>
<td>25%</td>
<td>0.5%</td>
<td>2%</td>
</tr>
<tr>
<td><strong>Colour Difference</strong></td>
<td>TLC</td>
<td>TLC</td>
<td>TLC</td>
<td>TLC</td>
</tr>
<tr>
<td></td>
<td>5%</td>
<td>21%</td>
<td>24%</td>
<td>10%</td>
</tr>
<tr>
<td><strong>Uniform</strong></td>
<td>comparable</td>
<td>comparable</td>
<td>comparable</td>
<td>comparable</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1000</th>
<th>2000</th>
<th>5000</th>
<th>10000</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cost Equals Colour</strong></td>
<td>TLC</td>
<td>TLC</td>
<td>Acc-T</td>
<td>Acc-T</td>
</tr>
<tr>
<td></td>
<td>8%</td>
<td>24%</td>
<td>1%</td>
<td>3%</td>
</tr>
<tr>
<td><strong>Cost Equals Quadratic</strong></td>
<td>TLC</td>
<td>TLC</td>
<td>TLC</td>
<td>Acc-T</td>
</tr>
<tr>
<td></td>
<td>14%</td>
<td>19%</td>
<td>1%</td>
<td>3%</td>
</tr>
<tr>
<td><strong>Random</strong></td>
<td>TLC</td>
<td>TLC</td>
<td>TLC</td>
<td>Acc-T</td>
</tr>
<tr>
<td></td>
<td>9%</td>
<td>25%</td>
<td>0.5%</td>
<td>2%</td>
</tr>
<tr>
<td><strong>Colour Difference</strong></td>
<td>TLC</td>
<td>TLC</td>
<td>TLC</td>
<td>TLC</td>
</tr>
<tr>
<td></td>
<td>5%</td>
<td>21%</td>
<td>24%</td>
<td>10%</td>
</tr>
<tr>
<td><strong>Uniform</strong></td>
<td>comparable</td>
<td>comparable</td>
<td>comparable</td>
<td>comparable</td>
</tr>
</tbody>
</table>

Table 10: Summary of non-uniform cost algorithms with respect to the reductions in switch ratios for large buffers for the Sequential Block Sequence. The first line in each row gives the name of the algorithm that achieves a higher reduction in switch ratios when compared to its competitors, while the second line gives the percentage range by which it performs better.

In the case of Sequential Block Sequence, no algorithm has lower cost ratios than the other across all buffer sizes. Hence we have expanded the results for the Sequential Block Sequence into a separate table, Table 10.
From Table 10, TLC achieves a lower cost ratio than Acc-T against the Cost-Equals-Colour, Cost-Equals-Quadratic-Colour, Random-Cost and Colour-Difference cost functions, for buffer sizes 1000, 2000 and 5000. An exception occurs while considering the Cost-Equals-Colour cost function with buffer size 5000, in which case Acc-T achieves a lower cost ratio. For buffer size 10,000, Acc-T achieves a lower cost ratio than TLC for the Cost-Equals-Colour, Cost-Equals-Quadratic-Colour and Random-Cost cost functions. An exception occurs while considering the Colour-Difference cost function with buffer size 10,000 where TLC has a lower cost ratio. Finally, Both algorithms achieve comparable cost ratios for the Uniform-Cost cost function across all buffer sizes.

4.8 Comparison with Previous Work

All experimental results presented thus far have only considered random sequences against uniform costs with a fixed number of colours. In this thesis we have made an attempt to extend the experimental results in three directions. Firstly, in addition to random sequences, we also consider input sequences that have specific patterns. Secondly, we have formulated fixed and variable cost functions and compared combinations of input sequence types, buffer sizes and cost functions. Finally, we have varied the number of colours that can appear in the input sequence and compared the algorithms while varying numbers of colours and with different buffer sizes and cost functions.

As mentioned in Section 2.3, Krokowski et al. [18], present experimental results for the algorithms RC, RR and BW for the polygon rendering application of the RBM problem in the online setting. Their results suggested that while all algorithms achieve a good reordering, BW achieves a slightly higher reduction in switch ratios than RC and RR. While our experiments are not tailored for the polygon rendering
application, our observation that BW achieves the highest reduction in switch ratios is consistent with theirs, for uniform costs.

Our results differ from the results obtained by Englert et al. in two ways. They specifically design input sequences to bring out the best and worst case performance ratios for MAP. We do not generate algorithm-specific input sequences but have tried to compare the performance of several algorithms with different input sequences. Secondly, their results are focused on experimentally verifying that MAP achieves a competitive ratio of $O(\log k)$ and hence they have compared the performance of their algorithms against an optimal offline algorithm. As our experimental results are focused on getting the maximum reduction in switch and cost ratios, and not to verify competitive ratios, we have not implemented an optimal offline algorithm and used only switch and cost ratios as our performance metrics.

