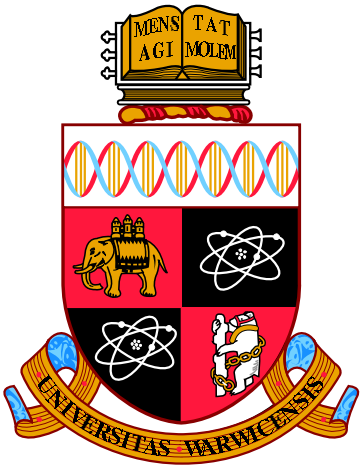


# Approximation Algorithms for Packing and Buffering Problems

by

**Nicolaos Matsakis**

**Thesis**

Submitted to the University of Warwick

for the degree of

**Doctor of Philosophy**

**Department of Computer Science**

September 2015

THE UNIVERSITY OF WARWICK

# Contents

# Acknowledgments

First and foremost, I am indebted to Matthias Englert for his guidance as my supervisor in Warwick. His help into teaching me how to arrange my ideas into concrete mathematical proofs has been crucial.

I am, also, indebted to my current advisors Artur Czumaj and Ranko Lazić and to my former advisor Maxim Sviridenko, for helping me to improve my research potential. I would, also, like to thank my collaborator Marcin Mucha for discussing with me some of the ideas I had on various research problems.

Finally, I was very fortunate to have made very good friends, during the years that I have spent in Warwick. I would like to thank Ebrahim, Lehilton, Lukáš, Matthew, Michail and Raphael for all the good times we had and I wish to each one of them all the best for his career.

# Declarations

None of the work presented in this thesis has been submitted for a previous degree at any university. All work was conducted during my period of study in the University of Warwick.

Chapter 2 is based on joint work with Matthias Englert and Marcin Mucha (Englert et al. [2014]), which was published in the proceedings of the 11th Latin American Theoretical Informatics Symposium (LATIN 2014), held in Montevideo, Uruguay, between March 31 and April 4 2014.

Chapter 3 is based on joint work with Matthias Englert.

Chapter 4 includes individual work (Matsakis [2016], to appear) which has been accepted for publication by the Student Research Forum of the 42nd International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2016), to be held in Harrachov, Czech Republic on January 2016.

# Abstract

This thesis studies online and offline approximation algorithms for packing and buffering problems.

In the second chapter of this thesis, we study the problem of packing linear programs online. In this problem, the online algorithm may only increase the values of the variables of the linear program and his goal is to maximize the value of the objective function of it. The online algorithm has initially full knowledge of all parameters of the linear program, except for the right-hand sides of the constraints which are gradually revealed to him by the adversary. This online problem has been introduced by Ochel et al. [2012]. Our contribution (Englert et al. [2014]) is to provide improved upper bounds for the competitiveness of both deterministic and randomized online algorithms for this problem, as well as an optimal deterministic online algorithm for the special case of linear programs involving two variables.

In the third chapter we study the offline COLORFUL BIN PACKING problem. This problem is a variant of the BIN PACKING problem, where each item is associated with a color and where there exists the additional restriction that two items packed consecutively into the same bin cannot share the same color. The COLORFUL BIN PACKING problem has been studied mainly from an online perspective and has been introduced as a generalization of the BLACK AND WHITE BIN PACKING problem (Balogh et al. [2012]), i.e., the special case of this problem for two colors. We provide (joint work with Matthias Englert) a 2-appoximate algorithm for the COLORFUL BIN PACKING problem.

In the fourth chapter we study the Longest Queue Drop (LQD) online algorithm for shared-memory switches with three and two output ports. The Longest Queue Drop algorithm is a well-known online algorithm used to direct the packet flow of shared-memory switches. According to LQD, when the buffer of the switch becomes full, a packet is preempted from the longest queue in the buffer to free buffer space for the newly arriving packet which is accepted. We show (Matsakis [2016], to appear) that the Longest Queue Drop algorithm is (3/2)-competitive for three-port switches, improving the previously best upper bound of 5/3 (Kobayashi et al. [2007]). Additionally, we show that this algorithm is exactly (4/3)-competitive for two-port switches, correcting a previously published result claiming a tight upper bound of $\frac{4M-4}{3M-2} < 4/3$, where $M \in \mathbb{Z}^+$ denotes the buffer size.

# Chapter 1

# Introduction

This thesis deals with several problems in the areas of online and offline computation. Let us start by providing the informal definition of one of the problems considered in this thesis: Assume that we are given a set of rectangular items, each of which has a unit width and a height which is at most one. Suppose, also, that we are supplied with an unlimited number of square bins of unit dimensions. We are asking what is the minimum number of bins used, so that each item is packed into a bin and the sum of the sizes of the packed items into each bin is at most one.

This problem is called BIN PACKING and its various settings have a wide range of practical applications from the design of integrated circuits to the backup file storage and the cargo shipping in the transport industry. Hence, it should come as no surprise that BIN PACKING is one of the most well-studied problems in the area of theoretical computer science, with relevant publications ranging over a period of several decades (Ullman [1971]; Johnson [1973]; Karmarkar and Karp [1982]; Simchi-Levi [1994]; Rothvoß [2013]; Dósa and Sgall [2013]).

Assuming that the conjecture $P \neq NP$ is true, as most of the researchers in the area believe, it is not possible to derive an algorithm which will run in polynomial time in the size of the input to this problem, for any input to BIN PACKING, guaranteeing to solve this problem exactly. Taking into consideration the fact that exact

exponential-time algorithms may require prohibitive running times even for practical input instances of every-day use, we need to look towards obtaining algorithms that will run in polynomial time in the size of the input (or to be computationally *efficient* algorithms, as we shall say) to BIN PACKING, relaxing our requirement of solving this problem exactly to solving it approximately, but within a reasonable amount of time.

We emphasize on the fact, however, that any algorithm for BIN PACKING is assumed to have complete information on the input instance of this problem a priori, that is the sizes of all given items are known in advance and this information can be used during the computation.

Let us, now, proceed towards giving a second example of a problem: Suppose that tasks arrive one by one over time and each of them has to be assigned upon its arrival to a processor, from a finite set of parallel processors of, possibly, different processing speeds. The processing of the assigned tasks starts when the last arriving task has been assigned to a processor and each task assignment is irrevocable, which means each task has to be processed by the processor that it has been initially assigned to. Unfortunately, we are provided with no information regarding the processing time requirement of any task until this task arrives and we are unaware of the number of remaining tasks to arrive until the last task arrives. Our objective is to assign each task to a processor such that the time it takes until all processors end their processing is minimized. This problem is one of the many variants of LOAD BALANCING. The various settings of LOAD BALANCING have been, also, extensively studied in the relevant literature (Graham [1969]; Azar et al. [1994]; Bartal et al. [1995]; Azar et al. [1995]; Albers [1999]).

In the case of LOAD BALANCING we are provided with incomplete knowledge on the input of the problem. An algorithm for this problem has no information regarding the processing time requirement of any task and the number of remaining tasks to arrive, a priori. Even if we assume the unrealistic scenario that we are

given unlimited computational resources (contrary to our first example of the BIN PACKING problem), an algorithm for LOAD BALANCING is restricted to perform under a state of uncertainty regarding the sequence of arriving tasks.

It is, also, worth mentioning that since each task assignment is irrevocable, the decision of an algorithm to assign any task to a processor, may have an irreversible impact on its overall performance. In other words, the task assignment that an algorithm for LOAD BALANCING outputs, may not be one of those assignments which minimize the time it takes for all processors to complete their processing, for a given input of arriving tasks. We note that this may be the case even if the given input sequence is comprised by relatively few tasks and, therefore, for an input where a reasonable amount of time for the completion of the computation is not an issue of concern.

The aforementioned computational efficiency and the lack of information regarding the input instance of the problem are not the only restrictions that an algorithm may be imposed with. We may recall, here, the space storage requirements that an algorithm may be, also, imposed with. In this thesis, however, we shall only deal with algorithms that are restricted to complete in time which is upper-bounded by a polynomial in the size of the input to the given problem or algorithms that are restricted to perform under a state of incomplete information regarding the input instance.

The two described problems, BIN PACKING and LOAD BALANCING, are two examples of optimization problems. An optimization problem can be either a cost minimization problem or a profit maximization problem.

For each input $I$ to a minimization problem $\mathcal{P}$ (belonging to a set $\mathcal{I}$ of legal inputs to $\mathcal{P}$), there exists a set of feasible outputs $F(I)$ and for each feasible output $O \in F(I)$ there exists a positive real number $g(I, O)$, which is the *cost*. The function $g$ is called as *objective function* or *cost function*. The *optimal solution* to an input instance of $\mathcal{P}$ is a feasible output that achieves the smallest objective function value,

for this given input. Finally, an *optimal algorithm* for $\mathcal{P}$ is an algorithm which for any input $I \in \mathcal{I}$ outputs an optimal solution.

In a similar way, but for the case of a maximization problem $\mathcal{P}'$, there exists an objective function (or *profit* function) that associates each legal input to the problem $\mathcal{P}'$ and the derived output, with a positive real number, which is the *profit*. The optimal solution to an input instance of $\mathcal{P}'$ is a feasible output achieving the largest value of the objective function, for the given input. Finally, an optimal algorithm for $\mathcal{P}'$ is an algorithm which for any legal input to this problem, outputs an optimal solution.

An optimization problem, where the information on the input instance is gradually received by the algorithm and the output of the algorithm has to be produced in an online manner, is called *online* problem. This lack of information refers solely to the input instance and not to the parameters of the problem. For example, in the case of the previously described variant of LOAD BALANCING, the number of parallel processors and the processing speed of each of those are considered to be parameters that are known in advance.

Even though the running times of algorithms are generally considered to be an irrelevant issue for the area of online computation, we prefer obtaining online algorithms (as we shall call the algorithms solving online problems) that are computationally efficient. In fact, it is quite interesting to mention that the great majority of published online algorithms in the relevant literature happen to be computationally efficient as well (Borodin and El-Yaniv [1998]; Fiat and Woeginger [1998]).

On the other hand, if complete information on the input is provided to the algorithm a priori, then the optimization problem is called *offline* and this algorithm is called *offline*, as well. The analysis of offline approximation algorithms for NP-hard optimization problems deals with the subject of deriving computationally efficient algorithms which will guarantee to produce solutions of a satisfactory approximation. In other words, we are seeking to derive approximation algorithms

that will be considered preferable from exact algorithms that run in exponential time in the size of the input to this problem.

Whilst some optimization problems can be viewed as being naturally offline and others as having an online character, there exists a large number of problems that have been studied from both an online and an offline aspect, due to their rich variety of practical applications from both of these perspectives. For instance, the offline BIN PACKING problem has been extensively studied in the relevant literature, as we have already mentioned. However, the online BIN PACKING problem has been, also, studied to a great extent (Lee and Lee [1985]; Ramanan et al. [1989]; Seiden [2002]; Balogh et al. [2015c]) as we shall see in Chapter 3.

Let us, now, concentrate solely on offline computation, for a while. The study of offline approximation algorithms for NP-hard optimization problems started to flourish around the early 1970's, after the breakthrough papers of Cook [1971] and Karp [1972] were published. However, approximation algorithms for NP-hard problems had been derived before even the concept of NP-hardness was introduced; we may recall here an approximation algorithm for the MINIMUM EDGE COLORING problem which is due to Vizing [1964].

An immediate question that may arise is how two offline approximation algorithms for the same NP-hard optimization problem are compared to each other, or to rephrase this, how the quality of the offline approximation is being quantified. We shall only deal with deterministic offline algorithms here without having to mention this explicitly.

Hence, letting $\mathsf{OPT}$ denote an optimal algorithm for a cost minimization offline problem $\mathcal{P}$ and assuming that $\mathsf{ALG}(I)$ denotes the cost of any algorithm $\mathsf{ALG}$ when provided with an input $I \in \mathcal{I}$ for $\mathcal{P}$, we have the next definition.

**Definition 1.1.** *The approximation ratio of an offline algorithm* $\mathsf{A}$ *for* $\mathcal{P}$ *is defined as* $R_{\mathsf{A}} = \sup_I \{\frac{\mathsf{A}(I)}{\mathsf{OPT}(I)}\}$*, where* $I$ *ranges over all legal inputs to* $\mathcal{P}$*.*

It holds $R_{\mathsf{A}} \geq 1$, since the cost was defined as a positive real number and

because no algorithm may give a cost function value smaller than the optimal one, for the same input. Finally, we say that the offline algorithm $\mathsf{A}$ is $\alpha$-approximate, if it holds $\alpha \geq R_{\mathsf{A}}$ for an $\alpha \geq 1$.

In the case of a profit maximization offline problem $\mathcal{P}'$, we similarly obtain the next definition, where we assume that $\mathsf{ALG}(I)$ denotes the profit of any algorithm $\mathsf{ALG}$ when provided with a legal input $I$ for $\mathcal{P}'$ and $\mathsf{OPT}$ an optimal algorithm for the same problem:

**Definition 1.2.** *The approximation ratio of an offline algorithm* $\mathsf{B}$ *for* $\mathcal{P}'$ *is defined as* $R'_{\mathsf{B}} = \inf_I\{\frac{\mathsf{B}(I)}{\mathsf{OPT}(I)}\}$*, where* $I$ *ranges over all legal inputs to* $\mathcal{P}'$*.*

It holds $0 < R'_{\mathsf{B}} \leq 1$ since the profit was defined as a positive real number and because no algorithm may provide a profit function value greater than the optimal one, for the same input. Finally, we say that $\mathsf{B}$ is a $\beta$-approximate algorithm, if it holds $\beta \leq R'_{\mathsf{B}}$ for a positive $\beta$.

It is, sometimes, the case to study the performance of an approximation algorithm for a cost minimization problem, solely for inputs for which the optimal cost is very large. For this, we define the *asymptotic* approximation ratio. Therefore, denoting as $\mathsf{OPT}(I)$ the optimal cost for a legal input $I$ to a cost minimization problem $\mathcal{P}$ and as $\mathsf{A}(I)$ the cost of an offline algorithm $\mathsf{A}$ for the same input, we have the next definition:

**Definition 1.3.** *The asymptotic approximation ratio of* $\mathsf{A}$ *for the cost minimization problem* $\mathcal{P}$ *is defined as* $R^{\infty}_{\mathsf{A}} = \limsup_{n\to\infty} \sup_I\{\frac{\mathsf{A}(I)}{\mathsf{OPT}(I)} | \mathsf{OPT}(I) = n\}$*, where* $I$ *ranges over the set of all legal inputs to* $\mathcal{P}$*.*

The analysis of the performance of any algorithm, whether this is offline or online, can be categorized as being average-case analysis or worst-case analysis. The approximation ratio was just defined in terms of the latter approach, as it may be easily observed.

Especially for the case of online computation, on which we shall concentrate from now and until the end of this brief introduction, the worst-case analysis studies the performance of the examined online algorithm towards the performance of an optimal algorithm which is provided with the advantage of having complete information of the input, in advance. Note that such an optimal offline algorithm, as we shall simply refer to it from now, may not be a realizable algorithm, since complete information about the input instance is usually an unrealistic scenario for practical applications of online problems.

On the other hand, the average-case analysis works by first assuming a distribution of inputs to the examined online problem and then, based on this input distribution, the expected performance of the online algorithm is obtained.

A disadvantage of the worst-case analysis is that the comparison with an optimal offline algorithm which is provided with a power that may be unrealistic could underestimate the performance of the online algorithm. On the other hand, an undisputable disadvantage of the average-case analysis is the difficulty in obtaining an input distribution representative of the one that the online algorithm is actually faced with. Obviously, a non-representative input distribution may lead to an erroneous estimation of the actual algorithmic performance, an issue which is easily bypassed by the worst-case analysis, which makes no probabilistic assumptions on the input.

It is not in our scope to further discuss on the advantages and disadvantages of these two approaches. We shall only emphasize on the fact that it is the worst-case analysis that has become the predominant approach of examining the performance of online algorithms and it is known as *competitive analysis* (Karlin et al. [1988]); however, the study of online algorithms in the framework of competitive analysis had already been initiated several years before the influential publication of Karlin et al. [1988].

More specifically, the first implicit result of competitive analysis is considered

to be the paper of Graham [1969], dealing with a simple greedy algorithm for the LOAD BALANCING problem, when all processors are assumed to have the same processing speed. The competitive analysis started to evolve significantly during the 1980's after the publication of the seminal paper of Yao [1980] about the online BIN PACKING problem and, especially, after the publication of Sleator and Tarjan [1985] about two of the most important online problems, the PAGING and the LIST UPDATE problem. Borodin and El-Yaniv [1998] and Fiat and Woeginger [1998] provide excellent surveys regarding the evolution of the competitive analysis.

A question that, again, naturally arises is how the quality of an online algorithm ONL is measured in terms of the competitive analysis. For this, let $cost_{\mathsf{ALG}}(I)$ denote the cost of any algorithm ALG for a cost minimization online problem $\mathcal{P}$, when provided with a legal input $I$ and let OPT denote an optimal offline algorithm for $\mathcal{P}$. Then, we have the next definition.

**Definition 1.4.** *The competitive ratio of* ONL *is defined as* $\inf\{c|cost_{\mathsf{ONL}}(I) \leq c \cdot cost_{\mathsf{OPT}}(I)\}$*, for all legal inputs* $I$ *to* $\mathcal{P}$*.*

It holds $c \geq 1$, since the cost was defined as a positive real number and because no algorithm may give an objective function value smaller than the one that an optimal offline algorithm outputs, for the same input. We say that the online algorithm ONL is $\hat{c}$-competitive for a $\hat{c} \geq 1$, if the competitive ratio of ONL is equal to $c \leq \hat{c}$. If $\hat{c} = c$, we say that ONL is an *exactly* $c$-competitive algorithm. Finally, if the infimum in the above definition is infinity, we shall say that ONL is a *non-competitive* algorithm.

In the case of a profit maximization online problem $\mathcal{P}'$, we can similarly denote as $profit_{\mathsf{ALG}}(I)$ the profit of any algorithm ALG for this problem, when provided with a legal input $I$ and as OPT an optimal offline algorithm for $\mathcal{P}'$. Then, we have the next definition.

**Definition 1.5.** *The competitive ratio of an online algorithm* ONL$'$ *for* $\mathcal{P}'$ *is equal*

*to* $\sup\{p|profit_{\mathsf{ONL}'}(I) \geq p \cdot profit_{\mathsf{OPT}}(I)\}$*, for all legal inputs* $I$ *to* $\mathcal{P}'$*.*

It holds $0 < p \leq 1$, since the profit was defined as a positive real number and because no algorithm may give a profit function value, greater than the one that an optimal offline algorithm outputs for the same input. In a similar way as for the case of a cost minimization problem, we say that the online algorithm $\mathsf{ONL}'$ is $\hat{p}$-competitive for a positive $\hat{p} \leq 1$ if the competitive ratio of $\mathsf{ONL}'$ is equal to $p \geq \hat{p}$. If $\hat{p} = p$, then we shall say that $\mathsf{ONL}'$ is an *exactly* $p$-competitive algorithm. Finally, if the supremum in the above definition is 0, we shall say that $\mathsf{ONL}'$ is a *non-competitive* algorithm.

We shall only employ competitive analysis on any result related to online computation, throughout this thesis.

For the cases of those online problems where we want to study the performance of an online algorithm for a cost minimization problem, only for inputs for which the optimal cost is very large, we have the definition of the *asymptotic* competitive ratio. Hence, assuming that $\mathcal{P}$ is a cost minimization online problem, that $\mathsf{OPT}$ denotes an optimal offline algorithm for this problem and $\mathsf{ALG}(I)$ the cost of any algorithm $\mathsf{ALG}$ for an input $I$, we have the following definition:

**Definition 1.6.** *The asymptotic competitive ratio of an online algorithm* $\mathsf{ONL}$ *for* $\mathcal{P}$ *is defined as* $C^{\infty}_{\mathsf{ONL}} = \limsup_{n\to\infty} \sup_I \{\frac{\mathsf{ONL}(I)}{\mathsf{OPT}(I)} | \mathsf{OPT}(I) = n\}$*, where* $I$ *ranges over all legal inputs to* $\mathcal{P}$*.*

We may refer to the competitive ratio as established in Definition 1.4, as *absolute* competitive ratio, in order to distinguish it from the asymptotic competitive ratio.

It is worth mentioning that an idea of time progression is inherent in the online problems. To make this more perceivable, a sequence of requests is made by the adversary which have to be answered (served) one at a time by the online algorithm (Ben-David et al. [1990]). The procedure of answering the requests of the

adversary continues in a repetitive manner until the last request has been served and the online computation completes.

In the case of deterministic online algorithms, this usually leads us in visualizing the online problem as a game between an online player and an *adversary* who has the power of generating the input sequence. The online player, who is provided with no information on the remaining part of the input at any point in time, runs the online algorithm and outputs a partial solution. The adversary who controls the remaining part of the input, modifies it in such a way that the overall ratio of the performance of the online algorithm to the performance of the optimal offline algorithm, deteriorates for the side of the online player. Fairly enough, the adversary is usually referred to as *malicious* in the literature of online computation (Borodin and El-Yaniv [1998]; Fiat and Woeginger [1998]).

In the case of randomized online algorithms and in order to make a similar discussion to that of the previous paragraph, we have to first define what kind of information the adversary is given about the online algorithm. For this, we shall distinguish between the following two different types of adversaries (Raghavan and Snir [1989]; Ben-David et al. [1990]).

The first type of adversary fixes the request sequence based on the description of the online algorithm. However, the sequence has to be fixed before the online algorithm starts its computation. It follows that this model of adversary has no information on any of the random choices made by the online algorithm. This model of adversary is called *oblivious*.

For this, let $\mathcal{P}$ denote a cost minimization online problem and $\mathsf{OPT}(\sigma)$ be the cost of an optimal offline algorithm serving an input sequence $\sigma$ for $\mathcal{P}$. Then, assuming that $\mathcal{R}\mathsf{ALG}$ is a randomized online algorithm for $\mathcal{P}$, distributed over a set $\{\mathsf{ALG}_y\}$ of deterministic online algorithms for $\mathcal{P}$, we obtain the next definition.

**Definition 1.7.** *The randomized online algorithm* $\mathcal{R}\mathsf{ALG}$ *is c-competitive against an oblivious adversary, if* $\mathbb{E}_Y[\mathsf{ALG}_y(\sigma)] \leq c \cdot \mathsf{OPT}(\sigma) + \gamma$ *for any input sequence* $\sigma$

*to $\mathcal{P}$, where $\gamma$ is an additive constant independent of the input.*

We note that $\mathbb{E}_Y[\mathsf{ALG}_y(\sigma)]$ in Definition 1.7, denotes the expectation with respect to the distribution $Y$, over the set of deterministic online algorithms $\{\mathsf{ALG}_y\}$ which defines the randomized online algorithm $\mathcal{R}\mathsf{ALG}$. Finally, since the online problem $\mathcal{P}$ is a cost minimization problem, we have $c \geq 1$, by the same reasoning as the one which follows Definition 1.4.

Definition 1.7 can be easily modified to hold for the case of a profit maximization online problem $\mathcal{P}'$. More specifically, assuming that $\mathsf{OPT}(\sigma)$ denotes the profit of an optimal offline algorithm serving the input sequence $\sigma$ for $\mathcal{P}'$, we have the following definition.

**Definition 1.8.** *The randomized online algorithm $\mathcal{R}\mathsf{ALG}'$ (distributed over a set $\{\mathsf{ALG}_x\}$ of deterministic online algorithms for $\mathcal{P}'$) is p-competitive against an oblivious adversary, if $\mathbb{E}_X[\mathsf{ALG}_x(\sigma)] + \delta \geq p \cdot \mathsf{OPT}(\sigma)$ for any input sequence $\sigma$ to $\mathcal{P}'$, where $\delta$ is an additive constant independent of the input.*

By the same reasoning as the one which follows Definition 1.5, it holds that $0 < p \leq 1$. We note that $\mathbb{E}_X[\mathsf{ALG}_x(\sigma)]$ denotes the expectation with respect to the distribution $X$, over the set of deterministic online algorithms $\{\mathsf{ALG}_x\}$ that defines the randomized online algorithm $\mathcal{R}\mathsf{ALG}'$.