To the best of our knowledge, the only algorithms that have previously been implemented are FIFO, LRU, MCF, BW, RC, RR and MAP. In this thesis, we have omitted the algorithms that have the lowest reordering, based on previous experimental results (FIFO, LRU and MCF). We also include two new algorithms, TLC and Acc-T, and compare their performance together with other algorithms while varying the combinations of input sequences, cost functions, buffer sizes and number of colours.
Chapter 5

Conclusion

In this chapter we present a summary of the results presented in this thesis along with directions for future research.

5.1 Summary of Contributions

The contributions made in this thesis are twofold.

**Accelerated Threshold:** In Chapter 3 we presented our new algorithm, Accelerated Threshold (Acc-T), for the non-uniform variant of the online RBM problem. Our algorithm is derived from Threshold or Lowest Cost (TLC).

Acc-T differs from TLC in two ways. Firstly, we perform updates to our counters over $k$ sub-iterations on each occasion that the next new active colour is chosen; TLC only does a single update on each such occasion. Secondly, we eliminate TLC’s need to sometimes fall back to selecting the colour with the lowest cost, as our increased updates always ensure that at least one colour has reached its threshold.

**Experimental Evaluations:** In Section 4.6 we presented experimental results for the uniform cost algorithms implemented in this thesis. For the uniform cost algorithms, our conclusion is that Bounded Waste (BW) achieves a lower switch ratio than Round Robin (RR) and Random Choice (RC) for all input sequences across all
combinations of buffer sizes and number of colours.

In Section 4.7, we compared the performance of the non-uniform cost algorithms implemented in this thesis. For small buffer sizes, no algorithm is able to achieve any significant reordering, regardless of input sequence or cost function. This behaviour was not surprising as small buffers cannot produce long subsequences of items of the same colour. Our conclusion is that small buffer sizes are unsuitable to achieve any significant reordering.

From our results for both medium and large buffer sizes, our conclusion is that, among the three algorithms presented, Acc-T gives the best switch ratio across all combinations of input sequence types and cost functions.

For medium and large buffer sizes, the best results for Acc-T, when comparing switch ratios against its competitors, are observed with the Cost-Equals-Quadratic-Colour cost function. The least improvement is observed with the Colour-Difference cost function. For the Uniform-Cost cost function, in most cases, all the algorithms have comparable performance, except for cases described in Section 4.7.6.

When comparing the cost ratios for medium and large buffer sizes with non-uniform cost algorithms, Acc-T has slightly better performance than its competitors for the Alternation, Delta and Random Sequences for the Cost-Equals-Colour, Cost-Equals-Quadratic-Colour and the Random-Cost cost functions. Unlike with switch ratios, the best results are observed using Cost-Equals-Colour cost function, followed by the Cost-Equals-Quadratic-Colour cost function and then the Random-Cost cost function.

For medium and large buffer sizes, Acc-T does not achieve good performance for the Colour-Difference cost function when comparing cost ratios, making it unsuitable for scenarios that have variable cost functions. Finally, Acc-T does not achieve good performance for most cases of the Sequential Block Sequence when cost ratios are compared.
5.2 Future Work

In this section, we present a few directions for future research.

1. In Chapter 3 we present our algorithm Acc-T for the RBM problem. In Chapter 4 we present experimental results, but the algorithm has not yet been analyzed theoretically. Acc-T is computationally very similar to TLC which has been analyzed using an intricate linear programming relaxation technique. Using the same technique, can our algorithm be proven to have a better competitive ratio than TLC?

2. In Chapter 4 we observe that Acc-T has a lower switch ratio for the Colour-Difference cost function, yet always has a higher cost ratio regardless of the sequence types, buffer sizes and number of colours used. Would it be possible to modify the algorithm so that it achieves a better cost ratio for variable cost functions?

3. An interesting extension to the experimental results would be to consider more sophisticated input sequences (e.g., Gaussian Distribution, Fibonacci, Non-Uniform Random) and cost functions (e.g., Cost-Equals-Square-Root-Colour, Colour-Product) and observe whether the results for Acc-T continue hold good.

4. Recent extensions to the RBM problem [13] use advice bits to give the algorithm some information about the future or some information about an optimal offline algorithm. Would it be possible to incorporate advice bits into Acc-T to achieve better performance?

5. RBM algorithms presented thus far do not take into account the probability distribution and the dependency information between the items in the input sequence. An interesting extension would be to take into account the distribution and learn the dependencies between items of different colours in the input
sequence. This information can be modelled as a Markov chain which can be
used by Acc-T to choose the next new active colour. Would this technique
result in a higher reduction in switch and cost ratios that can be achieved by
Acc-T?
List of References