The second type of adversary that is considered in the relevant literature, is called *adaptive*. Though we shall not deal with adaptive adversaries in this thesis, we proceed towards defining this model of adversary, for consistency.

We distinguish between two types of adaptive adversaries, the *adaptive-online* and the *adaptive-offline* adversary.

The adaptive-online adversary is aware of the random choices made so far by the online algorithm, at any point in time. Based on this information, the adaptive-online adversary serves immediately the current request and modifies the remaining part of the request sequence. For the case of deterministic online algorithms, the

adaptive-online adversary has the same power as the oblivious adversary since the behaviour of the deterministic algorithm is known in advance (Ben-David et al. [1990]).

The adaptive-offline adversary, also, modifies the remaining part of the sequence based on the random choices made so far by the online algorithm but serves it optimally when the online algorithm has completed its computation. It follows that an adaptive-offline adversary is aware of any random choice made by the online algorithm, contrary to the case of the adaptive-online adversary.

The cost (or profit, respectively) of an adaptive adversary for serving a sequence of requests is a random variable, since an adaptive adversary is aware of random choices made by the online algorithm during its computation. This is an important difference compared to the case of an oblivious adversary, who has to fix the sequence in advance having no knowledge on any of the random choices made by the online algorithm.

This complete lack of information regarding any random choice made by the online algorithm is a source of weakness for an oblivious adversary. To state this in a different way, randomization can be exploited to a greater extent by the online player, when he plays against an oblivious adversary rather than against an adaptive-online adversary. On the other side, the adaptive-offline adversary is certainly stronger than the adaptive-online adversary; in fact, it can be shown that randomization cannot be used against an adaptive-offline adversary (Ben-David et al. [1990]).

Concluding, there exist online problems for which randomized online algorithms against oblivious adversaries perform better in expectation, compared to the optimal deterministic online algorithms that we have for these problems. The LIST UPDATE problem, to which we referred to before, is one of these problems (Albers et al. [1995]).

## 1.1 Our Contribution

Before describing the results obtained in this thesis, we need to state some definitions.

A linear program is an optimization problem which asks for either the maximization or the minimization of an objective function of $d \in \mathbb{Z}^+$ variables, subject to a set of inequalities or equalities, which are called *constraints*. The objective function and all constraints have to be linear functions of the given $d$ variables. Since any equality constraint can be substituted by inequality constraints, we usually assume that all constraints of the linear program are inequalities.

A maximization linear program of $d$ variables has the following standard form:

$$\begin{aligned} \max \quad & b_0 x_0 + \ldots + b_{d-1} x_{d-1} \\ \text{subject to} \quad & A \begin{pmatrix} x_0 \\ \vdots \\ x_{d-1} \end{pmatrix} \leq \begin{pmatrix} c_0 \\ \vdots \\ c_{m-1} \end{pmatrix} \\ & x_0, \ldots, x_{d-1} \geq 0 \end{aligned}$$

All components of the vectors $b = (b_0,\ldots,b_{d-1})$ and $c = (c_0,\ldots,c_{m-1})$, as well as all entries of the matrix $A$ are assumed to be real numbers. A minimization linear program has an analogous standard form, where the first $m$ inequalities are reversed and the objective is the minimization of the given linear function.

In case all components of the vectors $b$ and $c$ as well as all entries of the matrix $A$ are non-negative, then the maximization linear program is called as *packing* linear program and the minimization linear program is called as *covering* linear program. Apart from this and most importantly, there exists a systematic way to obtain a covering (or packing) linear program from a given packing (or, respectively, covering)

linear program, where the newly obtained linear program is called as *dual* linear program and the initial linear program is called as *primal* linear program.

Any solution of a linear program which does not violate a constraint, is called *feasible* solution. The *feasible region* is the set of all feasible solutions. The optimal solution of the linear program is a point in the feasible region for which the value of the objective function is maximized or minimized, depending on whether the linear program is a maximization or, respectively, a minimization linear program. Finally, a linear program is *infeasible* if it has no feasible solution, that is the feasible region is empty.

Many NP-hard optimization problems can be formulated as integer linear programs, that is linear programs with the additional constraint that variables may take only integer values. As a consequence, many offline approximation algorithms are based on linear programming, since even though solving an integer linear program is, in general, an NP-hard problem, solving a linear program admits computationally efficient exact algorithms. Hence, taking the relaxation of an integer linear program where the aforementioned restriction on the variable values is dropped, can be used in obtaining computationally efficient approximation algorithms for a great number of NP-hard optimization problems. For this, the primal-dual schema that we previously referred to is widely used, due to some very useful properties which hold for the feasible solutions of the dual linear program (Vazirani [2001]; Williamson and Shmoys [2011]).

Apart from this, quite recently there has been observed a research interest into solving linear programs in an online manner, as well (Buchbinder and Naor [2009b]). This was mainly in order to facilitate the development of improved online algorithms for some central online programs, such as one of the most well-studied and important problems in the area of online computation, the $k$-server problem (Bansal et al. [2010, 2011]).

Hence, in Chapter 2 we study the competitiveness of online algorithms for

the problem that we describe in the following two paragraphs.

First of all, we suppose that we have a packing linear program in the standard form that was described before. We assume that the vector $b$ and all entries of $A$ are initially revealed to the online algorithm by the adversary. However, the adversary gradually reveals the components of the vector $c$ to the online player, i.e., at any point in time $t$ a vector $\ell^t$ is revealed to the online player for which it has to hold $\ell^t \leq c$ component wise. The vector $\ell^t$ may be viewed as the current right-hand side values of the linear program. The online player responds at the current point in time, by increasing the values of variables of his choice whilst ensuring that a feasible solution is maintained. The goal of the online player is to maximize the value of the objective function, when no variable value can be further increased.

A parameter $\alpha > 1$, which is initially revealed to the online player, plays a central role in this online problem, since the adversary is required to ensure at any point in time $t$ that it holds $(c - Ax) \leq \alpha \cdot (\ell^t - Ax)$, where $x$ is the vector denoting the current online solution.

This problem is introduced by Ochel et al. [2012] as an application of lifetime optimization in wireless sensor networks. As we shall see, the right-hand sides of the constraints of the packing linear program may be viewed as lifetimes of batteries that power sensors in wireless networks. Assuming that we can only estimate each remaining battery lifetime within some fixed $\alpha > 1$ approximation of its actual remaining lifetime, the objective of an online algorithm is to choose amongst a set of broadcasting scenarios, so that the number of sensor broadcasts is maximized, until the point in time when the first battery becomes exhausted.

Ochel et al. give a $\Theta(\frac{\ln \alpha}{\alpha})$-competitive deterministic algorithm for this online problem and prove that the competitive ratio of any deterministic or randomized online algorithm for it, is $\mathcal{O}(\frac{1}{\sqrt{\alpha}})$.

We improve the upper bound on the competitive ratio of any online algorithm, whether deterministic or randomized, for this problem, to $\mathcal{O}(\frac{\ln^2 \alpha}{\alpha})$. We, also,

provide an optimal deterministic $\Theta(\frac{1}{\sqrt{\alpha}})$-competitive online algorithm for linear programs that involve two variables.

In Chapter 3 we turn our attention to another packing problem. More specifically, we study a variant of the offline BIN PACKING problem which is called COLORFUL BIN PACKING problem. Recall from our previous discussion on this problem, that the offline BIN PACKING problem has been extensively studied in the relevant literature; one of the reasons is that many of its different settings have a number of practical applications.

Hence, in the COLORFUL BIN PACKING problem each item is additionally associated with a color and two items packed consecutively into the same bin cannot share the same color. This problem was introduced by Balogh et al. [2012] for the special case of two colors, called BLACK AND WHITE BIN PACKING problem. The COLORFUL BIN PACKING problem has been mainly studied from an online perspective (Dósa and Epstein [2014]; Böhm et al. [2014]; Balogh et al. [2015b]).

A motivating application of the BLACK AND WHITE BIN PACKING problem is the optimized distribution of advertisement breaks in television or radio station programs (Balogh et al. [2012]). To see that, note that the bins may be viewed as the program blocks (these usually correspond to one-hour intervals, for the case of radio station programs), the white items correspond to the advertisement breaks and the black items correspond to the actual broadcasted program. The goal is to assign advertisement breaks into the program minimizing the program blocks used.

We provide a 2-approximate algorithm for the offline COLORFUL BIN PACKING problem, which runs in time $\mathcal{O}(n \log n)$, where $n \in \mathbb{Z}^+$ denotes the number of items of the input.

This thesis concludes with the study of an online buffering problem. First of all, it should be acknowledged that the area of packet transmission management is strongly related to that of online computation. This is due to the unpredictability

of transmission requests that is observed in networks, giving the related problems an inherently online character. More specifically, due to the high volume of packet transmissions, some packets have to be discarded; hence, the objective of an online algorithm has to be the minimization of the number of lost packets when the information given in advance regarding the remaining sequence of arriving packets is incomplete.

Therefore, in Chapter 4 we study the Longest Queue Drop (LQD) online algorithm for shared-memory switches with three and two output ports. A shared-memory switch is a device equipped with a buffer and a number of input and output ports. The switches are widely used in directing the packet flow of various networks and the Longest Queue Drop algorithm is a well-known online algorithm used in directing the packet flow of switches (Aiello et al. [2008]).

Each arriving packet to the switch is destined to a single output port of it and can be either accepted or be irreversibly rejected by the switch, upon its arrival time. Assuming that time is divided into time steps, one packet is transmitted by each output port of the switch to which at least one packet stored in the buffer is destined, at the current time step.

According to the Longest Queue Drop online algorithm, any arriving packet is accepted by the buffer if there exists free buffer space at the time step of its arrival. On the contrary, if the buffer is full at the time step when the packet arrives, a packet destined to the output port to which the most packets currently stored in the buffer are destined, is irrevocably rejected from the switch (or it is *preempted*, as we shall say). The preempted packet releases buffer space for the arriving packet, which is immediately accepted by the buffer. After all packet acceptances and preemptions take place, at each time step, one packet is forwarded from the buffer to its destined output port, so that it is transmitted.

We show that LQD is 1.5-competitive for shared-memory switches equipped with three output ports, improving the previously established best upper bound

(Kobayashi et al. [2007]) of 5/3, for three-port switches. We, also, show a tight upper bound of 4/3 for the competitive ratio of LQD for shared-memory switches equipped with two output ports. This corrects upon a previously published result claiming a tight upper bound of $\frac{4M-4}{3M-2} < 4/3$ for the competitive ratio of LQD for two-port shared-memory switches, where $M \in \mathbb{Z}^+$ denotes the buffer size.

# Chapter 2

# Online Packing Linear Programs

## 2.1 Introduction

During the recent years there has been observed an interest into solving linear programs in an online manner. Without doubt, the influential paper of Buchbinder and Naor [2009a] has been the most important work in this area. In this publication, Buchbinder and Naor study a range of online problems via a primal-dual schema that they introduce. Their techniques have triggered applications on subsequent papers, dealing with a variety of central online problems, including one of the most important problems in the field of online computation, the $k$-server problem (Bansal et al. [2011]).

Let us proceed towards describing briefly one of the online problems that have been studied by Buchbinder and Naor [2009a]. The introduced primal-dual schema along with some of its applications on various online problems, is described in detail by its authors in Buchbinder and Naor [2009b].

Buchbinder and Naor assume that incomplete information about the following packing linear program is given to the online player by the adversary:

$$\begin{aligned} \max \quad & b_0 x_0 + \ldots + b_{d-1} x_{d-1} \\ \text{subject to} \quad & A \begin{pmatrix} x_0 \\ \vdots \\ x_{d-1} \end{pmatrix} \leq \begin{pmatrix} c_0 \\ \vdots \\ c_{m-1} \end{pmatrix} \\ & x_0, \ldots, x_{d-1} \geq 0 \end{aligned}$$

More specifically, all components of the vector $c$ are assumed to be initially revealed to the online player by the adversary but the components of the vector $b$ and the entries of the matrix $A$ are gradually revealed to the online player in the following way: The adversary reveals all coefficients of a variable $x_j$ at a time of his choice, i.e., assuming that time is divided into rounds, at round $j$ the coefficient $b_j$ and the $j$-th column of the matrix $A$ are revealed to the online player.

On the other side, the online player may increase the value of any variable at any point in time but never decrease the value of any variable, ensuring that a feasible online solution is always maintained. The goal of the online player is to maximize the value of the objective function $b^T x$.

We focus on a related online problem which has been introduced by Ochel et al. [2012]. The linear program is the packing linear program which has the standard form described above.

We assume that the components of the vector $b$ and the entries of the matrix $A$ (which we shall call as *constraint matrix* from now) are initially revealed to the online player, but the components of the vector $c$ (which we shall call as *capacity vector*) are gradually revealed to the online player by the adversary, in a way that we describe in the next paragraph.

Time is assumed to be discrete. At time $t$ the adversary reveals to the online player a vector $\ell^t$, the components of which can be seen as the current right-hand

sides of the constraints of the linear program. On the other side, the online player responds at $t$, by increasing the values of variables of his choice. The online player may never decrease any variable value and has to additionally ensure that each of the constraints of the linear program is satisfied, i.e., that a feasible online solution is always maintained. The goal of the online player remains the same as in the model of Buchbinder and Naor, that is the maximization of the objective function value.

Finally, the adversary ensures that the following two conditions hold:

- Each component of the revealed vector $\ell^t$ lower-bounds the respective component of the capacity vector $c = (c_0, ..., c_{m-1})$ *and*
- Assuming that $x$ is the current online solution, it holds $(c - Ax) \leq \alpha \cdot (\ell^t - Ax)$ where $\alpha > 1$ is a parameter which is initially revealed to the online player.

We shall denote as *revealed remaining slack* at $t$ the quantity $(\ell^t - Ax)$ and as *true remaining slack* at $t$, the quantity $(c - Ax)$.

We shall study the competitiveness of deterministic and randomized online algorithms in dependence of the parameter $\alpha > 1$, for the aforementioned online problem.

The fact that no variable value may decrease is what makes this problem interesting from an online perspective. This is because, otherwise, the online player would be able to obtain full information about the underlying linear program and hence identify the point of the feasible region which maximizes the value of the objective function.

An interesting application of this online problem, as noted by Ochel et al., refers to the lifetime optimization of wireless sensor networks. A wireless sensor network consists of a number of sensors each of which is powered by its own battery and which broadcasts information wirelessly to neighbour sensors, within its range. All such information is collected wirelessly by a base sensor. The lifetime of such a

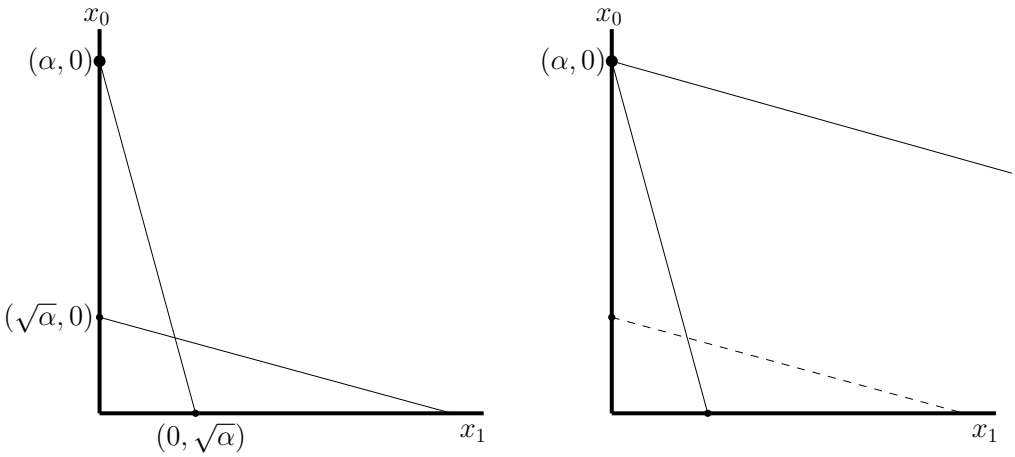


Figure 2.1: In the above example for linear programs with two variables, we assume that the objective function is $x_0 + x_1$. The two constraints initially presented to the online algorithm are $x_0 + \sqrt{\alpha}x_1 \leq \alpha$ and $\sqrt{\alpha}x_0 + x_1 \leq \alpha$ (on the left). However the actual constraints of the linear program are $x_0 + \sqrt{\alpha}x_1 \leq \alpha$ and $\sqrt{\alpha}x_0 + x_1 \leq \alpha\sqrt{\alpha}$ (on the right). The optimal point $(\alpha, 0)$ is on the $x_0$ axis, hence an optimal offline algorithm will only increase the value of variable $x_0$ and leave the value of $x_1$ equal to 0.

network is defined as the time from the first broadcast performed anywhere in the network, until the first battery becomes exhausted.

In this case and assuming that all entries of the matrix $A$ are positive, each component of the right-hand sides of the constraints of the linear program corresponds to the lifetime of one battery powering a network sensor. We assume that we are aware of the lower bounds of the remaining battery lifetimes, but the actual remaining battery lifetimes are supposed to be within a fixed factor $\alpha > 1$ of the revealed battery lifetime values. Given this information, our objective is to choose amongst a set of broadcasting scenarios so that the number of performed network broadcasts is maximized, before a battery becomes exhausted.

Ochel at al. provide a $\Theta(\frac{\ln \alpha}{\alpha})$-competitive deterministic online algorithm for this problem and prove that any online algorithm, whether deterministic or randomized, is $\mathcal{O}(\frac{1}{\sqrt{\alpha}})$-competitive.

Our first result (Englert et al. [2014]) is an improvement of the deterministic

upper bound to $\mathcal{O}(\frac{d^2\alpha^{1/d}}{\alpha})$ for linear programs, having at least $d$ variables. It is important to note here that our upper bound is, also, dependent on the number of variables of the linear program and not only on the parameter $\alpha$.

Our second result (Englert et al. [2014]) is an upper bound of $\mathcal{O}(\frac{m^2\alpha^{1/m}}{\alpha})$ on the competitive ratio of any randomized online algorithm against an oblivious adversary for this problem, for linear programs that involve $m^{\lceil m! \ln \alpha \rceil}$ variables, where there also exists a dependence between the achieved upper bound and the number of variables of the linear program.

Both of the these upper bounds become $\mathcal{O}(\ln^2 \alpha/\alpha)$ for a sufficiently large number of variables.

Finally, we give a simple deterministic $\Theta(\frac{1}{\sqrt{\alpha}})$-competitive algorithm for linear programs that have $d = 2$ variables (Englert et al. [2014]). This algorithm is optimal for the special case of linear programs involving exactly two variables since its competitive ratio matches the upper bound of $\mathcal{O}(\frac{1}{\sqrt{\alpha}})$ that has been established by Ochel et al.

## 2.2 The upper bound for deterministic online algorithms

In the current section, we shall prove the next theorem.

**Theorem 2.1.** *The competitive ratio of any deterministic online algorithm for the problem of packing linear programs with at least $d \geq 2$ variables, is $\mathcal{O}(\frac{d^2\alpha^{1/d}}{\alpha})$.*

Since the bound of Theorem 2.1 is minimized for a number of variables equal to $d = \Theta(\ln \alpha)$, we have the next corollary.

**Corollary 2.2.** *The competitive ratio of any deterministic online algorithm for the problem of packing linear programs, is $\mathcal{O}(\frac{\ln^2 \alpha}{\alpha})$.*

We proceed with the construction of the adversary to prove Theorem 2.1. As a high-level approach, a completely symmetric linear program is initially presented

to the online player by the adversary. Once the online player has increased a variable so much that the revealed remaining slacks of some constraints become sufficiently small, the adversary decides that these constraints are exactly those constraints for which the revealed right-hand sides were already quite close to the true right-hand sides, i.e., to the respective components of the capacity vector $c$. As a consequence, the value of this variable cannot be increased much further in the future and the online algorithm is left, more or less, with a similarly constructed linear program, but for the remaining variables.

Hence, the initial linear program presented to the online algorithm is the following:

$$\begin{aligned} & \max && \sum_{i=0}^{d-1} x_i \\ \forall \text{ permutations } \pi : & && \sum_{i=0}^{d-1} \alpha^{i/d} \cdot x_{\pi(i)} \leq \alpha \\ & && x_0, x_1, \ldots, x_{d-1} \geq 0 \end{aligned}$$

Our construction uses exactly $d$ variables. Theorem 2.1 follows, since any additional variables $x_d, x_{d+1}, \ldots$ can be made irrelevant by adding to the linear program constraints of the form $x_{d+1} \leq 0$, $x_{d+2} \leq 0$, etc.

The adversary maintains a set of *active* constraints and a set of *active* variables. Initially all $d!$ constraints and all $d$ variables are considered to be active. From the point in time when a constraint or a variable becomes inactive, this constraint or variable, respectively, will never become active until the online algorithm terminates.

The adversary proceeds in $d$ rounds, which are numbered downwards as $d-1, d-2, \ldots, 0$. The round $r \in \{0, \ldots, d-1\}$ ends in the first point in time when there exists an active constraint with a revealed remaining slack at most equal to

$\alpha^{r/d}$. At the end of round $r$ each of the next steps takes place in the following order:

- The adversary determines the index $k_r$ of the active variable which has the maximum value among all variables that are currently active, breaking ties arbitrarily.
- The adversary increases the right-hand side (i.e., the respective component of $\ell^t$) of all active constraints that do not correspond to permutations with $\pi(r) = k_r$, to $\alpha \cdot \alpha^{r/d}$. Note that this does not violate the condition that revealed remaining slacks always have to be an $\alpha$-approximation of the true remaining slacks, since all these constraints have a revealed remaining slack which is at least equal to $\alpha^{r/d}$.
- The adversary removes from the set of active constraints all the constraints that their right-hand side was increased in the previous step of the current round.
- The adversary removes the variable with index $k_r$ from the set of active variables and the current round completes.

Before continuing, we note that we may assume that the online algorithm ends its computation after all $d$ rounds are completed, i.e., when no variable is active any more. This is because, for any online algorithm that terminates before round 0 is complete, there exists an other online algorithm that acts exactly as the former online algorithm until the point in time that this former algorithm terminates and which obtains at least the same online profit, since no variable value may decrease.

We start our proof with two easy observations. First of all, recall that at round $r$ we increase the right-hand sides of all active constraints that do not correspond to permutations for which it holds $\pi(r) = k_r$ and we, subsequently, remove these constraints from the set of active constraints. Taking into consideration the fact that the round numbers are decreasing, we obtain the following observation.

**Observation 2.3.** *A permutation $\pi$ corresponds to a constraint active in round $r$ if and only if it holds $\pi(s) = k_s$ for all $s > r$.*

Now, let $x_i^r$ be the value of the variable $x_i$ at the end of round $r$, for $i, r \in \{0, \ldots, d-1\}$. Since for non-negative numbers $\alpha_0 \geq \alpha_1 \geq \ldots$ and non-negative numbers $b_i$, we have that the quantity $\sum_i \alpha_i \cdot b_{\pi(i)}$ is maximized if $b_{\pi(0)} \geq b_{\pi(1)} \geq \ldots$, we obtain the next observation.

**Observation 2.4.** *Let $r = 0, \ldots, d-1$ be a round. Also, let $\pi$ be a permutation corresponding to a constraint active in round $r$, such that it holds*

$$x^r_{\pi(r)} \geq x^r_{\pi(r-1)} \geq \ldots \geq x^r_{\pi(0)}.$$

*Note that such a permutation exists due to Observation 2.3. Then, the constraint corresponding to $\pi$ has the smallest revealed remaining slack among all permutations active in round $r$.*

Now, for any round $r = 0, \ldots, d-1$, let $\pi_r$ denote the permutation corresponding to the constraint that causes round $r$ to end. According to Observation 2.4 we obtain:

$$x^r_{\pi_r(r)} \geq x^r_{\pi_r(r-1)} \geq \ldots \geq x^r_{\pi_r(0)} \tag{2.5}$$

and, therefore, we may assume from now, without loss of generality, that it holds $\pi_r(r) = k_r$.

**Lemma 2.6.** *For any round $r < d-1$ we have*

$$x^r_{\pi_r(i)} \geq x^{r+1}_{\pi_{r+1}(i)}$$

*for any $i = 0, \ldots, d-1$.*

*Proof.* We distinguish between the cases that $i \geq r+1$ and $i \leq r$. We begin with the case that $i \geq r+1$.

Due to Observation 2.3, we have $\pi_r(i) = \pi_{r+1}(i) = k_i$ for any $i > r+1$ since the constraints corresponding to $\pi_r$ and $\pi_{r+1}$ are both active in rounds $d-1, \ldots, r+1$. Since $\pi_r(i)$ is, also, active in round $r$ and it holds $\pi_{r+1}(r+1) = k_{r+1}$ as already argued after (2.5), we have $\pi_r(i) = \pi_{r+1}(i) = k_i$ for $i = r+1$. Taking into consideration the fact that variable values can only increase, this implies for $i \geq r+1$ that it holds $x^r_{\pi_r(i)} = x^r_{k_i} \geq x^{r+1}_{k_i} = x^{r+1}_{\pi_{r+1}(i)}$. This completes the proof for the case that $i \geq r+1$.

Now, assume that $i \leq r$. Since $\pi_r(i) = \pi_{r+1}(i) = k_i$ for $i > r+1$, the sequence $\pi_{r+1}(r+1), \pi_{r+1}(r), \ldots, \pi_{r+1}(0)$ is a permutation of the sequence $\pi_r(r+1), \pi_r(r), \ldots, \pi_r(0)$. Therefore, the sets of variables $\{x_{\pi_{r+1}(r+1)}, \ldots, x_{\pi_{r+1}(0)}\}$ and $\{x_{\pi_r(r+1)}, \ldots, x_{\pi_r(0)}\}$ are identical. Hence, since variable values can only increase, the sequence

$$x^r_{\pi_r(r+1)} \geq x^r_{\pi_r(r)} \geq \ldots \geq x^r_{\pi_r(0)}$$

is obtained from the sequence

$$x^{r+1}_{\pi_{r+1}(r+1)} \geq x^{r+1}_{\pi_{r+1}(r)} \geq \ldots \geq x^{r+1}_{\pi_{r+1}(0)}$$

by increasing the values of one or more variables and rearranging the sequence so that it becomes sorted. Hence, the claim follows for the case that $i \leq r$, completing the proof. □

We can now show the next key lemma.

**Lemma 2.7.** *For any round $r \in \{0, \ldots, d-1\}$ and $i \leq r$, it is $x^r_{\pi_r(i)} \leq (d-r) \cdot \alpha^{1/d}$.*

*Proof.* We shall use downward induction on $r$. The claim is clear for $r = d-1$, since during round $d-1$ all constraints still have right-hand sides equal to $\alpha$ and for any variable $x_i$ there is a constraint that contains this variable with a coefficient equal to $\alpha^{1-1/d}$.

Consider, now, any round $r < d-1$. We can, easily, obtain the following equality:

$$\sum_{i=0}^{d-1} \alpha^{i/d} \cdot x^r_{\pi_r(i)} = \sum_{i=0}^{d-1} \alpha^{i/d} \cdot x^{r+1}_{\pi_{r+1}(i)} + \sum_{i=0}^{d-1} \alpha^{i/d} \cdot \left( x^r_{\pi_r(i)} - x^{r+1}_{\pi_{r+1}(i)} \right) \tag{2.8}$$

The first sum of the right-hand side of (2.8) is lower-bounded by $\alpha - \alpha^{(r+1)/d}$ by the definition of permutation $\pi_{r+1}$. Moreover, by Lemma 2.6 we have $x^r_{\pi_r(i)} \geq x^{r+1}_{\pi_{r+1}(i)}$ for any $i = 0, \ldots, d-1$. Therefore we have the next inequality:

$$\sum_{i=0}^{d-1} \alpha^{i/d} \cdot x^r_{\pi_r(i)} \geq \alpha - \alpha^{(r+1)/d} + \sum_{i=0}^{d-1} \alpha^{i/d} \cdot \left( x^r_{\pi_r(i)} - x^{r+1}_{\pi_{r+1}(i)} \right)$$

with all the terms in the sum of its right-hand side, being non-negative. Since the constraint corresponding to $\pi_r$ is active and feasible in round $r$, the left-hand side of the last inequality is upper-bounded by $\alpha$. By ignoring all terms except the one corresponding to $i = r$ in the sum of its right-hand side we obtain:

$$x^r_{\pi_r(r)} \leq x^{r+1}_{\pi_{r+1}(r)} + \frac{\alpha^{(r+1)/d}}{\alpha^{r/d}} = x^{r+1}_{\pi_{r+1}(r)} + \alpha^{1/d} \leq (d-r)\alpha^{1/d} \tag{2.9}$$

where the last inequality of (2.9) follows from induction.

Overall, we have $x^r_{\pi_r(r)} \leq (d-r)\alpha^{1/d}$. By this and (2.5), we obtain the lemma for $x^r_{\pi_r(i)}$ when $i < r$. □

Now, let us denote as $x'_i$ (where $i = 0, \ldots, d-1$) the final value of variable $x_i$, that is the value of $x_i$ when the online algorithm terminates. Due to the next lemma, we can upper-bound the online profit which equals $\|x'\|_1$, where $\|.\|_1$ denotes the $L_1$-norm and $x'$ denotes the vector $(x'_0, ..., x'_{d-1})$.

**Lemma 2.10.** *It holds $x'_i = \mathcal{O}(d\alpha^{1/d})$ for any $i = 0, \ldots, d-1$.*

*Proof.* The last round performed is round $r = 0$, which means that there exists a constraint for which the right-hand side is set equal to $\alpha \cdot \alpha^{0/d} = \alpha$ at the end of this round, i.e., this constraint stays as is. Let $\pi$ denote the permutation corresponding

to this constraint.

Note that no variable is active at the end of round 0 and each variable has an index $k_j = \pi_j(j) = \pi(j)$ for some $j \geq 0$. We have:

$$\sum_{i=0}^{d-1} \alpha^{i/d} \cdot x'_{\pi(i)} = \sum_{i=0}^{d-1} \alpha^{i/d} \cdot x^j_{\pi_j(i)} + \sum_{i=0}^{d-1} \alpha^{i/d} \cdot \left( x'_{\pi(i)} - x^j_{\pi_j(i)} \right) \tag{2.11}$$

According to the definition of $\pi$, we obtain the next inequality as in the proof of Lemma 2.7:

$$\sum_{i=0}^{d-1} \alpha^{i/d} \cdot x^j_{\pi_j(i)} \geq \alpha - \alpha^{j/d} \tag{2.12}$$

Taking into consideration the fact that the sum in the left-hand side of (2.11) is at most $\alpha$, since the right-hand side of this constraint is not altered and stays equal to $\alpha$, we have from (2.11) and (2.12), similarly to the reasoning in the proof of Lemma 2.7:

$$x'_{k_j} \leq x^j_{k_j} + \frac{\alpha^{j/d}}{\alpha^{j/d}} \leq (d-j)\alpha^{1/d} + 1.$$

where we used Lemma 2.7 in order to upper-bound $x^j_{k_j}$ in the last inequality. This completes the proof. □

In the next lemma, we bound the profit that an optimal offline algorithm can achieve.

**Lemma 2.13.** *An optimal offline algorithm can obtain a profit of $\alpha$.*

*Proof.* An offline algorithm can set the value of the variable $x_{k_0}$ equal to $\alpha$ and each of the other variable values equal to 0. We shall now show that this is a feasible solution.

For any $i \geq 1$, consider any of the $(d-1)!$ constraints in which $x_{k_0}$ has a coefficient equal to $\alpha^{i/d}$. In other words, a constraint corresponding to a permutation with $\pi(i) = k_0$. Such a constraint becomes inactive by the end of round $i$ the latest,

since $\pi(i) = k_0 \neq k_i$. Due to the strategy of the adversary, this means that, once the constraint becomes inactive, the right-hand side increases to $\alpha \cdot \alpha^{i/d}$ and therefore the constraint is satisfied.

The constraints in which the coefficient of $x_{k_0}$ is 1 are clearly satisfied as well, since the right-hand side of all constraints is at least $\alpha$. □

By combining Lemmas 2.10 and 2.13 we obtain Theorem 2.1. This completes the analysis for the competitive ratio upper bound for the case of deterministic online algorithms.

## 2.3 The upper bound for randomized online algorithms

The upper bound on the competitive ratio of randomized online algorithms against oblivious adversaries is based on the construction from Section 2.2. Recall that, according to the adversary construction from this section, each round ends when the revealed remaining slack of at least one active constraint drops below a certain threshold value. Then, the adversary identifies the active variable having the greatest value and increases appropriately the right-hand sides of specific constraints. The identified variable becomes inactive from this point and on and, hence, its value cannot be further increased much, until the online algorithm completes its computation.

To obtain our upper bound on randomized algorithms, we shall use Yao's principle, which is an elegant and very useful tool in obtaining bounds for the competitive ratio of any randomized online algorithm against oblivious adversaries. Since an oblivious adversary is weaker than any adaptive adversary, as already mentioned in Chapter 1, these bounds hold for randomized online algorithms against any type of adversary.

The first application of Yao's principle, related to theoretical computer science, was, quite reasonably, due to Yao [1977]. However, the origin of this principle

is, in fact, much older and can be traced back to the Minimax Theorem of von Neumann [1928].

Assuming that $\mathcal{P}$ is a cost minimization online problem, letting $\{\mathsf{ALG}_n\}$ denote the set of deterministic online algorithms for $\mathcal{P}$ (where $n \in N$) and assuming that $\mathsf{OPT}$ is an optimal offline algorithm for $\mathcal{P}$, then Yao's principle states the following:

**Theorem 2.14.** *If $Z$ is a probability distribution over a set of input instances $\sigma_x$ to $\mathcal{P}$ and there exists a $c \geq 1$ such that it holds $\inf_{n \in N} \mathbb{E}_Z[\mathsf{ALG}_n(\sigma_x)] \geq c \cdot \mathbb{E}_Z[\mathsf{OPT}(\sigma_x)]$, then $c$ lower-bounds the competitive ratio of any randomized online algorithm against an oblivious adversary, for $\mathcal{P}$.*

For the case of a profit maximization online problem $\mathcal{P}'$ and assuming that $\{\mathsf{ALG}_l\}$ (where $l \in L$) denotes the set of deterministic online algorithms for this problem whilst $\mathsf{OPT}$ denotes an optimal offline algorithm for it, Yao's principle can be stated as in the following:

**Theorem 2.15.** *If $Z$ is a probability distribution over a set of input instances $\sigma_x$ to $\mathcal{P}'$ and there exists a positive $p \leq 1$ such that it holds $\sup_{l \in L} \mathbb{E}_Z[\mathsf{ALG}_l(\sigma_x)] \leq p \cdot \mathbb{E}_Z[\mathsf{OPT}(\sigma_x)]$, then $p$ upper-bounds the competitive ratio of any randomized online algorithm against an oblivious adversary, for $\mathcal{P}'$.*

In simple words, Yao's principle enables us instead of proving bounds for randomized online algorithms, to construct an input distribution which, in expectation, foils any deterministic online algorithm for the same online problem. Since for the case of randomized algorithms, it is usually more difficult to bound the expected cost (or, respectively, profit) compared to the case of deterministic online algorithms, Yao's principle simplifies analyses substantially.

Now, let us turn to our problem of packing linear programs. The analysis of the previous section involves a constant interaction between the online player and the adversary, since a value increment of any variable by the online player is followed

by an update of the right-hand sides of specific constraints of the linear program, by the adversary.

Recall from the introductory Chapter 1 that in order to define oblivious adversaries, we need to allow the adversary to fix his complete behavior in advance; it follows that we have to remove this interaction between the adversary and the online player.

For this, the input will consist, as before, of a packing linear program, whose constraints are given by $Ax \leq c$. Additionally, for each constraint $i \in \{0, \ldots, m-1\}$, the adversary specifies a monotonically increasing function $\ell_i(\lambda_i)$ of the left-hand side of the constraint $\lambda_i := (Ax)_i$. This function models the right-hand side of the constraint in dependence of the current value of the left-hand side. This function has to satisfy equality $\ell_i(\lambda_i) = c_i$ for $\lambda_i \geq c_i$ and inequality $c_i - \lambda_i \leq \alpha \cdot (\ell_i(\lambda_i) - \lambda_i)$ for $\lambda_i < c_i$.

If $\lambda_i$ is the current value of the left-hand side of the $i$-th constraint, then the online player is aware of all values of $\ell_i(z)$ for $z \leq \lambda_i$ but is not aware of any values for $z > \lambda_i$.

Note that we only need basic threshold functions to construct the deterministic upper bound from the previous section. For this, we define functions $f_j(\lambda)$, for $j \in \{0, 1, \ldots, d-1\}$, as

$$f_j(\lambda) := \begin{cases} \alpha \ , & \lambda < \alpha - \alpha^{j/d} \\ \alpha^{1+j/d} \ , & \lambda \geq \alpha - \alpha^{j/d} \end{cases}$$

The linear program is the same as the one in the previous section. The monotonically increasing function assigned to a constraint that corresponds to permutation $\pi$, is $f_r$ if $r$ is the largest integer such that $\pi(r) \neq k_r$, where $k_r$ is the index of the largest active variable at the end of round $r$.

This way we can set up a predefined behavior of the adversary, as required

in order to apply Yao's principle.

Now, in the construction of the adversary of the previous section, the order in which the variables become inactive defines a permutation $\pi_o$ of the set $\{0, \ldots, d-1\}$. In other words, if the variable with index $k_r$ becomes inactive in round $r$ we have $\pi_o(r) = k_r$. Consider an adversary that acts in exactly the same way as before, but he guesses the permutation $\pi_o$ and proceeds as if the variables would actually become inactive in this order.

If the adversary guesses correctly, the construction works as intended and the algorithm can only obtain a value of $\mathcal{O}(d^2\alpha^{1/d})$, while the optimal value is $\alpha$. If, however, the adversary chooses the permutation uniformly at random, then the success probability is $1/(d!)$, that is with this probability, the same upper bound as in the previous section is achieved. However, if the adversary guesses incorrectly the algorithm can perform better than $\mathcal{O}(d^2\alpha^{1/d})$ and even obtain a value of $\alpha$, while the optimal value remains $\alpha$.

We are not going to analyze the average performance of an algorithm in the setting described in the previous paragraph. Instead, we will increase the success probability for the adversary from $1/(d!)$ to almost $1 - 1/\alpha$. This is done in a similar way to the one that Ochel et al. provide their upper bound for randomized online algorithms against oblivious adversaries. More specifically, we work as in the following.

We assume that we have $K$ adversaries with random $\pi_o$ sequences working in parallel and require the online algorithm to beat all of them, for some sufficiently large $K$. Hence, suppose we want to have $K$ $d$-dimensional adversaries. We construct a packing linear program with $d^K$ variables $\{x_{i_1,\ldots,i_K} | 0 \leq i_1, \ldots, i_K \leq d-1\}$. The objective function is the sum of all variables.

For the $k$-th adversary, we add $d!$ constraints to the linear program. The constraints have the same form as the ones in the previous section but instead of variables $x_j$ the variables are:

$$x_j^k := \sum_{(i_1,\ldots,i_K):i_k=j} x_{i_1,\ldots,i_K} \tag{2.16}$$

For each adversary $k \in \{1,\ldots,K\}$, a permutation $\pi^k$ of the set $\{0,\ldots,d-1\}$ is chosen independently and uniformly at random and the functions $f_r$ are randomly assigned to constraints based on this permutation, as described earlier.

For any adversary $k$, the objective function is:

$$\sum_{0\le i_1,\ldots,i_K\le d-1} x_{i_1,\ldots,i_K} = \sum_{j=0}^{d-1} x_j^k \tag{2.17}$$

Therefore, the objective function is also equal to $\min_k \sum_{j=0}^{d-1} x_j^k$.

We allow the online player to increase the value of $x_j^k$ directly instead of increasing the values of the underlying $x_{i_1,\ldots,i_K}$ variables. Note that, in reality, an algorithm cannot increase the $x_j^k$ variables completely independently of each other, since increasing one of the underlying $x_{i_1,\ldots,i_K}$ variables always affects multiple $x_j^k$ variables. However, allowing the online player to directly and individually increase $x_j^k$ variables can only provide him with more power.

This completes the construction of an input that combines $K$ adversaries from the previous section, each of them independently guessing a random order in which variables become inactive. The profit of the online algorithm against one of the adversaries is bounded by $\mathcal{O}(d^2\alpha^{1/d})$ with probability $\frac{1}{d!}$ (that is, if the adversary guessed the correct permutation) and by $\alpha$ with probability $(1-\frac{1}{d!})$.

The total profit of the online algorithm is bounded by the minimum profit the algorithm achieves against any of the $K$ adversaries, since the objective function equals $\min_k \sum_{j=0}^{d-1} x_j^k$. Hence, by choosing $K = \lceil d! \ln \alpha \rceil$ we get that the expected overall profit of the online algorithm is bounded by:

$$\left(1-\frac{1}{d!}\right)^K \alpha + \left(1-\left(1-\frac{1}{d!}\right)^K\right)\mathcal{O}(d^2\alpha^{1/d}) = \mathcal{O}(d^2\alpha^{1/d}) \tag{2.18}$$

At the same time, the optimal profit is always $\alpha$. To see this set $x_{\pi_1(0),\ldots,\pi_K(0)} = \alpha$ and all other variables to 0. Consider the constraints that belong to the $k$-th adversary. Then $x_j^k$ is equal to $\alpha$ for $j = \pi^k(0)$ and 0 otherwise. This is exactly the feasible solution from Lemma 2.13, applied to the constraints of the $k$-th adversary. Taking into consideration the fact that the constructed linear program has $d^K$ variables, this gives us the next theorem.

**Theorem 2.19.** *The competitive ratio of any randomized online algorithm against an oblivious adversary is* $\mathcal{O}(\frac{m^2\alpha^{1/m}}{\alpha})$ *for linear programs involving* $m^{\lceil m! \ln \alpha \rceil}$ *variables.*

By the same reasoning as the one following Theorem 2.1, we have the next corollary.

**Corollary 2.20.** *The competitive ratio of any randomized online algorithm against an oblivious adversary for the problem of packing linear programs, is* $\mathcal{O}(\frac{ln^2\alpha}{\alpha})$.

This completes the analysis for the competitive ratio upper bound, for the case of randomized online algorithms.

## 2.4 An optimal online algorithm for linear programs with two variables

In this section, we give a deterministic $\Theta(\frac{1}{\sqrt{\alpha}})$-competitive algorithm for packing linear programs involving two variables, $x_0$ and $x_1$. As ALG we shall denote this algorithm and as OPT an optimal offline algorithm for this problem, until the end of Chapter 2. We shall, also, use ALG and OPT to refer to the total profit of these algorithms, respectively.

The algorithm ALG is optimal for the special case of linear programs involving two variables, since its competitive ratio matches the competitive ratio upper bound of $\mathcal{O}(\frac{1}{\sqrt{\alpha}})$.

First of all, it would be easier for the analysis to concentrate solely on the objective function $x_0+x_1$, instead of the more general form $b_0x_0+b_1x_1$, with positive $b_0$ and $b_1$. For this, the algorithm initially normalizes the variables by dividing each entry $a_{ij}$ of the matrix $A$ by $b_j$, where $j \in \{0,1\}$. This does not change the profit of an optimal solution. To see that, note that an increment of the value of $x_i$ by an amount $\epsilon > 0$ in the normalized linear program, corresponds to exactly an increment of the value of $x_i$ by $\epsilon/b_i$ in the initial linear program and both of these increments, only increase the objective function value of the respective linear program, by the same amount, which is $\epsilon$.

Now, let $x^*(z) = (x_0^*(z), x_1^*(z))$ denote an optimal solution of the linear program where the right-hand sides of the constraints are given by the vector $z$. Let $(x_0, x_1)$ be the current online solution and let $\ell_j^t$ denote the $j$-th component of the vector $\ell^t$ at $t$, where $j \in \{0, \ldots, m-1\}$.

At this time $t$, ALG does the following:

---

1. If any constraint is tight then ALG stops, *else*
2. If $x_0^*(\ell^t) > x_0 \cdot \gamma$, then ALG increases variable $x_0$ for an infinitesimal amount, *else*
3. If $x_1^*(\ell^t) > x_1 \cdot \gamma$, then ALG increases variable $x_1$ for an infinitesimal amount, *else*
4. ALG stops.

---

We, now, proceed towards establishing the competitiveness of ALG.

**Lemma 2.21.** *For $\gamma = 1 + 1/\sqrt{\alpha}$, it holds* $\mathsf{ALG} \geq \frac{1}{\sqrt{\alpha}+1} \cdot \mathsf{OPT}$.

*Proof.* Let $x' = (x_0', x_1')$ indicate the point on the plane, where ALG stops. Since ALG will either stop due to Step 1 or due to Step 4, we distinguish between the following two cases:

**ALG stops due to Step 1.**

In this case, there has to exist at least one tight constraint, by the definition of the algorithm. Let us pick arbitrarily one of these tight constraints and write it as $a_0x_0 + a_1x_1 \leq \ell_k^t$, where $k \in \{0, \ldots, m-1\}$. Since $(c - Ax) \leq \alpha \cdot (\ell^t - Ax)$, this implies that it holds:

$$a_0x_0' + a_1x_1' = c_k \tag{2.22}$$

Assume, without loss of generality, that it is $a_0 \geq a_1$. Then, the optimal profit can be at most $c_k/a_1$.

For every $t$, $x^*(\ell^t)$ has to satisfy $a_0x_0^*(\ell^t) + a_1x_1^*(\ell^t) \leq \ell_k^t \leq c_k$. Therefore, we obtain $x_0^*(\ell^t) \leq c_k/a_0$. Additionally, since ALG increases variable $x_0$ only if $x_0 \cdot \gamma < x_0^*(\ell^t) \leq c_k/a_0$, it has to hold $x_0' \leq c_k/(a_0 \cdot \gamma)$. By this and due to (2.22), we obtain that $a_1x_1' = c_k - a_0x_0' \geq (1 - 1/\gamma) \cdot c_k$.

Therefore, the profit of ALG has to be at least equal to $(1-1/\gamma)\cdot c_k/a_1$. Since the optimal profit is at most $c_k/a_1$ (as already argued) and with $\gamma = 1 + 1/\sqrt{\alpha}$, we obtain $\mathsf{ALG} \geq \frac{1}{\sqrt{\alpha}+1} \cdot \mathsf{OPT}$ .

**ALG stops due to Step 4.**

In this case and due to the choice of the stopping condition, we have $x_0^*(\ell^t) \leq x_0' \cdot \gamma$ and $x_1^*(\ell^t) \leq x_1' \cdot \gamma$. Adding these two inequalities together, we have $\|x' \cdot \gamma\|_1 \geq \|x^*(\ell^t)\|_1$.

Since it holds $(c - Ax) \leq \alpha \cdot (\ell^t - Ax)$, we obtain $\ell^t \geq ((\alpha - 1) \cdot Ax + c)/\alpha$. Hence, this gives us the following:

$$\begin{aligned}\|x' \cdot \gamma\|_1 &\geq \|x^*(\ell^t)\|_1 \\ &\geq \Big\|x^*\Big(\frac{(\alpha-1)Ax'+c}{\alpha}\Big)\Big\|_1 \\ &\geq \Big(1-\frac{1}{\alpha}\Big)\|x^*(Ax')\|_1 + \frac{\|x^*(c)\|_1}{\alpha} \\ &\geq \Big(1-\frac{1}{\alpha}\Big)\|x'\|_1 + \frac{\|x^*(c)\|_1}{\alpha} \ .\end{aligned}$$

The second inequality, above, follows from the fact that $\ell^t \geq ((\alpha-1)Ax + c)/\alpha$; therefore the feasible region that is defined by setting the capacity vector to $((\alpha-1)Ax+c)/\alpha$ is enclosed by the feasible region defined by setting the capacity vector to $\ell^t$. The last inequality follows from the fact that the point $x'$ is a feasible solution if the capacity vector is set equal to $Ax'$.

Solving for $x'$ gives $\|x'\|_1 \geq \|x^*(c)\|_1/(\alpha\gamma-\alpha+1)$ where the right-hand side is equal to $\|x^*(c)\|_1/(\sqrt{\alpha}+1)$ for $\gamma = 1+1/\sqrt{\alpha}$, showing again that $\mathsf{ALG} \geq \frac{1}{\sqrt{\alpha}+1}\cdot\mathsf{OPT}$. This completes the proof. □

By Lemma 2.21, it follows that our algorithm is $\Theta(\frac{1}{\sqrt{\alpha}})$-competitive and, therefore, is optimal for linear programs involving two variables.

## 2.5 Open Problems

We significantly narrowed the gap between the upper bound of $\mathcal{O}(\frac{\ln^2\alpha}{\alpha})$ and the lower bound of $\Omega(\frac{\ln\alpha}{\alpha})$ for the competitive ratio of any (deterministic or randomized) online algorithm for the problem of packing linear programs. However, it remains an open question to further narrow this gap, by establishing better online algorithms or by providing improved upper bounds. At this point, we conjecture that the simple greedy algorithm which we describe next, is better than $\Theta(\frac{\ln\alpha}{\alpha})$-competitive.

Let $z_s^t = \min_j \frac{\ell_j^t}{A_{j,s}}$ for a variable $s \in \{0,\ldots,d-1\}$ for which the respective entry $A_{j,s}$ is non-zero. Then, the aforementioned deterministic online algorithm,

does the following:

---

1. Let $t$ be the current point in time. Choose the variable $i$ at $t$, for which it holds $z_i^t = \max_s z_s^t > 0$ (breaking ties arbitrarily) *else if* $\max_s z_s^t = 0$ then Stop.
2. Increase the value of variable $i$ for an infinitesimal amount.
3. Repeat Step 1 for the next point in time.

---

Our established bounds for deterministic and randomized online algorithms suggest that it is interesting to study the influence of the number of variables of the linear program, on the achievable competitive ratio. We would be interested in bounds that are tight for any fixed number of variables and not just when the number of variables of the linear program is very large.

This is further emphasized by the fact that when the number of variables $d$ is very large, there seems to be little difference in the power of randomized and deterministic online algorithms. However, taking $d$ into consideration for the particular interesting case of moderately large values of $d$, randomization has the potential to improve performance, compared to determinism.

It is, for instance, easy to verify that the simple randomized online algorithm that picks uniformly at random one of the $d$ variables and increases the value of this variable until a constraint becomes tight is $(\frac{1}{d})$-competitive. Taking, for example, a number of variables $d = 2$, this gives an improvement over the $\Theta(\frac{1}{\sqrt{\alpha}})$ competitive ratio that deterministic algorithms can achieve for the case of linear programs involving two variables.

# Chapter 3

# The COLORFUL BIN PACKING problem

## 3.1 Introduction

The BIN PACKING problem is one of the most well-studied optimization problems, from both an offline and an online perspective. This is mainly due to the large number of practical applications that many of the variant problems of BIN PACKING have. Another reason is, perhaps, that many well-performing algorithms for this problem, being quite natural and simple to implement, gained the attention of the scientific community, already from the very beginning when the areas of approximation algorithms for NP-hard optimization problems and that of competitive analysis, started to form.

In the current chapter we start with a brief overview of the research, for both the offline and the online BIN PACKING problem. We continue discussing about a variant of BIN PACKING, the COLORFUL BIN PACKING problem, for which we design a fast 2-approximate algorithm. Concluding, we give an overview of the open problems related to the offline and the online COLORFUL BIN PACKING problem.

Let us first concentrate on the offline BIN PACKING problem, providing the

definition of an asymptotic polynomial-time approximation scheme for a cost minimization offline problem $\mathcal{P}$:

**Definition 3.1.** *An asymptotic polynomial-time approximation scheme (or asymptotic PTAS) for $\mathcal{P}$ is a family of computationally efficient algorithms $\{\mathsf{A}_\epsilon\}$, such that for each fixed positive constant $\epsilon$, the algorithm $\mathsf{A}_\epsilon$ outputs a solution of cost at most equal to $(1+\epsilon)\cdot \mathsf{OPT}(I) + c$, for any input $I$.*

In Definition 3.1, it is assumed that $c$ is a positive constant independent of the input. In case it is $c = 0$, then we refer to the family of algorithms $A_\epsilon$ as *polynomial-time approximation scheme* (or PTAS, in short). Finally, in the case the running time of an asymptotic PTAS is polynomial in $(1/\epsilon)$ as well, then we shall refer to the respective family of algorithms as *asymptotic fully polynomial-time approximation scheme* (or AFPTAS, in short).

Recall the definition of the offline (one-dimensional) BIN PACKING problem:

**Definition 3.2.** *Given a set of items, each of which has a size in $(0, 1]$, we are asking what is the minimum number of unit-capacity bins, that we can pack all items into.*

As already stated, there exist many variant problems of BIN PACKING. A straightforward generalization are the multidimensional bin packing problems, such as the two-dimensional BIN PACKING problem (Bansal and Khan [2014]) where we assume that each item corresponds to a rectangle of a width and a height, each being at most equal to 1 and the three-dimensional BIN PACKING problem where an item depth in $(0, 1]$ is additionally introduced (Martello et al. [2000]). In the latter case, a bin corresponds to a rectangular solid of unit volume and an item corresponds to a rectangular solid of height, width and length, each at most 1.

A few other variants include the BIN PACKING with cardinality constrains where the number of items in each bin should be bounded (Epstein and Levin [2010]) and the BIN PACKING with color constraints where each item has a color and where the number of different colors in a bin must not exceed a given number (Dawande

et al. [2001]). Also, the BIN PACKING with conflicts where conflicts are introduced between packed items (Jansen and Öhring [1997]) and the BIN PACKING with rejection where a rejection penalty is associated with each item, which contributes to the total algorithmic cost in case the specific item is chosen not to be packed into any bin (Epstein [2006]).

It is straightforward to show that some natural heuristics perform quite well for the one-dimensional BIN PACKING problem. As an example, consider the simple algorithm First-Fit, which packs an item into any already opened bin that the item fits into, otherwise it opens a new bin to pack this item.

For any input of items, it holds that at most one bin may be packed with a total item size at most 1/2, in the output of First-Fit. This is because, otherwise First-Fit due to its design, would be able to pack all items placed into any pair of bins each of which is packed with a total item size at most 1/2, into a single bin. Taking into consideration the next observation, it follows that First-Fit is a 2-approximate algorithm:

**Observation 3.3.** *Let $\ell \geq 1$ be the number of bins that are packed with a total item size strictly greater than 1/2, in the output of* First-Fit. *Then, the number of bins of an optimal packing is at least $(\ell + 1)/2$.*

*Proof.* If First-Fit packs $\ell$ bins with a total item size strictly greater than 1/2, then the total item size should be strictly greater than $\ell/2$. Since any algorithm can pack items of total size at most 1 in each bin, then an optimal algorithm has to use more than $\ell/2$ bins. Taking into consideration the fact that the number of packed bins must be an integer number, we obtain the observation. □

The algorithm First-Fit has been shown to have an approximation ratio equal to 1.7 (Dósa and Sgall [2013]). The variant of this heuristic, called First-Fit Decreasing, which first sorts the items in descending order of sizes and then applies First-Fit on it, has been shown to have an approximation ratio equal to 1.5 (Simchi-Levi

[1994]). This is the best possible approximation ratio that any computationally efficient algorithm for the BIN PACKING problem can have, unless it holds $P = NP$. To see that, we state the following definition of PARTITION which is a well known NP-complete problem:

**Definition 3.4.** *We are given a set $S$ of $n \geq 2$ positive integers $s_1, ..., s_n$, where $\sum_{j=1}^{n} s_j = 2 \cdot k$, for some $k \in \mathbb{N}$ and we are asking whether it holds $\sum_{j \in S'} s_j = \sum_{j \in S \setminus S'} s_j$ for any $S' \subseteq S$.*

If a polynomial-time algorithm for BIN PACKING with an approximation ratio $(3/2) - \epsilon$ (for any constant $\epsilon > 0$) were possible, then we could decide PARTITION in polynomial time. This is because we could reduce PARTITION to an instance of BIN PACKING, where for each integer $s_i$ ($i \in \{1, \ldots, n\}$) we have an item of a size equal to $2 \cdot s_i / \sum_{j=1}^{n} s_j$. This gives us the next observation:

**Observation 3.5.** *There does not exist a PTAS for the* BIN PACKING *problem, unless $P = NP$.*

The question on whether there exists an asymptotic PTAS for BIN PACKING was answered affirmatively by Fernandez de la Vega and Lueker [1981]. They gave an approximation algorithm for the BIN PACKING problem that always outputs a solution of a number of bins upper-bounded by $(1 + \epsilon) \cdot \mathsf{OPT} + \mathcal{O}(\epsilon^{-2})$, for any fixed $\epsilon > 0$. We note that a variant of the algorithm presented by Fernandez de la Vega and Lueker [1981], gives a solution of a number of bins upper-bounded by $(1 + \epsilon) \cdot \mathsf{OPT} + 1$. The idea behind this can be, roughly, described as in the following:

- Each item having a size at most equal to the fixed $\epsilon > 0$ is excluded from the input $I$, so that we obtain a subset of items $I' \subseteq I$.
- The sizes of the items in $I'$ are adjusted so that a constant number of different item sizes is obtained. Also, an optimal packing for this modified set of items is obtained, in polynomial time in the number of different item sizes.

- The optimal packing of the previous step is modified so that a packing of the items in $I'$ is derived and, finally, First-Fit is applied for the excluded items of the first step.

A celebrated approximation algorithm was derived in the seminal paper of Karmarkar and Karp [1982], producing a number of bins upper-bounded by $\mathsf{OPT}+\mathcal{O}(\log^2(\mathsf{OPT}))$. Recently, Rothvoß [2013] gave an improved approximation algorithm, which produces an output that has a number of bins at most equal to $\mathsf{OPT}+\mathcal{O}(\log(\mathsf{OPT})\cdot\log(\log(\mathsf{OPT})))$.

Let us, now, focus on the online BIN PACKING problem. The (one-dimensional) online BIN PACKING problem may be viewed as a variant of the LOAD BALANCING problem, where we have an infinite number of processors each of unit processing speed (corresponding to the bins) and a sequence of tasks each of which has a processing time requirement at most equal to 1 (corresponding to the items), which must be irrevocably assigned to the processors. The goal in this case is the minimization of the used processors.

As in the case of the offline BIN PACKING, there exist many variant problems of the online BIN PACKING problem, the most notable of which are, perhaps, its multidimensional variants.

One of the first results regarding the competitiviness of online algorithms for the online BIN PACKING, was that First-Fit has an asymptotic competitive ratio equal to 1.7 (Ullman [1971]). Apart from this, First-Fit is an exactly 1.7-competitive algorithm, that is the asymptotic competitive ratio of First-Fit matches the absolute competitive ratio of this algorithm. This follows by Dósa and Sgall [2013] since First-Fit needs no information on the remaining part of the input. The online algorithm with the best asymptotic competitive ratio that we currently have for the online BIN PACKING problem has an asymptotic competitive ratio upper-bounded by 1.58889 and it is due to Seiden [2002]. The latter result involves several complex arguments, generalizing upon a simple though clever idea that appeared several years before, in

Lee and Lee [1985].

Recently, a (5/3)-competitive algorithm was derived (Balogh et al. [2015c]) for the online BIN PACKING problem and this algorithm is optimal, since its competitive ratio matches the universal lower bound for the competitive ratio of any deterministic algorithm for the online BIN PACKING problem. Finally, it has been shown by Balogh et al. [2010] that no online algorithm with an asymptotic competitive ratio smaller than 1.54037 may exist for this problem, improving upon the lower bound of 1.54014 that had been established several years before by van Vliet [1992]. Closing the gap between the universal lower bound of 1.54037 and the upper bound of 1.58889, for the asymptotic competitive ratio of the online BIN PACKING, is an important open problem.

From this point and until the end of Chapter 3, we shall turn our attention solely on a recently introduced variant problem of the BIN PACKING problem, the COLORFUL BIN PACKING problem.

In the COLORFUL BIN PACKING problem each item has a color and for the packing to be valid, two consecutive items packed into the same bin (i.e., one item above the other, in the same bin) cannot have the same color. As it is the case for the one-dimensional BIN PACKING problem, each bin is assumed to have a unit capacity.

The offline COLORFUL BIN PACKING problem becomes equivalent to the one-dimensional BIN PACKING problem, if no two items in the input have the same color and this, also, shows that this problem is NP-hard. The special case in which there are only two different colors is known as BLACK AND WHITE BIN PACKING and has been introduced by Balogh et al. [2012].

Balogh et al. [2012], also, point out that three different settings can be considered for the BLACK AND WHITE BIN PACKING problem and, consequently, for the COLORFUL BIN PACKING problem. The first setting is the online setting in which items arrive one by one and an online algorithm has to pack them into bins

in their arriving order, without having any information about items arriving in the future. The second setting is the offline setting in which items, again, arrive one by one and have to be packed into bins in that order, but decisions can be based on the entire input. This setting is called "restricted offline". The third setting is, also, an offline setting, but items are presented as an unordered set. This setting is called "unrestricted offline".

The difference between the restricted offline and the unrestricted offline setting is that in the former, the items in a bin, from bottom to top, have to form a subsequence of the input sequence, whereas the latter setting does not have such a restriction. Note that in the COLORFUL BIN PACKING problem, the order in which items are packed into a bin, i.e., in what order they are stacked on top of each other, matters. For this reason the two offline settings are not equivalent. We shall only consider the latter offline setting for our approximation algorithm.

Another way of formulating this problem is that the input consists of a set of items and that the algorithm produces a partition of this set into subsets such that, for each of these subsets both of the following conditions have to hold:

- The sum of the item sizes in the subset does not exceed 1 *and*
- If the subset contains $w \in \mathbb{Z}^+$ items, then at most $(w+1)/2$ of them may share the same color.

A motivating application of the BLACK AND WHITE BIN PACKING problem, as mentioned by Balogh et al., is the optimized distribution of advertisement breaks in television or radio station programs. In this case, the bins may be viewed as the program blocks (these usually correspond to one-hour intervals, for the case of radio stations), the white items correspond to the advertisement breaks and the black items correspond to the actual broadcasted television or radio program. The objective is the broadcasting of a given set of advertisements into as few program blocks as possible.

Balogh et al. [2015b] study the BLACK AND WHITE BIN PACKING problem in the online setting. They give a 3-competitive algorithm and show that the standard heuristics of the BIN PACKING problem First-Fit, Best-Fit (which packs an item into any bin that the item fits, leaving the least empty space) and Worst-Fit (which packs an item into any bin that the item fits, leaving the most empty space) have a competitive ratio which is lower-bounded by 3. Of course these algorithms are modified in the case of the BLACK AND WHITE BIN PACKING problem, so that each packing follows the color restriction.

Böhm et al. [2014] show that the competitive ratio of First-Fit, Best-Fit and Worst-Fit is exactly 3. Balogh et al. [2015b], also, show a lower bound of 1.7213 on the competitive ratio for any deterministic online algorithm for the BLACK AND WHITE BIN PACKING problem. Dósa and Epstein [2014] improve this universal lower bound to 2.

Balogh et al. [2015a] investigate the unrestricted offline version of BLACK AND WHITE BIN PACKING and provide a $\mathcal{O}(n \log n)$-time 2-approximate algorithm, where $n$ denotes the number of items of the input. Our algorithm matches this running time and approximation ratio but works for any number of colors. In this paper, Balogh et al., also provide an asymptotic PTAS for the unrestricted offline BLACK AND WHITE BIN PACKING and improve this further to get a fully asymptotic PTAS, which has a worse additive error but an improved running time dependency on $\epsilon$. The asymptotic PTAS can also be used to get an approximation ratio of 3/2, but at the cost of a significantly worse running time than the 2-approximate algorithm.

The COLORFUL BIN PACKING problem with more than two colors has only been studied in the online setting. Dósa and Epstein [2014] give a 4-competitive online algorithm for this problem. Böhm et al. [2014] improve on this, providing a 3.5-competitive online algorithm as well as a lower bound of 2.5 on the competitive ratio of any deterministic algorithm.

Dósa and Epstein [2014], also, note that going from two to three or more

colors makes the problem harder in a certain sense. For inputs in which all items have a zero size, there exists an optimal (1-competitive) online algorithm for two colors, whereas no such algorithm exists if the input contains three colors or more. Concluding, for an arbitrary number of colors, but with all items having a zero size, Böhm et al. [2014] give a (5/3)-competitive algorithm, which is optimal for the case that all items have a zero size.

We provide a 2-approximate algorithm for the COLORFUL BIN PACKING problem, running in time $\mathcal{O}(n \log n)$, where $n \in \mathbb{Z}^+$ denotes the number of items of the input.

## 3.2 An approximation algorithm for the COLORFUL BIN PACKING problem

### 3.2.1 Preliminaries

For our analysis, we shall call an item that has a size which is at most equal to 1/2 as *small* and an item which has a size strictly greater than 1/2 as *large*. We shall call a bin of a total packed item size at most equal to 1/2 as *light* and a bin of a total packed item size strictly greater than 1/2 as *heavy* bin.

Let $c$ denote the color of which we have the greatest number of small items in the input breaking ties arbitrarily. Let $c'$ denote the color of which we have the second greatest number of small items in the input breaking ties arbitrarily. Here and in the remainder, we shall assume some arbitrary fixed tie-breaking rule between colors.

In the first step of the algorithm we try to pair large items with small items of color $c$ and pack these pairs together into bins. In the second step of the algorithm, we place all remaining large items into separate bins. More precisely, the first two steps of our algorithm are the following:

1. We compute a maximum matching between small items each of which has color $c$ and large items each of which has a color other than $c$. Two such items may be matched, if their combined size is at most 1. Let $m \in \mathbb{Z}^+$ denote the size of this matching. Also, let $d \geq 0$ denote the difference between the number of items of color $c$ and the number of items of color $c'$. We open $\min\{m, d\}$ bins and pack a matched pair into each of these bins. The small item of each pair becomes the bottom item in the bin and the large item is packed on top of it.

2. We open a new bin for each of the remaining large items, i.e., for all large items of color $c$ and (possible) unassigned large items of different colors.

The third step of our algorithm proceeds in rounds numbered $1, 2, \ldots$. In each of these rounds we pack a remaining (small) item into a bin. For this, no previously opened bins are used. For the description of this third step, it would be convenient to use the following notation:

- Let $n_i(x)$ denote the number of small items each of which has color $x$, that are not yet packed at the start of round $i$.

- Let $\alpha_i := \arg\max_x n_i(x)$ denote the color of which we have the greatest number of small items left unpacked at the start of round $i$.

- Let $\beta_i := \arg\max_{x \neq \alpha_i} n_i(x)$ denote the color of which we have the second greatest number of small items left unpacked at the beginning of round $i$.

The following gives a precise description of a round $i$ in the third step of the algorithm. Note that we only start a new round $i$ if there are still items left unassigned and hence it is $n_i(\alpha_i) > 0$:

3. At every point in time, there is one active bin. Initially in round 1, this is a new empty bin.

- If the top item in the active bin is not of color $\alpha_i$ (or the bin is empty), we add an item of color $\alpha_i$ if there is one such item that fits into the bin. If no item of color $\alpha_i$ fits into the bin then we open a new bin which becomes the new active bin and we add an item of color $\alpha_i$ to this new bin.
- Otherwise, if the top item in the active bin is of color $\alpha_i$, we add an item of color $\beta_i$ if there is one that fits into the bin. If no item of color $\beta_i$ fits into the bin, then we open a new bin which becomes the new active bin and we add an item of color $\alpha_i$ to it.

In the fourth and final step of the algorithm, we try to merge some of the bins:

4. We compute a maximum matching between all bins that have an item of color $c$ at the bottom and an item of a color other than $c$ at the top and all light bins that have items of color $c$ at both the bottom and top. A pair of these bins may be matched if their total size is at most 1, that is if they fit into the bin together. We merge the matched bins by putting the items from the light bin that has both items of color $c$ at its top and bottom, on top of the items of the other bin.

We shall denote our algorithm as ALG. Before continuing, we proceed towards establishing the running time of ALG: Each matching can be computed in $\mathcal{O}(n \log n)$ time. To see that, for the matching of Step 1, assume that both sets of small items of color $c$ and large items of a color other than $c$, are ordered according to the item size. If a small item of color $c$ matches with a large item of a different color, any large item of a color other than $c$ of a smaller size than the latter item, also, matches with the former item. Hence, selecting pairs of items in a greedy way, no augmenting paths are obtained in the resulting matching[1]. The same idea works

[1]By Berge's Lemma (Berge [1957]), it follows that the matching is maximum.

for each computed matching of the algorithm.

Regarding the third step of the algorithm, we arrange all small items that have not been packed at the end of Step 2, in descending order of $n_1(x)$, where our arbitrary fixed tie-breaking rule between colors is used. We define the quantity $\gamma_i = \arg\max_{x \neq \alpha_i, x \neq \beta_i} n_i(x)$ and maintain a priority queue at each round $i \leq n$ of the third step, according to $n_i(x)$, for all colors. The priority queue is modified at round $i \geq 2$ if $n_i(\alpha_{i-1}) < n_i(\beta_{i-1})$ (or $n_i(\beta_{i-1}) < n_i(\gamma_{i-1})$), by deleting element corresponding to color $\alpha_{i-1}$ (or $\beta_{i-1}$) from the queue and reinserting it back into it, with its new priority which can be efficiently found using binary search, in $\mathcal{O}(\log n)$ time.

Finally, all other computations performed during any step of the algorithm require $\mathcal{O}(n \log n)$ time, as it can be easily verified; therefore the algorithm performs in $\mathcal{O}(n \log n)$ time.

### 3.2.2 Analysis

We denote an optimal algorithm for the COLORFUL BIN PACKING problem as OPT. We shall, also, use OPT and ALG to refer to the total cost of an optimal algorithm and of our algorithm, respectively.

We start by formally proving a rather intuitive lemma about the bins opened in the third step of the algorithm.

**Lemma 3.6.** *If $n_t(\alpha_t) = n_t(\beta_t)$ at some round $t$ of the third step of* ALG*, then from the active bin in round $t$ and all bins opened afterwards, at most one will remain a light bin in* ALG*'s final output.*

*Proof.* We will frequently use the fact that for two different colors $x$ and $x'$, we always have that $n_i(\beta_i) \geq \min\{n_i(x), n_i(x')\}$, for any round $i$.

We show that the following two invariants hold, for any $i \geq t$, using induction:

1. $n_i(\alpha_i) - n_i(\beta_i) \leq 1$ and

2. If $n_i(\alpha_i) - n_i(\beta_i) = 1$ then the item packed at round $i-1$ does not have color $\alpha_i$.

Clearly, since $n_t(\alpha_t) = n_t(\beta_t)$, both invariants hold at $i = t$. Now, suppose that they hold for some $i \geq t$. We shall show that they, also, hold for $i+1$ by distinguishing between the following four cases.

**Case 1:** It holds $n_i(\alpha_i) - n_i(\beta_i) = 1$ and in round $i$ we pack an item of color $\alpha_i$. Note that it is $n_i(\alpha_i) \neq n_i(\beta_i)$. That is, there is only one color of which we have $n_i(\alpha_i)$ small items and the number of small items of any other color is strictly less. Therefore, we obtain $n_{i+1}(\alpha_{i+1}) = n_i(\alpha_i) - 1 = n_i(\beta_i)$.

Since $\alpha_i$ and $\beta_i$ are two different colors, this gives us that $n_{i+1}(\beta_{i+1}) \geq \min\{n_{i+1}(\alpha_i), n_{i+1}(\beta_i)\} = \min\{n_i(\alpha_i) - 1, n_i(\beta_i)\} = n_i(\beta_i)$. Combining these two facts, we get $n_{i+1}(\alpha_{i+1}) - n_{i+1}(\beta_{i+1}) \leq 0$. It follows that both invariants are satisfied for $i+1$.

**Case 2:** It holds $n_i(\alpha_i) - n_i(\beta_i) = 1$ and in round $i$ we pack an item which has color $\beta_i$.

Due to the second invariant, ALG would be able to pack an item of color $\alpha_i$ in round $i$. But, by design, ALG would have done so. Therefore this case cannot occur.

**Case 3:** It holds $n_i(\alpha_i) - n_i(\beta_i) = 0$ and in round $i$ we pack an item of color $\alpha_i$. We have $\alpha_{i+1} = \beta_i \neq \alpha_i$ and therefore the second invariant is satisfied for $i+1$.

On one hand, it holds $n_{i+1}(\alpha_{i+1}) = n_{i+1}(\beta_i) = n_i(\beta_i)$. On the other hand, it is $n_{i+1}(\beta_{i+1}) \geq \min\{n_{i+1}(\alpha_i), n_{i+1}(\beta_i)\} = \min\{n_i(\alpha_i) - 1, n_i(\beta_i)\} = \min\{n_i(\beta_i) - 1, n_i(\beta_i)\} = n_i(\beta_i) - 1$. Combining these two facts, we obtain the first invariant for $i+1$.

**Case 4:** It holds $n_i(\alpha_i) - n_i(\beta_i) = 0$ and in round $i$ we pack an item of color $\beta_i$. We have $\alpha_{i+1} = \alpha_i$ but we pack an item of a different color, namely $\beta_i$. Therefore

the second invariant is satisfied for $i+1$.

On one hand, it holds $n_{i+1}(\alpha_{i+1}) = n_{i+1}(\alpha_i) = n_i(\alpha_i)$. On the other hand, it is $n_{i+1}(\beta_{i+1}) \geq \min\{n_{i+1}(\alpha_i), n_{i+1}(\beta_i)\} = \min\{n_i(\alpha_i), n_i(\beta_i) - 1\} = \min\{n_i(\alpha_i), n_i(\alpha_i) - 1\} = n_i(\alpha_i) - 1$. Combining these two facts, we obtain the first invariant for $i+1$.

At this point, we have established our two invariants and we can proceed towards showing the lemma. Suppose that ALG has opened a bin. In round $i$, there are three possible reasons why ALG would leave a bin light and either terminate Step 3 or open a new bin:

1. None of the unpacked items fits into the bin because the total size of the bin would exceed one.
2. There are no unpacked items left.
3. Some items are still left, but all of them have the same color (as the top item in the bin), i.e., it is $n_i(\beta_i) = 0$.

In the first case, the bin cannot be light, since we only have unpacked small items at this point. But if the bin was light then all of these items would fit.

The second case can only happen once, in the last round and therefore can only contribute one light bin in total.

Finally, the third case cannot happen. Due to our two invariants we know that in this case it is $n_i(\alpha_i) = 1$ and the item packed at round $i-1$ (which is the current top item of the bin) does not have color $\alpha_i$. Therefore, ALG could pack the last item of color $\alpha_i$ into the bin. □

We, now, strengthen the result obtained by the previous lemma:

**Lemma 3.7.** *If $n_t(\alpha_t) = n_t(\beta_t)$ for a round $t$ of Step 3 of* ALG*, then from all bins opened during Step 3 of the algorithm, at most one will remain a light bin in* ALG*'s final output.*

*Proof.* We first argue that any bin that becomes inactive before the first round $t$ for which it holds $n_t(\alpha_t) = n_t(\beta_t)$ is heavy in ALG's final output. Assume for contradiction that there is a bin $B$ that is not heavy. Consider the round $i < t$ when $B$ was active but no item could be added to it and a new bin is opened. We know that it holds $n_i(\alpha_i) > n_i(\beta_i) > 0$. But ALG could neither place a (small) item of color $\alpha_i$ nor a (small) item of color $\beta_i$ into the bin although one of these two colors must be different than the current top item color in the bin. Hence, the total size of the packed items in the bin must exceed $1/2$ and the bin should be heavy.

Due to Lemma 3.6, any bin, apart from possibly one, that becomes inactive after the first round $t$ for which it holds $n_t(\alpha_t) = n_t(\beta_t)$, is heavy as well. □

We now distinguish between two cases.

**Case 1:** $n_t(\alpha_t) = n_t(\beta_t)$ for some round $t$ in Step 3.

Due to Lemma 3.7, at most one light bin is created in the third step of ALG. All bins from the first two steps contain a large item and are, therefore, heavy as well. From these two facts, combined with Observation 3.3 which is easy to see that it holds for ALG in this case, we have that $\mathsf{ALG} \leq 2 \cdot \mathsf{OPT}$ which completes analysis for Case 1.

**Case 2:** For all rounds $t$ in Step 3, it holds $n_t(\alpha_t) > n_t(\beta_t)$.

Note that in this case, the most frequent color $c$ for small items remains the most frequent color throughout all rounds $t$, i.e., it is $\alpha_i = c$ for all rounds $i$. We begin with an easy observation.

**Observation 3.8.** *All bins in the output of* ALG *either start with an item of color $c$ or consist of a single large item having a different color than $c$.*

*Proof.* All bins created in the first step and the third step of ALG start with an item of color $c$. All bins created in the second step of ALG only contain a single large

item. In the fourth step of ALG, only bins created in the first step and the third step are merged, all of which start with an item of color $c$. Therefore, the merged bin still starts with an item of color $c$. □

According to Observation 3.8, we can categorize the bins in the final output of ALG into the following four different types:

- $X_1$: bins containing no item of color $c$.
- $X_2$: bins with a bottom item of color $c$ and a top item not of color $c$.
- $X_3$: heavy bins with a bottom and a top item of color $c$.
- $X_4$: light bins with a bottom and a top item of color $c$.

In fact, it is easy to see that only bins of type $X_4$ can be light:

**Observation 3.9.** *$X_1$ and $X_2$ bins are heavy.*

*Proof.* By Observation 3.8, we have that each $X_1$ bin has to be heavy.

Now, regarding the $X_2$ bins, assume for contradiction that there exists some $X_2$ bin $B$ that is not heavy. The bin $B$ must have been created during the third step of ALG, since each bin created in the first and the second step of ALG contains a large item and, hence, it is a heavy bin.

Therefore, consider the round $i$ when $B$ was active but no item could be added to this bin, so that a new bin was opened by ALG. Since a new bin was opened, it follows that there existed an unpacked item, that is it has to be $n_i(\alpha_i) > 0$. Also, the top item in $B$ did not have color $c = \alpha_i$, otherwise the bin would not be of type $X_2$.

Hence, the reason why we could not place a (small) item of color $c$ into bin $B$ must be that the size of such an item combined with the items already packed into the bin would exceed 1. Therefore the bin should be heavy. □

Next we argue that due to the design of our algorithm and the fact that we are in the Case 2, in which throughout all rounds of the third step of ALG it holds $c = \alpha_t$, we have that every other item in a bin must be of color $c$:

**Observation 3.10.** *The $X_2$, $X_3$ and $X_4$ bins have the property that every other item in the bin is of color $c$.*

*Proof.* Due to Observation 3.8 all $X_2$, $X_3$, and $X_4$ bins start with color $c$. Bins created in the first and the second step of ALG contain at most two items and therefore every other item has color $c$.

In the third step, it holds $\alpha_t = c$ for all rounds $t$. Therefore, whenever the top item in the active bin is not of color $c$, ALG places an item of color $c$ next. Hence, all bins created during the third step, also have the property that every other item in each of them, has color $c$.

In the fourth step, two bins are merged that both start with color $c$ and in which every other item is of color $c$. Therefore, it follows that the new merged bin also has every other item of color $c$. □

The following observation is an easy consequence from Observation 3.10 and the definitions of $X_2$, $X_3$ and $X_4$ types of bins.

**Observation 3.11.** *Regarding the $X_2$ bins, the number of $c$-colored items in the bin is equal to the number of items not of color $c$. For the $X_3$ and $X_4$ bins, the number of $c$-colored items in the bin is equal to the number of items not of color $c$, plus 1.*

Let $x_1$, $x_2$, $x_3$, and $x_4$ be the number of bins of type $X_1$, $X_2$, $X_3$, and $X_4$, respectively. We distinguish between the following two sub-cases of Case 2:

**Case 2.1:** $x_4 \leq x_2$.

If we combine any $X_2$ and any $X_4$ bin, we have that their total size together must be greater than 1. This is because otherwise it would have been feasible to combine the two bins in the fourth step of the algorithm.

Therefore, any subset of size $x_4$ of the bins of type $X_2$ combined with all bins of type $X_4$ must have items of total size greater than $x_4$. Due to Observation 3.9, all remaining bins, i.e., all $X_3$ and $X_1$ bins and $x_2 - x_4$ of the $X_2$ bins, are heavy and therefore have a total size which is at least equal to $(x_3 + x_1 + (x_2 - x_4))/2$. In total, the sum of the sizes of all items must be more than $x_4 + (x_3 + x_1 + (x_2 - x_4))/2 = (x_3 + x_2 + x_1 + x_4)/2 = \mathsf{ALG}/2$. Since $\mathsf{OPT}$ can pack items of total size at most 1 in each bin, we have $\mathsf{ALG} \leq 2 \cdot \mathsf{OPT}$ which completes the analysis for sub-case 2.1.

**Case 2.2:** $x_4 > x_2$.

Say $\mathsf{OPT}$ uses $g \geq 0$ bins that do not contain any item of color $c$ and $h \geq 0$ bins that contain at least one item of color $c$.

First, note that in the first step of our algorithm it holds $m \leq d$. This is because since otherwise, at the beginning of the third step, it should have been $n_1(\alpha_1) = n_1(\beta_1)$ which cannot happen for Case 2.

Second, we observe that $g \geq x_1$. Suppose for contradiction that this is not true and let $y$ denote the total number of large items that do not have color $c$. Then $y - g > y - x_1$ of these items must be together in a bin with a small item that has color $c$. Therefore, in the first step, $\mathsf{ALG}$ would be able to construct a matching of size greater than $y - x_1$. However, the size of the maximum matching is at most $y - x_1$; to see this, note that due to Observation 3.8 all bins in the output of $\mathsf{ALG}$ which have no item of color $c$ have to be packed with only one large item (of color other than $c$). This is a contradiction.

Finally, we turn our attention towards the bins created by $\mathsf{OPT}$ that do contain items of color $c$. Suppose that a bin contains $i$ items of color $c$ and $j$ items of a color different than $c$. Then, if this bin is validly packed, it has to be $i \leq j + 1$. Now, consider the $h$ bins of $\mathsf{OPT}$ that do contain an item of color $c$. Taking the sum over all these $h$ bins we get that the total number $k$ of items of color $c$ must be at most $h$ plus the total number $r$ of items not of color $c$ in these bins. In other words, we have $h \geq k - r$. Overall, we obtain that $\mathsf{OPT} = g + h \geq x_1 + k - r$.

We, now, need a final observation to complete the analysis for sub-case 2.2:

**Observation 3.12.** *If $k$ is the number of items of color $c$, then the number of items not of color $c$ is $k - x_3 - x_4 + x_1$.*

*Proof.* This follows from the fact that (i) all $X_1$ bins contain exactly one item not of color $c$, (ii) $X_2$ bins contain an equal number of items of color $c$ and items not of color $c$, and (iii) $X_3$ and $X_4$ bins each contain one more item of color $c$ than items not of color $c$. Here (i) follows from Observation 3.8 and (ii) and (iii) follow from Observation 3.11. □

Since $\mathsf{OPT} \geq x_1 + k - r$ and the total number of items not of color $c$ is at least $r + x_1$, i.e., items not of color $c$ that share a bin with an item of color $c$ and items not of color $c$ that do not, Observation 3.12 gives us $\mathsf{OPT} \geq x_1 + k - (k - x_3 - x_4) = x_1 + x_3 + x_4 > x_3 + (x_2 + x_4)/2 + x_1 \geq (x_3 + x_2 + x_4 + x_1)/2 = \mathsf{ALG}/2$, which completes the analysis for sub-case 2.2 and, therefore, for Case 2, giving us the following theorem.

**Theorem 3.13.** ALG *is a 2-approximate algorithm for the* COLORFUL BIN PACKING *problem.*

## 3.3 Open Problems

Starting from the offline COLORFUL BIN PACKING problem, an open problem is the design of an approximation scheme. Recall that there exists a fully asymptotic PTAS for the special case of two colors, i.e., for the offline BLACK AND WHITE BIN PACKING problem, from Balogh et al. [2015a].

Regarding the online COLORFUL BIN PACKING problem, the main open problem is to obtain an online algorithm which will be better than 3.5-competitive, improving upon the online algorithm that we have from Böhm et al. [2014] or to improve upon the universal lower bound of 2.5, that we, also, have from Böhm et al.

[2014]. However, even for the case of two colors, the 3-competitive algorithm of Balogh et al. [2012] remains the best deterministic online algorithm that we have for this problem, since the introduction of the BLACK AND WHITE BIN PACKING problem in this publication.

# Chapter 4

# Online Buffer Management

## 4.1 Introduction

The area of buffer management is strongly related to that of online computation due to the unpredictability of future requests that naturally arises in related problems. Therefore, the fact that a great deal of research has been dedicated to online algorithms which improve the performance of networking devices that incorporate buffers, comes as no surprise.

A switch is a networking device which is equipped with a set of input ports and a set of output ports. Assuming that time proceeds in discrete time steps, packets arrive at any time step to the input ports of the switch. Each arriving packet is labelled with an output port through which it may be only transmitted, assuming that the packet is accepted by the switch first. Each accepted packet has to be buffered until its transmission time step in the input ports, in the output ports or in some internal part of the switch, depending on the model of the switch that we consider.

An online algorithm has no knowledge of the future packet arrivals, contrary to an optimal offline algorithm which is aware of the entire incoming packet sequence upfront. The objective of an online algorithm is to maximize the total number of

transmitted packets or the total value of the transmitted packets in the case the packets are assigned with values, which correspond to their transmission priorities.

In the case the packets are assigned with values then we say that the network considers Quality of Service (QoS), a term that refers to the overall quality of the provided services to the users of the network such as packet loss rate due to the increasing traffic that a network may encounter (Aiello et al. [2000]; Kesselman et al. [2001]; Lotker and Patt-Shamir [2002]). For this, the packets assigned with higher values are considered more important for transmission and, therefore, are prioritized in order to be transmitted before packets assigned with lower values.

There exists a great variety of different switch models and a large number of different online policies that are used to improve their performance. For an excellent survey on the topic of buffer management policies for network switches, the reader may be referred to a survey by Goldwasser [2010].

The algorithms for network switch problems can be categorized as being either, *preemptive*, in which case an eviction of an already accepted packet by the switch is possible or as *non-preemptive* in which case each accepted packet has to be transmitted by the switch at some point in time.

In the current chapter we shall specifically focus on shared-memory switches with multiple input and multiple output ports. A shared-memory switch is comprised by a buffer of size $M \in \mathbb{Z}^+$, a number of $N$ input ports and a number of $N$ output ports, where $N \geq 2$. Any arriving packet may be either immediately accepted by the switch or be immediately rejected by it upon its arrival time. Each arriving packet is designated a single output port through which it may be only transmitted. Any rejected packet is irreversibly lost and any accepted packet is stored in the buffer. At any time step, one packet is transmitted by each output port of the switch, to which at least one packet currently stored in the buffer is destined.

Since there may exist at most $N$ different types of packets stored in the buffer

of the shared-memory switch at any time, we may assume that the packets stored in the buffer which are labelled with the same output port, are arranged in their own queue; that is we may have at most $N$ queues in the buffer at any time step. Finally, we may assume for ease of analysis, that instead of $N$ input ports there exists a single input port of a very large capacity, capable of storing any arriving packet at each arrival time step, before the decision is made on whether the arriving packet will be accepted by the buffer or will be rejected.

Kesselman and Mansour [2004] give a $(\ln(N)+2)$-competitive non-preemptive online algorithm for the problem of maximizing the number of transmitted packets from a shared-memory switch with $N$ output ports. Their algorithm, which is called Harmonic, sorts the queues at any time in order of decreasing length and divides the buffer size according to fractions that are proportional to $(1/i)$, where $i$ refers to the $i$-th longest queue in the buffer. Kesselman and Mansour, also, show a lower bound of $\Omega(\frac{\log N}{\log \log N})$ on the competitive ratio of any online deterministic non-preemptive algorithm for this problem, assuming all arriving packets have unit values.

On the other side, a preemptive online algorithm for the same problem is the Longest Queue Drop (LQD) algorithm. According to LQD, any incoming packet is accepted by the buffer if there exists free buffer space at the time step when this packet arrives. If the buffer is full, however, at the packet arrival time step then a packet is preempted from the longest queue in the buffer at this point in time, releasing buffer space for the incoming packet which is accepted. After all packet acceptances, preemptions and rejections take place at each time step, one packet from each queue is being sent to the output ports of the switch in order to be transmitted. The same procedure is followed at every time step until the last packet stored in the buffer is transmitted.

The LQD algorithm has been introduced by Wei et al. [1991]. Hahne et al. [2001] show a lower bound of 4/3 for the competitive ratio of any deterministic online algorithm, for the problem of maximizing the number of transmitted packets from

a shared-memory switch equipped with an arbitrary number of output ports. It has been, also, shown (Hahne et al. [2001]; Aiello et al. [2008]) that LQD is 2-competitive and at least $\sqrt{2}$-competitive.

Let us focus on the LQD competitive ratio upper bound of 2 for a while, since we shall follow a similar technique for our analysis as the one that Hahne et al. [2001] employ. The way that the LQD competitive ratio is upper-bounded by 2 is by assigning connections between packets accepted by the buffer of an optimal offline algorithm OPT and packets accepted by the buffer of the LQD algorithm, for the same incoming packet sequence. More specifically, Hahne et al. [2001] show that for each packet transmitted by a queue in the OPT buffer at a time step when this queue has no packets in the LQD buffer, the LQD algorithm has, already, transmitted one packet at some previous time step. After establishing the fact that each transmitted packet from the LQD buffer is never connected with two or more transmitted packets from the OPT buffer, it follows that the number of additional packets that OPT may transmit, compared to LQD, is upper-bounded by the number of packets that LQD transmits. Hence, LQD is a 2-competitive algorithm.

Kobayashi et al. [2007] claim a tight upper bound of $\frac{4M-4}{3M-2} < \frac{4}{3}$ for the LQD competitive ratio for two-port shared-memory switches, though the correct tight upper bound when $N = 2$ is $4/3$ as we prove in Section 4.2.4. In Kobayashi et al. [2007] an upper bound of $(2 - \frac{1}{M} \cdot \lfloor M/N \rfloor - \frac{N}{M})$ is, also, shown for the LQD competitive ratio. For the case of three-port switches, this bound is larger than $3/2$ when $M \geq 19$ and tends to $5/3$ when $M \to \infty$.

All the aforementioned results assume that the incoming packet sequence is comprised solely by packets of the same value. Apart from this, they assume that each packet has a unit processing cycle in the buffer; if this latter restriction is dropped and we assume that the packet processing times are heterogeneous then a non-constant lower bound for the competitiviness of LQD has been shown (Eugster et al. [2014, 2015]).

We show that LQD is (3/2)-competitive for three-port shared-memory switches (Matsakis [2016], to appear) when each arriving packet has the same value. This improves upon the previously established best upper bound of 5/3 for three-port switches. Many of the techniques that we shall employ as well as many of the lemmas and corollaries that we shall later derive, still hold for shared-memory switches equipped with an arbitrary number of output ports $N \leq M$. Hence, part of our analysis could be used in order to improve the LQD competitive ratio upper bound of $2 - o(1)$ for shared-memory switches equipped with any number of output ports $N \leq M$; in fact, we conjecture that LQD is a $(2 - \epsilon)$-competitive algorithm for a positive constant $\epsilon$.

We, also, provide a tight upper bound of 4/3 for the competitive ratio of LQD for two-port shared-memory switches, when each arriving packet has the same value. The latter result has been, already, studied in the past by Kobayashi et al. [2007], as already mentioned.

## 4.2 The LQD algorithm

We assume that the number of output ports $N$ is either 2 or 3 and that $M \geq N$, throughout the following analysis of Chapter 4. All results derived in Sections 4.2.1 and 4.2.2, however, hold for shared-memory switches equipped with any number of output ports $N \leq M$, starting from the optimal offline algorithm that we define in Section 4.2.1.

### 4.2.1 Preliminaries

#### The Optimal Offline Algorithm

Let ALG denote any (offline or online) algorithm for the problem of maximizing the number of transmitted packets from a shared-memory switch, from this point and until the end of Chapter 4. Also, let OPT denote an optimal offline algorithm for

the same problem.

Both LQD and OPT are assumed to use their own buffers, each of size $M$. Time proceeds in discrete time steps and $T \geq 1$ will denote the latest time step when a packet exists in any buffer. Analysis will proceed by comparing the two buffer contents at the same time steps. As $[n]$ we shall denote the set of positive integers $\{1, ..., n\}$ for a $n \in \mathbb{Z}^+$.

A packet accepted by the ALG buffer will be called ALG packet, in short. All definitions that follow in the next paragraph, refer to the buffer content of ALG, after all packet acceptances, rejections, preemptions and transmissions have taken place at a time step $t \in [T]$.

We denote as $p^t_{i,\mathsf{ALG}}$ the length of a queue $i \in [N]$ in the ALG buffer at a time step $t \in [T]$. A queue $i \in [N]$ is said to be *active* in the ALG buffer at a time step $t \in [T]$ if $p^t_{i,\mathsf{ALG}} > 0$ else if $p^t_{i,\mathsf{ALG}} = 0$ then $i$ is *inactive* in the ALG buffer at $t$. As $\ell^t_{\mathsf{ALG}}(q) \in [M]$, we denote the queue position of a packet $q$ in the ALG buffer, at a time step $t \in [T]$.

Let $t_1 \in [T]$ and $t_2 \in [T]$ be two time steps such that $t_1 \leq t_2$. As *period* we call the set of consecutive time steps from $t_1$ (inclusive) to $t_2$ (inclusive) and we denote the period as $[t_1, t_2]$.

If a packet labelled with a queue $i \in [N]$, is rejected or preempted from the LQD buffer at a time step $t \in [T]$, we say that $i$ *overflows* in the LQD buffer at $t$. At any time step when the LQD buffer overflows, this buffer is full, according to the definition of the LQD algorithm.

**Observation 4.1.** *The lengths of two queues in the* LQD *buffer that overflow in this buffer at the same time step $t \in [T]$, may differ by at most 1 at $t$.*

We denote as $h_\sigma(\mathsf{ALG}) \geq 0$ the number of packets that ALG transmits in the period $[1, T]$, where $\sigma$ denotes the incoming packet sequence.

We define as $T'$-*balanced* any algorithm ALG′ for this problem, for which it holds that if $p^t_{i,\mathsf{LQD}} \geq 1$ then $p^t_{i,\mathsf{ALG}'} \geq 1$, for each queue $i \in [N]$ and at each time

step $t$ of the period $[1, T']$ (where $T' \leq T$).

Since $M \geq N$, no (offline or online) algorithm can keep active at time step $t = 1$, strictly more queues than the number of active queues of $\mathsf{LQD}$ at $t = 1$, for any incoming packet sequence $\sigma$. Hence, no algorithm can transmit strictly more packets at $t = 1$ than the number of packets that $\mathsf{LQD}$ transmits at $t = 1$, for the same incoming packet sequence $\sigma$. Taking into consideration the fact that $\mathsf{LQD}$ is, trivially, an 1-balanced algorithm, we have Observation 4.2:

**Observation 4.2.** *Let $\mathsf{ALG}_{0b}$ denote any algorithm that keeps inactive at $t = 1$ an active queue in the $\mathsf{LQD}$ buffer at $t = 1$, for an incoming sequence $\sigma$. Then, there exists an 1-balanced algorithm $\mathsf{ALG}_{1b}$ such that $h_\sigma(\mathsf{ALG}_{1b}) \geq h_\sigma(\mathsf{ALG}_{0b})$.*

**Lemma 4.3.** *There exists an optimal offline algorithm which is $T$-balanced and which never preempts any packet.*

*Proof.* Let $\mathsf{ALG}_1$ be an offline algorithm which is $t'$-balanced but not $(t' + 1)$-balanced, for some time step $t' \in [T-1]$. We assume that $\mathsf{ALG}_1$ never preempts any packet, since otherwise we can modify $\mathsf{ALG}_1$ so that this algorithm never preempts any packet without decreasing $h_\sigma(\mathsf{ALG}_1)$, where $\sigma$ denotes the incoming packet sequence.

It follows that $p^{t'+1}_{j,\mathsf{ALG}_1} = 0$ and $p^{t'+1}_{j,\mathsf{LQD}} > 0$ for a queue $j \in [N]$. But then, there has to exist a queue $k \neq j$ and a time step $t'' \leq t' + 1$ so that:

- It holds $p^{t''}_{k,\mathsf{LQD}} < p^{t''}_{k,\mathsf{ALG}_1}$ *or*
- The sum of the empty $\mathsf{ALG}_1$ buffer space at $t''$ and $p^{t''}_{k,\mathsf{ALG}_1}$ is greater than $p^{t''}_{k,\mathsf{LQD}}$.

In case two or more different pairs of queues and time steps $\{k, t''\}$ exist, as defined above, then we choose the latest time step $t''$ in the period $[1, t' + 1]$ and we break ties arbitrarily for choosing the queue $k$, at this chosen time step $t''$.

We modify $\mathsf{ALG}_1$ so that one more packet for $j$ is accepted in the period $[1, t'+1]$. For this, one packet less for $k$ (if $p^{t''}_{k,\mathsf{LQD}} < p^{t''}_{k,\mathsf{ALG}_1}$) may need to be accepted by the modified algorithm, at the latest overflow time step in the $\mathsf{LQD}$ buffer of $k$, in the period $[1, t'']$. This gives us a $\hat{t}$-balanced algorithm $\mathsf{ALG}_2$ (for some time step $\hat{t} \in [t', T]$) for which it holds $h_\sigma(\mathsf{ALG}_2) \geq h_\sigma(\mathsf{ALG}_1)$, because $j$ becomes active in the $\mathsf{ALG}_2$ buffer at time step $t'+1$.

Working as above, we obtain a sequence of algorithms $\mathsf{ALG}_1$,...,$\mathsf{ALG}_v$ (for a $v \in \mathbb{Z}^+$), where $\mathsf{ALG}_1, ..., \mathsf{ALG}_{v-1}$ are not $T$-balanced, $\mathsf{ALG}_v$ is $T$-balanced and it holds $h_\sigma(\mathsf{ALG}_l) \geq h_\sigma(\mathsf{ALG}_{l-1})$ for each $l \in [v]$. Note that none of these algorithms preempts any packet, by design. It follows that $h_\sigma(\mathsf{ALG}_v) \geq h_\sigma(\mathsf{ALG}_1)$ which by Observation 4.2, completes the proof. □

**Classifying the queues and packets**

A queue $i \in [N]$ is called as *established* queue at time step $t \in [T]$, if $p^t_{i,\mathsf{LQD}} = 0$ and $p^t_{i,\mathsf{OPT}} > 0$. If $p^t_{i,\mathsf{LQD}} > p^t_{i,\mathsf{OPT}} \geq 1$ then $i$ is a *free* queue at $t$, else if $1 \leq p^t_{i,\mathsf{LQD}} \leq p^t_{i,\mathsf{OPT}}$ then $i$ is a *dominating* queue at $t$. We denote as $D^t$, $E^t$, $F^t$, the sets of dominating, established and free queues, respectively, at $t \in [T]$.

An $\mathsf{OPT}$ packet $q$ of a queue $i \in D^t$ at a time step $t \in [T]$, for which it holds $\ell^t_{\mathsf{OPT}}(q) > p^t_{i,\mathsf{LQD}}$, is called *extra* packet at $t$. An $\mathsf{OPT}$ packet of an established queue at a time step $t \in [T]$ is, also, called *extra* packet at $t$. An $\mathsf{LQD}$ packet of a queue $i \in F^t$ at a time step $t \in [T]$, is called *free* packet at $t$. An $\mathsf{LQD}$ packet of a dominating queue at a time step $t \in [T]$ is called *common* packet, at $t$.

We denote as $e(t)$ the number of extra packets in the $\mathsf{OPT}$ buffer at time step $t \in [T]$. Also, we denote as $f(t)$ the number of free packets in the $\mathsf{LQD}$ buffer, at $t$.

By Lemma 4.3, a queue cannot be inactive in the $\mathsf{OPT}$ buffer and active in the $\mathsf{LQD}$ buffer at the same time step. This gives us the next corollary:

**Corollary 4.4.** *Any queue at any time step $t \in [T]$ can either belong to set $D^t$ or to set $F^t$ or to set $E^t$ or be inactive in both buffers at $t$.*

By Corollary 4.4, the ratio of transmitted packets between OPT and LQD, for an incoming packet sequence $\sigma$, is defined as:

$$r_\sigma = \frac{h_\sigma(\mathsf{OPT})}{h_\sigma(\mathsf{LQD})} = 1 + \frac{\sum_{t=1}^{t=T} |E^t|}{h_\sigma(\mathsf{LQD})} \geq 1 \tag{4.5}$$

The quantity $\sum_{t=1}^{T} |E^t|$ in (4.5) equals the total number of transmitted extra packets by all queues in the period $[1, T]$, i.e., it equals the total number of OPT packets that are extra packets when transmitted by the OPT buffer.

**Observation 4.6.** *Since $M \geq N$, if a queue $i \in [N]$ is inactive in the* LQD *buffer at a time step $t \in [T-1]$ and at least one packet arrives destined to $i$ at $t+1$, then $i$ will be active in the* LQD *buffer at $t+1$.*

**Lemma 4.7.** *If $i \in E^{t+1}$ for any queue $i \in [N]$ where $t \in [T-1]$, then it holds $i \in D^t \cup E^t$.*

*Proof.* First, we show that if $i \in E^{t+1}$ then $p^t_{i,\mathsf{OPT}} > 0$: Assume that $p^t_{i,\mathsf{OPT}} = 0$ and, therefore, that it holds $p^t_{i,\mathsf{LQD}} = 0$ due to Lemma 4.3. If no packet arrives to $i$ at $t+1$, then $i \notin E^{t+1}$ by the definition of an established queue. Otherwise, if at least one packet arrives destined to $i$ at $t+1$ then $i$ has to be active in the LQD buffer at $t+1$, by Observation 4.6; therefore it is $i \notin E^{t+1}$.

We, now, show that if $i \in E^{t+1}$ then $i \notin F^t$: Assume that $i \in F^t$. The length of a free queue in the LQD buffer at any time step is at least 2, by the definition of a free queue and Lemma 4.3; hence since $i \in F^t$ then $i$ has to be active in the LQD buffer at $t+1$ and, therefore, it has to be $i \notin E^{t+1}$.

By Corollary 4.4 and the arguments of the first two paragraphs of this proof, the lemma follows. □

**Lemma 4.8.** *If $i \in F^t$ for any queue $i \in [N]$ at any time step $t \in [T-1]$, then it holds $i \in F^{t+1} \cup D^{t+1}$.*

*Proof.* The length of a free queue in the LQD buffer is at least 2 at any time step, by the definition of a free queue and Lemma 4.3. It follows that $p^t_{i,\mathsf{LQD}} \geq 2$. But then, $i$ has to be active in the LQD buffer at $t+1$, which by Corollary 4.4, completes the proof. □

The next lemma will be used extensively.

**Lemma 4.9.** *It holds $f(t) \geq e(t) + \sum_{i \in F^t} p^t_{i,\mathsf{OPT}}$ at any time step $t \in [T]$.*

*Proof.* First, let us say that at least one queue overflows in the LQD buffer at $t$. Assume for contradiction that $f(t) < e(t) + \sum_{i \in F^t} p^t_{i,\mathsf{OPT}}$, that is $\sum_{i \in F^t} p^t_{i,\mathsf{LQD}} < \sum_{i \in D^t \cup E^t} (p^t_{i,\mathsf{OPT}} - p^t_{i,\mathsf{LQD}}) + \sum_{i \in F^t} p^t_{i,\mathsf{OPT}}$, which gives us the next inequality:

$$\sum_{i \in F^t} p^t_{i,\mathsf{LQD}} + \sum_{i \in D^t} p^t_{i,\mathsf{LQD}} < \sum_{i \in D^t} p^t_{i,\mathsf{OPT}} + \sum_{i \in E^t} p^t_{i,\mathsf{OPT}} + \sum_{i \in F^t} p^t_{i,\mathsf{OPT}} \tag{4.10}$$

The left-hand side of (4.10) equals $M$, since the LQD buffer becomes full at $t$. But the right-hand side of (4.10) lower-bounds the OPT buffer size which has to be equal to $M$. Therefore, we obtain a contradiction and it holds $f(t) \geq e(t) + \sum_{i \in F^t} p^t_{i,\mathsf{OPT}}$.

Now, assume that no queue overflows at $t$ in the LQD buffer and let $t' < t$ be the latest time step before $t$ when a queue overflowed in the LQD buffer (if time step $t' < t$ does not exist, then the lemma holds trivially at $t$, since no extra packets may exist in the OPT buffer at $t$). By the argument of the first two paragraphs of this proof, it holds $f(t') - \sum_{i \in F^{t'}} p^{t'}_{i,\mathsf{OPT}} \geq e(t')$. Also, it holds $e(t') \geq e(t)$, since no new extra packets are obtained in the period $[t'+1, t]$, according to our assumption that no queue overflows in the LQD buffer, in this period. By the last two inequalities, we have:

$$f(t') - \sum_{i \in F^{t'}} p^{t'}_{i,\mathsf{OPT}} \geq e(t) \tag{4.11}$$

Since no queue overflows in the LQD buffer in the period $[t'+1, t]$, it follows by Lemma 4.8 that any queue that is a free queue at $t'$ will stay a free queue at every time step of the period $[t'+1, t]$ and that any arriving packet destined to a free queue in $[t'+1, t]$ will be accepted by the LQD buffer. This gives us the next inequality:

$$\sum_{i \in F^t} p^t_{i,\mathsf{LQD}} - \sum_{i \in F^t} p^t_{i,\mathsf{OPT}} \geq \sum_{i \in F^{t'}} p^{t'}_{i,\mathsf{LQD}} - \sum_{i \in F^{t'}} p^{t'}_{i,\mathsf{OPT}} \tag{4.12}$$

By (4.12) and since $f(t) = \sum_{i \in F^t} p^t_{i,\mathsf{LQD}}$ and $f(t') = \sum_{i \in F^{t'}} p^{t'}_{i,\mathsf{LQD}}$, we have:

$$f(t) - \sum_{i \in F^t} p^t_{i,\mathsf{OPT}} \geq f(t') - \sum_{i \in F^{t'}} p^{t'}_{i,\mathsf{OPT}} \tag{4.13}$$

By (4.11) and (4.13), we have $f(t) - \sum_{i \in F^t} p^t_{i,\mathsf{OPT}} \geq e(t)$, completing the proof.

□

Assume, now, that $i \in E^t$ at a time step $t \in [T]$ for a queue $i \in [N]$. We denote as $t^{\lambda_i}_{2,i} \in [T]$ the first time step before $t$ such that $i$ is an established queue at each time step of the period $[t^{\lambda_i}_{2,i}, t]$, where the superscript $\lambda_i \in \mathbb{Z}^+$ will be defined shortly. Also, as $t^{\lambda_i}_{1,i}$ we denote the latest time step before $t^{\lambda_i}_{2,i}$ when $i$ overflowed as a dominating queue in the LQD buffer and we know that such a time step exists by Observation 4.6 and Lemma 4.7.

This gives us a strict total order: $t^1_{1,i} < t^1_{2,i} < t^2_{1,i} < t^2_{2,i} < ...$, where the superscript denotes each pair of two consecutive time steps in the total order, starting from pair $\{t^1_{1,i}, t^1_{2,i}\}$, then pair $\{t^2_{1,i}, t^2_{2,i}\}$ etc, with the only possible exception of the last $t^{\lambda_i}_{1,i}$ time step in the total order that may not be paired with a time step $t^{\lambda_i}_{2,i}$. Finally, let us say that $\lambda_i \in [\Lambda_i]$ where $\Lambda_i \in \mathbb{Z}^+$, i.e., it holds $t^1_{1,i} < t^1_{2,i} < ... < t^{\Lambda_i}_{1,i} < t^{\Lambda_i}_{2,i}$

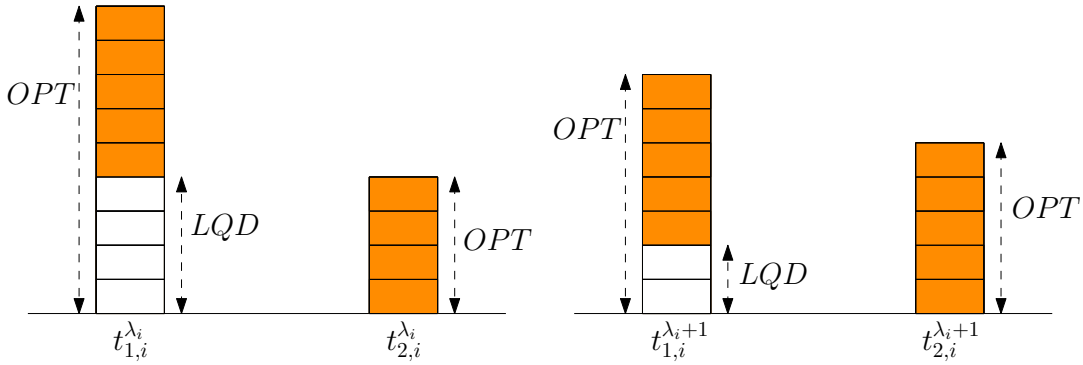


Figure 4.1: *Left*: A queue $i$ that becomes an established queue at time step $t_{2,i}^{\lambda_i}$ has to overflow as a dominating queue in the LQD buffer, for at least one time step in the past and we denote the latest such time step as $t_{1,i}^{\lambda_i}$ ($\lambda_i \in \mathbb{Z}^+$). *Right:* The next time step after $t_{2,i}^{\lambda_i}$ when $i$ overflows as a dominating queue in the LQD buffer (if such time step exists) is a lower bound of $t_{1,i}^{\lambda_i+1}$ and, consequently, of $t_{2,i}^{\lambda_i+1}$. We have identified the extra packets of $i$ in color.

or $t_{1,i}^1 < t_{2,i}^1 < ... < t_{1,i}^{\Lambda_i}$, depending on whether time step $t_{2,i}^{\Lambda_i}$ exists or not, as already mentioned (see Figure 4.1).

For example, assume that a queue $i \in [N]$ becomes established for the first time in $[1, T]$ at time step 10. Then, it is $t_{2,i}^1 = 10$ (note that time step 10 may be equal to $t_{1,j}^{\lambda_j}$ or $t_{2,j}^{\lambda_j}$ for another queue $j \neq i$ and for a $\lambda_j \in [\Lambda_j]$). By Observation 4.6 and Lemma 4.7, it follows that $i$ has overflowed in the LQD buffer as a dominating queue, for at least one time step before time step 10 and assume that the latest time step in the period [1,10] of such an overflow is time step 7, i.e., $t_{1,i}^1 = 7$. Assume, also, that the first time step after $t_{2,i}^1 = 10$ when $i$ overflows as a dominating queue in the LQD buffer (if such time step exists) is time step 30 and the first time step after time step 30 when $i$ becomes an established queue (if such time step exists) is time step 40. It follows that $t_{2,i}^2 = 40$ and that $t_{1,i}^2$ is the latest time step in the period [30, 39] when $i$ overflows as a dominating queue in the LQD buffer, due to Lemma 4.7. Continuing in the same way, we construct the aforementioned strict total order of time steps. This concludes our example.

We continue the proof, with the rather intuitive Lemmas 4.14 and 4.15.

**Lemma 4.14.** *Any queue $i \in [N]$ is a dominating queue at each time step of the period $[t_{1,i}^{\lambda_i}, t_{2,i}^{\lambda_i} - 1]$, for any $\lambda_i \in [\Lambda_i]$.*

*Proof.* Assume that there exists a time step $t \in [t_{1,i}^{\lambda_i}, t_{2,i}^{\lambda_i} - 1]$, when $i$ is an established queue. But this contradicts the selection of time step $t_{2,i}^{\lambda_i}$.

Assume that there exists a time step $t \in [t_{1,i}^{\lambda_i}, t_{2,i}^{\lambda_i} - 1]$ when $i$ is a free queue. But then, $i$ has to overflow in the LQD buffer as a dominating queue at a time step of the period $[t+1, t_{2,i}^{\lambda_i} - 1]$, due to Lemmas 4.7 and 4.8. But this contradicts the selection of time step $t_{1,i}^{\lambda_i}$.

Similarly, $i$ cannot become an inactive queue in both buffers at a time step $t \in [t_{1,i}^{\lambda_i}, t_{2,i}^{\lambda_i} - 1]$, since $i$ has to overflow in the LQD buffer as a dominating queue in at least one time step of the period $[t+1, t_{2,i}^{\lambda_i}]$, due to Observation 4.6 and Lemma 4.7.

By the arguments of the first three paragraphs of this proof and Corollary 4.4, the lemma follows. □

**Lemma 4.15.** *The number of extra packets of any queue $i \in [N]$ cannot strictly increase between any two consecutive time steps of the period $[t_{1,i}^{\lambda_i}, t_{2,i}^{\lambda_i}]$, for any $\lambda_i \in [\Lambda_i]$, i.e., it holds $p_{i,\mathsf{OPT}}^t - p_{i,\mathsf{LQD}}^t \geq p_{i,\mathsf{OPT}}^{t+1} - p_{i,\mathsf{LQD}}^{t+1}$ where $t \in [t_{1,i}^{\lambda_i}, t_{2,i}^{\lambda_i} - 1]$.*

*Proof.* For the number of extra packets of $i$ to strictly increase between two consecutive time steps $t$ and $t+1$ both of which belonging to a period $[t_{1,i}^{\lambda_i}, t_{2,i}^{\lambda_i}]$, the queue $i$ has to overflow in the LQD buffer at time step $t+1$ as a dominating queue.

But this contradicts the selection of time step $t_{1,i}^{\lambda_i}$ (if $(t+1) \in [t_{1,i}^{\lambda_i}+1, t_{2,i}^{\lambda_i}-1]$) or the selection of time step $t_{2,i}^{\lambda_i}$ (if $(t+1) = t_{2,i}^{\lambda_i}$). □

We conclude this part of the proof, with the following lemma.

**Lemma 4.16.** *The number of extra packets that any queue $i \in [N]$ has at time step $t_{2,i}^{\lambda_i}$ (for any $\lambda_i \in [\Lambda_i - 1]$) upper-bounds the number of transmitted extra packets of $i$ in the period $[t_{2,i}^{\lambda_i}, t_{1,i}^{\lambda_i+1} - 1]$. Also, the number of extra packets that any queue*

*$i \in [N]$ has at time step $t_{2,i}^{\Lambda_i}$ (if such time step exists) upper-bounds the number of transmitted extra packets of $i$ in the period $[t_{2,i}^{\Lambda_i}, T]$.*

*Proof.* Assume that the number of transmitted extra packets of $i$ in the period $[t_{2,i}^{\lambda_i}, t_{1,i}^{\lambda_i+1} - 1]$ (for a $\lambda_i \in [\Lambda_i - 1]$) is strictly greater than $p_{i,\mathsf{OPT}}^{t_{2,i}^{\lambda_i}}$ (which equals the number of extra packets that $i$ has at $t_{2,i}^{\lambda_i}$, since $i$ is an established queue at this time step). But this means that:

- The queue $i$ has overflowed in the LQD buffer as a dominating queue in at least one time step of the period $[t_{2,i}^{\lambda_i}, t_{1,i}^{\lambda_i+1} - 1]$ *and*
- The queue $i$ is an established queue in at least one time step of the period $[t' + 1, t_{1,i}^{\lambda_i+1} - 1]$, where $t'$ denotes the first time step in $[t_{2,i}^{\lambda_i}, t_{1,i}^{\lambda_i+1} - 1]$ when $i$ overflows as a dominating queue in the LQD buffer.

But the second argument, above, implies that $t_{2,i}^{\lambda_i+1} < t_{1,i}^{\lambda_i+1}$ which cannot happen for the strict total order defined before. This completes the proof for the first statement of the lemma.

Working in the same way, we prove the second statement, i.e., that the number of extra packets that $i \in [N]$ has at time step $t_{2,i}^{\Lambda_i}$, upper-bounds the number of transmitted extra packets of $i$ in the period $[t_{2,i}^{\Lambda_i}, T]$. □

**Introducing the packet connections**

As already mentioned in the introduction of the current chapter, the way that Hahne et al. [2001] upper-bound by 2 the competitive ratio of LQD, for the general case of shared-memory switches equipped with any number of output ports, is by assigning connections between OPT packets and LQD packets. We shall work in a similar way.

**Definition 4.17.** *A connection between an extra packet $e$ and an LQD packet $e'$ which is assigned at a time step $t \in [T]$ when both packets are in their respective buffers, will be called <u>valid</u> if $\ell_{\mathsf{LQD}}^t(e') \leq \ell_{\mathsf{OPT}}^t(e)$.*

All connections will be valid, hence a valid connection will be usually called from now as, simply, connection. We say that an extra packet $u$ is *connected* with an LQD packet $u'$ at a time step $t \in [T]$, if:

- These two packets were assigned the connection between them at a time step $t'$ of the period $[1, t]$ *and*
- These two packets remain connected together at each time step of the period $[t', t]$ *and*
- These two packets are in their respective buffers at $t$, i.e., packet $u$ is not transmitted in the period $[t', t]$ and $u'$ is not preempted or transmitted in the same period.

**Fact 4.18.** *From the time step $t_c \in [T]$ when an extra packet $p$ (destined to a queue $i \in [N]$) is assigned a connection with an* LQD *packet $y$, these two packets are assumed to stay connected at each later time step, until any of the following takes place, in any order:*

1. *The* LQD *packet $y$ is transmitted, or*
2. *The* LQD *packet $y$ is preempted, or*
3. *The extra packet $p$ becomes a non-extra packet at a time step $t'_c > t_c$, i.e., $p$ is still in the* OPT *buffer at $t'_c$ but it is not an extra packet at $t'_c$.*

In the first case of Fact 4.18, that is if the LQD packet $y$ is transmitted[1] at a time step $t_s > t_c$, we delete the connection between $p$ and $y$ at $t_s$, i.e., $p$ and $y$ are no more connected in the period $[t_s, T]$. Finally, we shall say that $p$ is *associated* with the transmitted LQD packet $y$ in the period $[t_s, T]$ (or, equivalently, that there exists an *association* between $p$ and $y$ in the period $[t_s, T]$) .

[1]Note that by Definition 4.17, a validly connected LQD packet cannot be transmitted at a later time step after the extra packet that it is connected with, is transmitted.

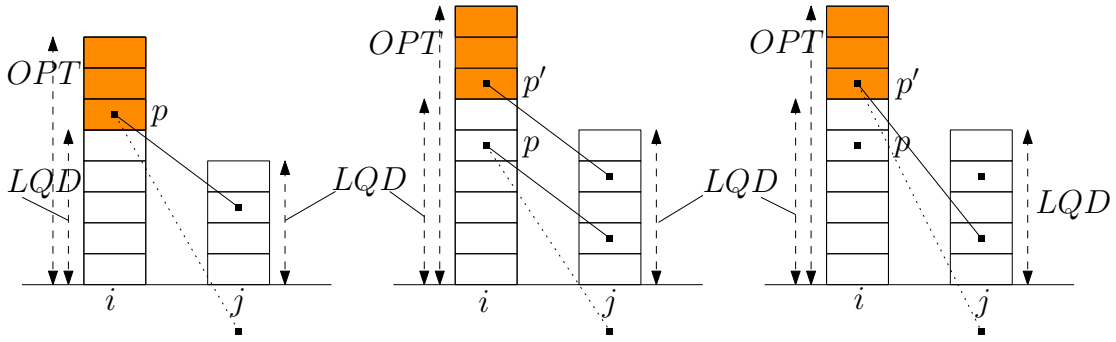


Figure 4.2: In the figure we have two queues ($i$ and $j \neq i$) and in color we identify the extra packets of $i$. *Left*: At time step $t'_c - 1$ the extra packet $p$ is connected with an LQD packet (bold line segment) and associated with a transmitted LQD packet (dotted line segment). *Center*: At $t'_c$ the OPT packet $p$ becomes a non-extra packet because $i$ accepts packets in the LQD buffer; therefore we assign (on the *right*, again at time step $t'_c$) to an extra packet $p'$ of the same queue $i$ the connection and association that $p$ had at $t - 1$, after deleting the single connection that $p'$ had.

In the second case of Fact 4.18, that is in the case the LQD packet $y$ is preempted at a time step $t_l > t_c$, we delete the connection between $p$ and $y$ at $t_l$ and we assign at $t_l$, a new connection between $p$ and the newly accepted LQD packet that takes the buffer space of $y$ at $t_l$. Since the newly accepted LQD packet cannot be located at a greater queue position than that of the preempted LQD packet $y$ at the preemption time step $t_l$, this new connection is valid.

Finally, in the third case of Fact 4.18, that is if $p$ is an extra packet at each time step of the period $[t_c, t'_c - 1]$ but not an extra packet at $t'_c$, then (see Figure 4.2):

- If there exists a non-empty set $X$ of extra packets belonging to the same queue with the OPT packet $p$ at $t'_c$ (i.e., belonging to queue $i$), each member of which has a strictly smaller sum of connections and associations compared to the sum of connections and associations that $p$ has at $t'_c$ then:

  1. We choose the extra packet $p' \in X$ that has the smallest queue position among all members of $X$ at $t'_c$ and for which it holds $\ell^{t'_c}_{\mathsf{OPT}}(p') > \ell^{t'_c}_{\mathsf{OPT}}(p)$

*and*

2. We delete all connections and associations of both $p$ and $p'$ at $t'_c$ *and*
3. We assign to $p'$ at $t'_c$ all connections[2] and associations that we just deleted from $p$ (in the second step) at $t'_c$.

- Else if $X = \emptyset$ then we delete all connections and associations of $p$ at $t'_c$.

This concludes analysis for Fact 4.18.

### 4.2.2 Analysis

By Observation 4.1 and Definition 4.17 we have Observation 4.19.

**Observation 4.19.** *A connection that is assigned to any extra packet of a queue $i \in [N]$ at any of its $t_{1,i}^{\lambda_i}$ time steps ($\lambda_i \in [\Lambda_i]$) is valid.*

**Lemma 4.20.** *Let $c$ be a common packet of a queue $i \in [N]$ at a $t_{1,i}^{\lambda_i}$ time step ($\lambda_i \in [\Lambda_i]$). Then, $c$ will never be preempted by the* LQD *buffer and will never become a free packet until its transmission.*

*Proof.* By the definition of time step $t_{1,i}^{\lambda_i}$ and Lemma 4.14, we have that the packet $c$ will never be preempted by the LQD buffer, i.e., it will be transmitted by the LQD buffer at a time step of the period $[t_{1,i}^{\lambda_i}, t_{2,i}^{\lambda_i}]$ since $i$ becomes inactive in the LQD buffer at $t_{2,i}^{\lambda_i}$.

Also, the packet $c$ will stay a common packet at each time step until its transmission from the LQD buffer. This is because $i$ cannot become a free queue at any time step of the period $[t_{1,i}^{\lambda_i}, t_{2,i}^{\lambda_i} - 1]$, by Lemma 4.14. Since the free packets are kept only by free queues in the LQD buffer and because any LQD packet can be either a free packet or a common packet at the same time step, the lemma follows. □

We shall, now, need some definitions in order to proceed.

[2]Each connection assigned to the extra packet $p'$ at $t'_c$ is valid since $\ell_{\mathsf{OPT}}^{t'_c}(p') > \ell_{\mathsf{OPT}}^{t'_c}(p)$ and due to Definition 4.17.

**Definition 4.21.** *Assume that $i \in D^t$ at a time step $t \in [T]$. Then, the upmost $h$ extra packets of $i$ at $t$ (where $h \in \mathbb{Z}^+$) are the $h$ extra packets of $i$ that occupy the $h$ greatest queue positions of $i$ at $t$.*

**Definition 4.22.** *An extra packet $s$ of a queue $i \in [N]$ for which it holds $\ell^t_{\mathsf{OPT}}(s) > 2 \cdot p^t_{i,\mathsf{LQD}}$ at a time step $t \in [T]$, is called upper extra packet of $i$ at $t$. Any other extra packet of $i$ at $t$ is called lower extra packet of $i$ at $t$.*

Let $l^t_i \geq 0$ and $u^t_i \geq 0$ denote the numbers of lower and upper extra packets, respectively, that a queue $i \in [N]$ has at a time step $t \in [T]$.

**Definition 4.23.** *If $2 \cdot p^t_{i,\mathsf{LQD}} < p^t_{i,\mathsf{OPT}}$ at a time step $t \in [T]$ for a queue $i \in [N]$, then $i$ is called primary dominating queue at $t$ else if $2 \cdot p^t_{i,\mathsf{LQD}} \geq p^t_{i,\mathsf{OPT}}$ then $i$ is called secondary dominating queue at $t$.*

By Observation 4.1 and since $M \geq N$, we have Observation 4.24:

**Observation 4.24.** *At any time step $t \in [T]$ when a packet of a queue $i \in [N]$ is preempted or rejected by the LQD buffer, it holds $p^t_{i,\mathsf{LQD}} \geq \lfloor M/N \rfloor$.*

**Lemma 4.25.** *If a queue $i \in [N]$ overflows in the LQD buffer at a time step $t \in [T]$ and $i$ is a primary dominating queue at $t$, then it has to be $|F^t| \geq 2$.*

*Proof.* First of all, it cannot be $|F^t| = 0$, because at least one extra packet of $i$ exists in the OPT buffer at $t$; therefore at least one free packet has to exist in the LQD buffer at $t$ due to Lemma 4.9.

Assume for contradiction that $|F^t| = 1$. The number of extra packets of $i$ at $t$ is at least $p^t_{i,\mathsf{LQD}} + 1$, by Definition 4.23. Therefore, by Lemma 4.9, we have $f(t) \geq (p^t_{i,\mathsf{LQD}} + 1) + |F^t| = p^t_{i,\mathsf{LQD}} + 2$, since the free queue will be active in the OPT buffer at $t$ (due to Lemma 4.3). But a single free queue at $t$ cannot have $p^t_{i,\mathsf{LQD}}+2$ free packets, since $i$ overflows at $t$ in the LQD buffer and due to Observation 4.1. Hence, it cannot be $|F^t| = 1$ and, therefore, it has to be $|F^t| \geq 2$. □

The next lemma follows easily.

**Lemma 4.26.** *If $i \in F^t$ and a dominating queue overflows in the* LQD *buffer at $t$, then it holds $p^t_{i,\mathsf{LQD}} \leq \lceil M/2 \rceil$.*

*Proof.* Assume for contradiction that $p^t_{i,\mathsf{LQD}} \geq M/2 + 1$. By Observation 4.1, the length of the dominating queue that overflows in the LQD buffer at $t$, is at least equal to the length of $i$ in the LQD buffer at $t$, minus 1. Taking the sum of the packet numbers for these two queues at $t$ in the LQD buffer, it follows that at least $M + 1$ packets exist in the LQD buffer at $t$, which cannot happen. □

**Lemma 4.27.** *If $|F^t| = 1$ then it is $e(t) \leq \lceil M/2 \rceil$, at any time step $t \in [T]$.*

*Proof.* If a dominating queue overflows in the LQD buffer at $t$, the lemma follows from Lemmas 4.9 and 4.26. If no dominating queue overflows at $t$, then let $t'$ be the latest time step before $t > t'$ when a dominating queue overflowed in the LQD buffer; hence we have $e(t') \geq e(t)$ since no new extra packets are obtained in the period $[t' + 1, t]$, according to our assumption that no dominating queue overflows in the LQD buffer in $[t' + 1, t]$.

By Lemma 4.9, at any time step when at least one extra packet exists in the OPT buffer, at least one free packet has to exist in the LQD buffer; therefore it is $|F^{t'}| \geq 1$. But it cannot be $|F^{t'}| \geq 2$, since a free queue has to become dominating at a time step of the period $[t' + 1, t]$ so that it holds $|F^t| = 1$. To see that, note that in order for a free queue at any time step $t_f \in [T - 1]$ to become dominating at $t_f + 1$, this queue has to overflow in the LQD buffer at $t_f + 1$, due to the definitions of a free queue and of a dominating queue. This contradicts our assumption that no dominating queue overflows in $[t' + 1, t]$. Therefore, it has to be $|F^{t'}| = 1$. By Lemmas 4.9 and 4.26 and since $e(t') \geq e(t)$, the proof is complete. □

According to the connection assignment procedure that we describe in Section 4.2.3 for the case of three-port switches and in Section 4.2.4 for two-port

switches, any extra packet of any queue $i \in [N]$ may be connected with at most two LQD packets at the same time step and if an extra packet is connected with two LQD packets at the same time step then one of its connections has to be with a common packet of the same queue $i$ and assigned at a $t_{1,i}^{\lambda_i}$ time step (for a $\lambda_i \in [\Lambda_i]$). This gives us the next lemma.

**Lemma 4.28.** *Each extra packet belonging to any queue $i \in [N]$ at any time step $t \in [T]$, may be connected with at most one* LQD *packet of a queue other than $i$.*

*Proof.* The lemma follows by Fact 4.18 and due to Lemma 4.20. To see that, note that if an extra packet has two connections, at least one of its connections has to be with a common packet of the same queue that the extra packet belongs to, which will never be preempted by the LQD buffer. □

**Lemma 4.29.** *Any extra packet may be connected with at most one free packet, at any time step of the period $[1, T]$.*

*Proof.* The lemma follows by Lemma 4.28 and because according to the definition of a free packet, a queue that has at least one extra packet in the OPT buffer at any time step, cannot have a free packet at the same time step in the LQD buffer. □

### 4.2.3 The LQD algorithm for three-port switches

Throughout Section 4.2.3, we assume that $N = 3$. We shall, now, assign valid connections between extra packets and LQD packets, proving that the next invariant holds at every time step $t \in [1, T]$:

**Invariant 4.30.** *No* LQD *packet is connected with two or more extra packets at $t$.*

Hence, assume that a queue $i \in [N]$ overflows in the LQD buffer at a $t_{1,i}^{\lambda_i}$ time step (for a $\lambda_i \in [\Lambda_i]$). For simplicity, let $t_{1,i} = t_{1,i}^{\lambda_i}$ and $t_{2,i} = t_{2,i}^{\lambda_i}$ for this given $\lambda_i$. We distinguish between the cases that $i$ is primary dominating or secondary

dominating, at $t_{1,i}$:

**Case 1: The queue $i$ is primary dominating at $t_{1,i}$**

By Lemma 4.25, since $N = 3$ and because $i$ is a dominating queue at $t_{1,i}$, it holds $|F^{t_{1,i}}| = 2$. Hence, we assign one connection to each lower extra packet of $i$ at $t_{1,i}$ with a different common packet of $i$. The number of common packets of $i$ at $t_{1,i}$ suffices so that each lower extra packet of $i$ is assigned one connection, by Definition 4.22. We, also, assign one connection to every extra packet of $i$ with a different free packet at $t_{1,i}$. The number of free packets at $t_{1,i}$ suffices to assign one connection to each extra packet of $i$ with a different free packet, due to Lemma 4.9. Any connection assigned at $t_{1,i}$ is valid, by Observation 4.19.

Wrapping up, we assigned two connections to each lower extra packet of $i$ at $t_{1,i}$ and one connection to each upper extra packet of $i$ again at $t_{1,i}$ (see Figure 4.3, on the left). Invariant 4.30 holds at $t_{1,i}$ since each LQD packet that is assigned a connection at this time step, is different. Before continuing, we prove the following lemma:

**Lemma 4.31.** *No packet in the* LQD *buffer at time step $t_{2,i} > t_{1,i}$ may be connected with a connection which was assigned at time step $t_{1,i}$.*

*Proof.* It suffices to show that the maximum queue position at $t_{1,i}$ of any LQD packet that is assigned a connection at $t_{1,i}$ is $p_{i,\mathsf{LQD}}^{t_{1,i}}$. This is because it will take at least $p_{i,\mathsf{LQD}}^{t_{1,i}}$ time steps for $i$ to become an established queue after $t_{1,i}$, i.e., $t_{2,i} \geq t_{1,i} + p_{i,\mathsf{LQD}}^{t_{1,i}}$, since $i$ is a dominating queue at each time step of the period $[t_{1,i}, t_{2,i} - 1]$ (by Lemma 4.14) and $i$ does not overflow in the LQD buffer in this period (by the definition of time step $t_{1,i} = t_{1,i}^{\lambda_i}$).

Therefore, since $p_{i,\mathsf{LQD}}^{t_{1,i}}$ is the length of $i$ at $t_{1,i}$ in the LQD buffer, the maximum queue position of any common packet that is assigned a connection at $t_{1,i}$ is, trivially, $p_{i,\mathsf{LQD}}^{t_{1,i}}$.

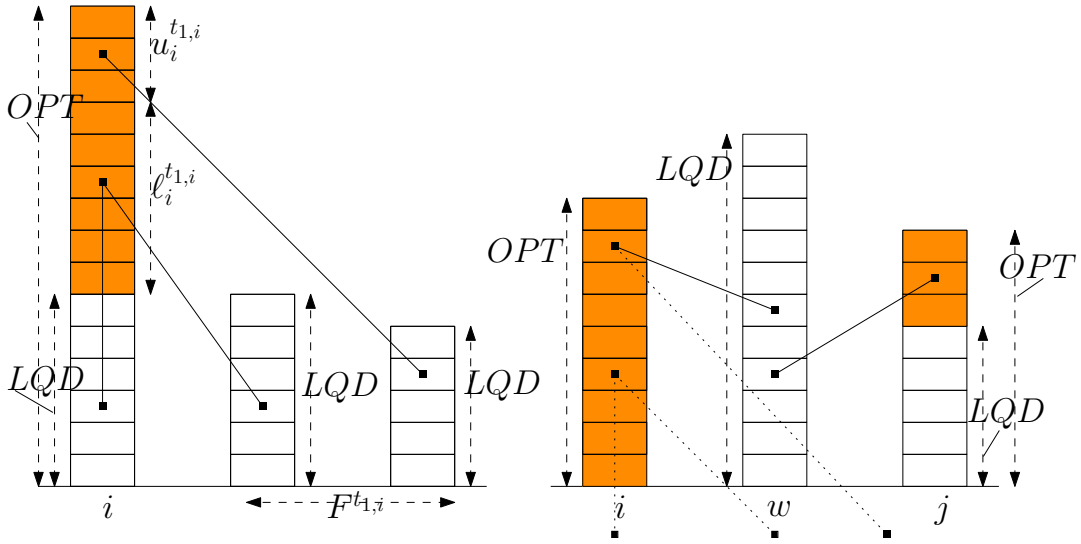


Figure 4.3: An example of the connection assignment of Case 1. *Left*: The queue $i$ overflows at $t_{1,i}$. Each lower extra packet of $i$ is assigned two connections and each upper extra packet of $i$, one connection. *Right*: At $t_{2,i}$ all connected LQD packets from $t_{1,i}$ have left the buffer by Lemma 4.31 (dotted line segments) and we assign one connection to each of the $(p_{i,\mathsf{OPT}}^{t_{2,i}} - l_i^{t_{1,i}})$ upmost extra packets of $i$. Even if another dominating queue $j$ exists at $t_{2,i}$ and each of $j$'s extra packets is connected with one free packet, Invariant 4.30 holds at $t_{2,i}$, as we show in sub-case (A2).

Now, for the free packets that are assigned connections at $t_{1,i}$, we have the following: By Lemmas 4.9 and 4.25, the number of free packets in the LQD buffer at $t_{1,i}$ is at least equal to the number of extra packets of $i$ at $t_{1,i}$, plus 2. Therefore, we do not need to assign connections to the (at most two) free packets that may be located at a queue position equal to $p_{i,\mathsf{LQD}}^{t_{1,i}}+1$ at $t_{1,i}$ (by Observation 4.1, $p_{i,\mathsf{LQD}}^{t_{1,i}}+1$ is the maximum queue position of any LQD packet at $t_{1,i}$). It follows that the maximum queue position at $t_{1,i}$ of any free packet which is assigned a connection at $t_{1,i}$ is $p_{i,\mathsf{LQD}}^{t_{1,i}}$, which completes the proof. □

If $l_i^{t_{1,i}} \geq p_{i,\mathsf{OPT}}^{t_{2,i}}$ then the connection assignment for Case 1 is complete. Otherwise, if $l_i^{t_{1,i}} < p_{i,\mathsf{OPT}}^{t_{2,i}}$ then we distinguish between the following sub-cases (A1), (A2) and (A3):

**(A1)** If $|F^{t_{2,i}}| = 2$, then it is only queue $i$ that has extra packet(s) at $t_{2,i}$, in the OPT buffer. By this and Lemma 4.31, we assign a connection at $t_{2,i}$ to each of the upmost $(p_{i,\mathsf{OPT}}^{t_{2,i}} - l_i^{t_{1,i}})$ extra packets of $i$ at $t_{2,i}$, with a different free packet so that Invariant 4.30 holds at this time step. Note that due to Lemma 4.9, the number of free packets suffices for these connections to be assigned at $t_{2,i}$.

**(A2)** If $|F^{t_{2,i}}| = 1$, then it is $e(t_{2,i}) \leq \lceil M/2 \rceil$, by Lemma 4.27. We denote the single free queue we have at $t_{2,i}$ as $w \neq i$ and the third queue that we may have at $t_{2,i}$ (which can be dominating or established) as $j \neq \{i, w\}$ (see Figure 4.3, on the right). By Observation 4.24 it is $p_{i,\mathsf{LQD}}^{t_{1,i}} \geq \lfloor M/3 \rfloor$, which by Definition 4.22 gives us $l_i^{t_{1,i}} \geq \lfloor M/3 \rfloor$. By the last inequality and since $e(t_{2,i}) \leq \lceil M/2 \rceil$, it is:

$$2 \cdot l_i^{t_{1,i}} \geq e(t_{2,i}) - 1 \tag{4.32}$$

But the number of extra packets in the OPT buffer at $t_{2,i}$ is equal to the number of extra packets of $i$ at $t_{2,i}$ (which is an established queue at this time step) plus the number of extra packets of $j$ at $t_{2,i}$:

$$e(t_{2,i}) = (p_{j,\mathsf{OPT}}^{t_{2,i}} - p_{j,\mathsf{LQD}}^{t_{2,i}}) + p_{i,\mathsf{OPT}}^{t_{2,i}} \tag{4.33}$$

By (4.32), (4.33) and since we have assumed that $l_i^{t_{1,i}} < p_{i,\mathsf{OPT}}^{t_{2,i}}$, it holds:

$$l_i^{t_{1,i}} \geq p_{j,\mathsf{OPT}}^{t_{2,i}} - p_{j,\mathsf{LQD}}^{t_{2,i}} \tag{4.34}$$

Therefore, we assign at $t_{2,i}$ a valid connection to each of the upmost $(p_{i,\mathsf{OPT}}^{t_{2,i}} - l_i^{t_{1,i}})$ extra packets of $i$ at $t_{2,i}$, without making a connection to a free packet which is already connected with an extra packet of $j$. This is because each of the upmost $(p_{i,\mathsf{OPT}}^{t_{2,i}} - l_i^{t_{1,i}})$ extra packets of $i$ at $t_{2,i}$, is located at a queue position at $t_{2,i}$ which is at least equal to $l_i^{t_{1,i}}$, since $i$ is an established queue at $t_{2,i}$. But, by (4.34) and due to Lemma 4.29, the number of free packets that may be already connected with

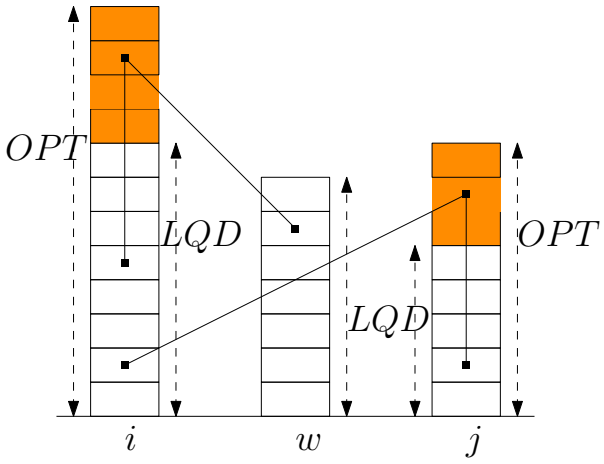


Figure 4.4: An example of the connection assignment of Case 2: One connection to each extra packet of $i$ is assigned with a free packet at $t_{1,i}$, by Lemma 4.9. A second connection to each extra packet of $i$ is assigned with a common packet of $i$, at $t_{1,i}$. Invariant 4.30 is not violated at $t_{1,i}$ as we show in the analysis of Case 2.

extra packets of $j$ at $t_{2,i}$ is at most $l_i^{t_{1,i}}$. By this and due to Lemma 4.31 it follows that Invariant 4.30 is not violated by any connection assigned at $t_{2,i}$.

**(A3)** It cannot be $|F^{t_{2,i}}| = 0$, since at least one extra packet of $i$ exists in the OPT buffer, at $t_{2,i}$. Therefore, by Lemma 4.9, it has to be $|F^{t_{2,i}}| \geq 1$.

The analysis for Case 1 is complete.

**Case 2: The queue $i$ is secondary dominating at $t_{1,i}$**

We distinguish between sub-cases $(B1)$, $(B2)$ and $(B3)$, for the first connection to each extra packet of $i$ at $t_{1,i}$:

**(B1)** If $|F^{t_{1,i}}| = 2$ then the only extra packets in the OPT buffer at $t_{1,i}$ belong to queue $i$. Hence, we assign at $t_{1,i}$ one connection to each extra packet of $i$ with a different common packet of $i$. The number of extra packets of $i$ at $t_{1,i}$ is at most equal to the number of common packets of $i$ at $t_{1,i}$, by Definition 4.23. It

follows that the number of common packets of $i$ suffices for these connections to be assigned at $t_{1,i}$ and, therefore, Invariant 4.30 holds at $t_{1,i}$. Finally, each connection is valid, due to Observation 4.19.

**(B2)** If $|F^{t_{1,i}}| = 1$, then one other dominating (or established) queue $j \neq i$ may exist at $t_{1,i}$. Let us denote as $w \neq \{i, j\}$ the single free queue we have at $t_{1,i}$. By Lemma 4.9 it is $e(t_{1,i}) \leq f(t_{1,i}) - 1$. Also, it is $f(t_{1,i}) = p_{w,\mathsf{LQD}}^{t_{1,i}} \leq p_{i,\mathsf{LQD}}^{t_{1,i}} + 1$ due to Observation 4.1. By the last two inequalities, we get:

$$e(t_{1,i}) \leq p_{i,\mathsf{LQD}}^{t_{1,i}} \tag{4.35}$$

By (4.35) and Lemma 4.28, the number of common packets of $i$ that are not connected at $t_{1,i}$ with extra packets of $j$, is at least equal to the number of extra packets that the queue $i$ has at time step $t_{1,i}$. Hence, we assign a connection to each extra packet of $i$ with a different common packet of $i$ at $t_{1,i}$, without violating Invariant 4.30 at $t_{1,i}$.

**(B3)** It cannot be $|F^{t_{1,i}}| = 0$, for the same reason as in sub-case (A3).

Finally, by Lemmas 4.9 and 4.29, we assign a second connection to each extra packet of $i$ with a different free packet at $t_{1,i}$, without violating Invariant 4.30 at this time step. This second connection is, also, valid by Observation 4.19.

The analysis for Case 2 is complete.

**Upper-bounding the LQD competitive ratio, when $N = 3$**

The connection assignment procedure that we described in Cases 1 and 2, is applied for every queue $k \in [N]$ and at each $t_{1,k}^{\lambda_k}$ time step (that is, for each $\lambda_k \in [\Lambda_k]$). This

gives us the next lemma.

**Lemma 4.36.** *Invariant 4.30 holds at every time step of* $[1, T]$.

*Proof.* Invariant 4.30 may be violated only at a time step when a connection between an extra packet and an already connected LQD packet is assigned. But we showed in both Cases 1 and 2 that at any time step when a connection is assigned to an extra packet of $i \in [N]$ (either time step $t_{1,i}$ or $t_{2,i}$), Invariant 4.30 is not violated. □

We, now, proceed towards proving the key lemma of Section 4.2.3:

**Lemma 4.37.** *Each transmitted extra packet $e$ is connected or associated with two* LQD *packets, at its transmission time step.*

*Proof.* The extra packet $e$ that is transmitted at a time step $t \in [T]$ belongs to a queue $i \in [N]$ which was either primary dominating or secondary dominating at the latest time step before $t$, when $i$ overflowed in the LQD buffer, due to Observation 4.6 and Lemma 4.7. Hence, we distinguish between the cases that $i$ was secondary dominating or primary dominating at this $t_{1,i}^{\lambda_i}$ time step ($\lambda_i \in [\Lambda_i]$):

Let us start with the case that $i$ was secondary dominating at $t_{1,i}^{\lambda_i}$: Each extra packet of $i$ is assigned two connections at $t_{1,i}^{\lambda_i}$, as we describe in the analysis of Case 2. But, due to Lemmas 4.15 and 4.16 we have:

- If $\lambda_i \in [\Lambda_i - 1]$, then the number of transmitted extra packets of $i$ in the period $[t_{2,i}^{\lambda_i}, t_{1,i}^{\lambda_i+1} - 1]$ is at most equal to the number of extra packets that $i$ has at $t_{1,i}^{\lambda_i}$, *else*
- If $\lambda_i = \Lambda_i$ and time step $t_{2,i}^{\Lambda_i}$ exists, then the number of transmitted extra packets of $i$ in the period $[t_{2,i}^{\Lambda_i}, T]$ is at most equal to the number of extra packets that $i$ has at $t_{1,i}^{\Lambda_i}$ (if time step $t_{2,i}^{\Lambda_i}$ does not exist, then the number of transmitted extra packets of $i$ in the period $[t_{1,i}^{\Lambda_i}, T]$ is, trivially, 0).

Therefore, by Fact 4.18, it follows that each transmitted extra packet of $i$ has a sum of two connections and associations at its transmission time step. To see

that, note that each transmitted extra packet of $i$ was either an extra packet at $t_{1,i}^{\lambda_i}$ and was assigned two connections at this time step or it was accepted by queue $i$ after time step $t_{1,i}^{\lambda_i}$ and a sum of two connections and associations were assigned to this packet according to the third case of Fact 4.18.

Assume, now, that $i$ was primary dominating at $t_{1,i}^{\lambda_i}$. At $t_{1,i}^{\lambda_i}$ we assigned two connections to each of the $l_i^{t_{1,i}}$ lower extra packets of $i$ and one connection to each of the upper $u_i^{t_{1,i}}$ extra packets of $i$. At $t_{2,i}^{\lambda_i}$ we assigned one connection to each of the $(p_{i,\mathsf{OPT}}^{t_{2,i}} - l_i^{t_{1,i}})$ upmost extra packets of $i$ (if $p_{i,\mathsf{OPT}}^{t_{2,i}} > l_i^{t_{1,i}}$) or no connection (if $p_{i,\mathsf{OPT}}^{t_{2,i}} \leq l_i^{t_{1,i}}$). In any of these two cases, by Lemma 4.14 and due to the third case of Fact 4.18, it follows that each of the $p_{i,\mathsf{OPT}}^{t_{2,i}^{\lambda_i}}$ extra packets that $i$ has at time step $t_{2,i}^{\lambda_i}$, has a sum of two connections and associations at $t_{2,i}^{\lambda_i}$ (that is, after all connections of time step $t_{2,i}$ have been assigned). But, by Lemma 4.16, we have:

- If $\lambda_i \in [\Lambda_i - 1]$, then the number of transmitted extra packets of $i$ in the period $[t_{2,i}^{\lambda_i}, t_{1,i}^{\lambda_i+1} - 1]$ is at most equal to the number of extra packets that $i$ has at $t_{2,i}^{\lambda_i}$, *else*

- If $\lambda_i = \Lambda_i$ and time step $t_{2,i}^{\Lambda_i}$ exists, then the number of transmitted extra packets of $i$ in the period $[t_{2,i}^{\Lambda_i}, T]$ is at most equal to the number of extra packets that $i$ has at time step $t_{2,i}^{\Lambda_i}$ (if time step $t_{2,i}^{\Lambda_i}$ does not exist, then the number of transmitted extra packets of $i$ in $[t_{1,i}^{\Lambda_i}, T]$ is, trivially, 0).

Therefore, by the third case of Fact 4.18 and since each extra packet of $i$ has a sum of two connections and associations at $t_{2,i}^{\lambda_i}$, each transmitted extra packet of $i$ has a sum of two connections and associations at its transmission time step. □

By Lemmas 4.36 and 4.37 and because the total number of transmitted extra packets by all queues in the period $[1, T]$ is $\sum_{t=1}^{T} |E^t|$, it follows that $2 \cdot \sum_{t=1}^{T} |E^t| \leq h_\sigma(\mathsf{LQD})$, for any incoming packet sequence $\sigma$.

Now, let $\Sigma$ denote the set of all inputs to three-port switches and $c \geq 1$ denote the $\mathsf{LQD}$ competitive ratio for three-port switches. Then, it holds:

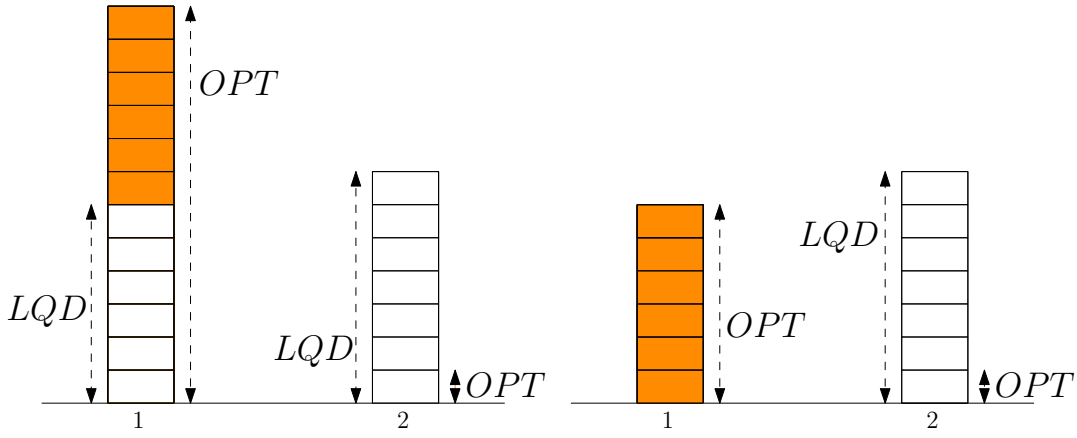


Figure 4.5: An example of the two buffer contents for the lower bound of the LQD competitive ratio for two-port switches, assuming that $M = 2k + 1$ for $k = 6$. *Left:* The OPT buffer accepts at time step $t = 1$ one packet for queue 2 and all arriving packets for queue 1 whilst LQD accepts $k + 1$ packets for queue 2 and $k$ packets for queue 1. *Right:* At time step $k+1$, the OPT buffer continues to forward two packets to the output ports whilst LQD drops to one packet; the same transmission pattern holds until each extra packet has been transmitted by OPT.

$$c = \sup_{\sigma \in \Sigma} \{r_\sigma^{\mathsf{OPT}}\} \tag{4.38}$$

By (4.5) and due to (4.38), we obtain the following theorem, completing analysis for the case of three-port shared-memory switches:

**Theorem 4.39.** LQD *is (3/2)-competitive for three-port shared-memory switches.*

### 4.2.4 The LQD algorithm for two-port switches

Throughout Section 4.2.4, we assume that $N = 2$. We shall show that LQD is exactly (4/3)-competitive for two-port shared-memory switches.

Let us start with the lower bound of the LQD competitive ratio for two-port switches. We describe the following sequence of incoming packet flow $\sigma$, for a buffer of size $M = 2k + 1$ (where $k \in \mathbb{Z}^+$). There exists a number of $M - 1$ arriving packets destined to each of the two output ports of the switch at time step $t = 1$. By Observation 4.6 and without loss of generality, we assume that

$p^1_{1,\mathsf{LQD}} = k$ and $p^1_{2,\mathsf{LQD}} = k+1$. On the other side, $\mathsf{OPT}$ accepts all packets arriving to queue 1 at $t = 1$ and only one packet for queue 2 at $t = 1$. Hence, it holds that $p^1_{1,\mathsf{OPT}} = 2k = M - 1$ and $p^1_{2,\mathsf{OPT}} = 1$.

No packet arrives to queue 1 in the period $[2, M-1]$ and one packet per time step arrives to queue 2 at each time step of the same period, which is accepted by both $\mathsf{LQD}$ and $\mathsf{OPT}$. It follows that queue 1 is active in the $\mathsf{OPT}$ buffer at each time step of the period $[1, M-2]$, that queue 2 is active in both buffers at each time step of the period $[1, M-1]$ and that queue 1 is active in the $\mathsf{LQD}$ buffer at each time step of the period $[1, k-1]$ but inactive in the $\mathsf{LQD}$ buffer at each time step of $[k, M-1]$ (see Figure 4.5).

The same pattern of incoming packet flow is repeated at $t = M$, then at $t = 2M$ and so on at each time step $t = \mu \cdot M$, where $\mu \in \mathbb{Z}^+$. More specifically, the $\mathsf{OPT}$ buffer accepts all arriving $M-1$ packets destined to the queue which $\mathsf{LQD}$ chooses to have the smaller length at the same time step, among the two active queues in the $\mathsf{LQD}$ buffer at each $t = \mu \cdot M$ time step. For $\mu \to \infty$, it follows that $h_\sigma(\mathsf{OPT}\ )/h_\sigma(\mathsf{LQD}) = 4/3$.

We, now, proceed towards deriving the upper bound of 4/3 for the $\mathsf{LQD}$ competitive ratio, for two-port switches. We shall assign valid connections to extra packets, as in the case of three-port shared-memory switches: Assume that $i \in [N]$ overflows in the $\mathsf{LQD}$ buffer at a time step $t_{1,i} = t^{\lambda_i}_{1,i}$ (for a $\lambda_i \in [\Lambda_i]$) and that $i$ becomes established for the first time after $t_{1,i}$ at $t_{2,i} = t^{\lambda_i}_{2,i}$.

Due to Lemma 4.9 and since $N = 2$ it holds $|F^{t_{1,i}}| = 1$. Each extra packet of $i$ is assigned two valid connections at $t_{1,i}$: The first connection to each extra packet of $i$ is with a different common packet of the same queue $i$. The number of common packets suffices for these connections to be assigned, since $i$ is a secondary dominating queue at $t_{1,i}$ due to Lemma 4.25 (note that a dominating queue that is not primary, can only be secondary dominating). The second connection to each

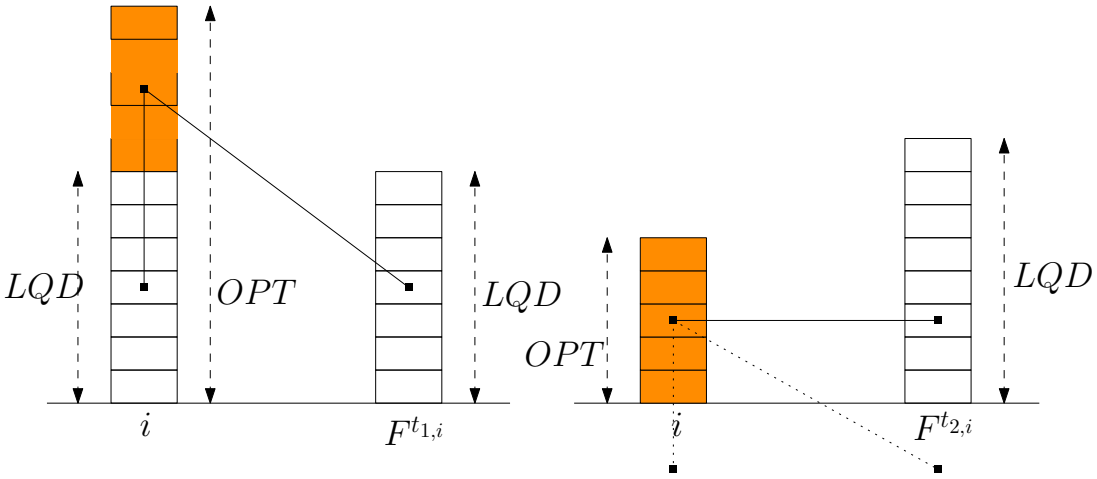


Figure 4.6: *Left:* Two connections are assigned to each extra packet of $i$ at $t_{1,i}$, one of which is with a common packet of $i$ and the other with a free packet. All connections are assigned with different LQD packets so that Invariant 4.30 holds at $t_{1,i}$. *Right:* A connection to each extra packet of $i$ is assigned at $t_{2,i} > t_{1,i}$ with a free packet.

extra packet of $i$ is with a different free packet at $t_{1,i}$. All connections assigned are valid due to Observation 4.19.

Wrapping up, we assigned two valid connections to each extra packet of $i$ at $t_{1,i}$, so that Invariant 4.30 holds at this time step (see Figure 4.6).

Before continuing, we need the following lemma:

**Lemma 4.40.** *No packet in the* LQD *buffer at time step $t_{2,i} > t_{1,i}$ may be connected with a connection which was assigned at time step $t_{1,i}$.*

*Proof.* As in the proof of Lemma 4.31, it suffices to show that the maximum queue position at $t_{1,i}$ of any LQD packet that is assigned a connection at $t_{1,i}$ is $p_{i,\mathsf{LQD}}^{t_{1,i}}$. This, obviously, holds for each common packet of $i$ that is assigned a connection at $t_{1,i}$, as in the proof of Lemma 4.31.

Now, for the free packets that are assigned connections at $t_{1,i}$, it suffices to show that we do not need to assign a connection at $t_{1,i}$ to the free packet that may be located at a queue position equal to $p_{i,\mathsf{LQD}}^{t_{1,i}} + 1$ at this time step. Note that no

free packet may be located at a queue position strictly greater than $p_{i,\mathsf{LQD}}^{t_{1,i}} + 1$ at $t_{1,i}$, since $i$ overflows at $t_{1,i}$ in the LQD buffer and due to Observation 4.1. But, by Lemma 4.9 and since $|F^{t_{1,i}}| = 1$, the number of free packets in the LQD buffer at $t_{1,i}$ is at least equal to the number of extra packets of $i$ at $t_{1,i}$, plus 1. Therefore, we do not need to assign a connection to the free packet that may be located at a queue position equal to $p_{i,\mathsf{LQD}}^{t_{1,i}} + 1$ at $t_{1,i}$. It follows that the maximum queue position at $t_{1,i}$ of any free packet which is assigned a connection at $t_{1,i}$ is $p_{i,\mathsf{LQD}}^{t_{1,i}}$, completing the proof. □

Therefore, at $t_{2,i}$ we assign one valid connection to each extra packet of $i$ at this time step, with a different free packet. The number of free packets suffices, due to Lemma 4.9. By Lemma 4.40, Invariant 4.30 is not violated by any connection assigned at time step $t_{2,i}$.

**Upper-bounding the LQD competitive ratio when $N = 2$**

Our analysis for the case of two-port switches, is similar to the analysis for the case of three-port switches.

**Lemma 4.41.** *Invariant 4.30 holds at every time step of* $[1, T]$.

*Proof.* As in the case of the proof of Lemma 4.36, Invariant 4.30 may be violated only at a time step when a connection between an extra packet and an already connected LQD packet is assigned. But we showed that at any time step when a connection is assigned to an extra packet of $i \in [N]$ (either $t_{1,i}$ or $t_{2,i}$), Invariant 4.30 holds. □

**Lemma 4.42.** *Each transmitted extra packet in the period* $[1, T]$ *has a sum of three connections and associations at its transmission time step.*

*Proof.* An extra packet that is transmitted at a time step $t \in [T]$ belongs to a queue $i \in [N]$ that was dominating at the latest time step before $t$, when $i$ overflowed in

the LQD buffer, due to Observation 4.6 and Lemma 4.7, i.e., this time step is $t_{1,i}^{\lambda_i}$ for some $\lambda_i \in [\Lambda_i]$.

Each extra packet of $i$ at $t_{1,i}^{\lambda_i}$ is assigned two connections at this time step and each extra packet of $i$ at $t_{2,i}^{\lambda_i}$ is assigned one connection at this time step, according to the connection assignment procedure that we described in Section 4.2.4. By Lemma 4.15 and due to Fact 4.18, it follows that each extra packet of $i$ at $t_{2,i}^{\lambda_i}$ should have a sum of three connections and associations at this time step (that is, after the connections of time step $t_{2,i}$ have been assigned). By Lemma 4.16, the proof is complete. □

By Lemmas 4.41 and 4.42, it follows that $3 \cdot \sum_{t=1}^{T} |E^t| \leq h_\sigma(\mathsf{LQD})$, for any incoming packet sequence $\sigma$. Therefore, by (4.5) and since the LQD competitive ratio for two-port shared-memory switches equals $\sup_{\sigma \in \Sigma}\{r_\sigma^{\mathsf{OPT}}\}$, where $\Sigma$ denotes here the set of all inputs to two-port shared-memory switches, we have that LQD is (4/3)-competitive when $N = 2$.

Taking into consideration the lower bound of (4/3) for two-port switches that we described in the beginning of Section 4.2.4, we have the following theorem, completing analysis for Chapter 4.

**Theorem 4.43.** LQD *is exactly* (4/3)*-competitive for two-port shared-memory switches.*

## 4.3 Open Problems

There exists a significant gap between the upper bound of 2 and the lower bound of $\sqrt{2}$ for the competitive ratio of LQD, for shared-memory switches equipped with an arbitrary number of output ports $N \geq 2$. Therefore, an interesting open problem is to show a $(2-\epsilon)$ upper bound for the LQD competitive ratio for a positive constant $\epsilon$ or to improve the lower bound of $\sqrt{2}$. We conjecture that LQD is $(2-\epsilon)$-competitive, for a constant $\epsilon > 0$.

Regarding the special case of $N = 3$ output ports, an open problem is to improve on the LQD competitive ratio upper bound of $3/2$ that we showed here or the lower bound of $4/3$ which already holds for the special case of $N = 2$ output ports and, therefore, for $N = 3$.

# Bibliography

W. Aiello, Y. Mansour, S. Rajagopolan, and A. Rosén. Competitive Queue Policies for Differentiated Services. In *Proceedings of the 19th IEEE International Conference on Computer Communications (INFOCOM)*, pages 431–440, 2000.

W. Aiello, A. Kesselman, and Y. Mansour. Competitive Buffer Management for Shared-Memory Switches. *ACM Transactions on Algorithms (TALG)*, 5(1), 2008.

S. Albers. Better Bounds for Online Scheduling. *SIAM Journal on Computing (SICOMP)*, 29(2):459–473, 1999.

S. Albers, B. von Stengel, and R. Werchner. A Combined BIT and TIMESTAMP Algorithm for the List Update Problem. *Information Processing Letters (IPL)*, 56(3):135–139, 1995.

Y. Azar, A. Z. Broder, and A. R. Karlin. On-line Load Balancing. *Theoretical Computer Science*, 130(1):73–84, 1994.

Y. Azar, J. Naor, and R. Rom. The Competitiveness of On-Line Assignments. *Journal of Algorithms (JALG)*, 18(2):221–237, 1995.

J. Balogh, J. Békési, and G. Galambos. New Lower Bounds for Certain Classes of Bin Packing Algorithms. In *Proceedings of the 8th Workshop on Approximation and Online Algorithms (WAOA)*, pages 25–36, 2010.

J. Balogh, J. Békési, G. Dósa, H. Kellerer, and Z. Tuza. Black and White Bin Packing. In *Proceedings of the 10th Workshop on Approximation and Online Algorithms (WAOA)*, pages 131–144, 2012.

J. Balogh, J. Békési, G. Dósa, L. Epstein, H. Kellerer, A. Levin, and Z. Tuza. Offline black and white bin packing. *Theoretical Computer Science*, 596:92–101, 2015a.

J. Balogh, J. Békési, G. Dósa, L. Epstein, H. Kellerer, and Z. Tuza. Online Results for Black and White Bin Packing. *Theory of Computing Systems*, 56(1):137–155, 2015b.

J. Balogh, J. Békési, G. Dósa, J. Sgall, and R. van Stee. The optimal absolute ratio for online bin packing. In *Proceedings of the 26th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1425–1438, 2015c.

N. Bansal and A. Khan. Improved Approximation Algorithm for Two-Dimensional Bin Packing. In *Proceedings of the 25th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 13–25, 2014.

N. Bansal, N. Buchbinder, and J. Naor. Towards the Randomized $k$-Server Conjecture: A Primal-Dual Approach. In *Proceedings of the 21st ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 40–55, 2010.

N. Bansal, N. Buchbinder, A. Madry, and J. Naor. A Polylogarithmic-Competitive Algorithm for the $k$-Server Problem. In *Proceedings of the 52nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 267–276, 2011.

Y. Bartal, A. Fiat, H. J. Karloff, and R. Vohra. New Algorithms for an Ancient Scheduling Problem. *Journal of Computer and System Sciences (JCSS)*, 51(3): 359–366, 1995.

S. Ben-David, A. Borodin, R. M. Karp, G. Tardos, and A. Wigderson. On the Power

of Randomization in Online Algorithms (extended abstract). In *Proceedings of the 22nd ACM Symposium on Theory of Computing (STOC)*, pages 379–386, 1990.

C. Berge. Two Theorems in Graph Theory. *Proceedings of the National Academy of Sciences of the United States of America (PNAS)*, 43(9):842–844, 1957.

M. Böhm, J. Sgall, and P. Veselý. Online Colored Bin Packing. In *Proceedings of the 12th Workshop on Approximation and Online Algorithms (WAOA)*, pages 35–46, 2014.

A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

N. Buchbinder and J. Naor. Online Primal-Dual Algorithms for Covering and Packing. *Mathematics of Operations Research (MOR)*, 34(2):270–286, 2009a.

N. Buchbinder and J. Naor. The Design of Competitive Online Algorithms via a Primal-Dual Approach. *Foundations and Trends in Theoretical Computer Science (FnT-TCS)*, 3(2-3):93–263, 2009b.

H. J. Chao and X. Guo. *Quality of Service Control in High-Speed Networks*. Wiley-IEEE Press, 2001.

H. J. Chao and B. Liu. *High Performance Switches and Routers*. Wiley-IEEE Press, 2007.

S. A. Cook. The Complexity of Theorem-Proving Procedures. In *Proceedings of the 3rd ACM Symposium on Theory of Computing (STOC)*, pages 151–158, 1971.

M. Dawande, J. Kalagnanam, and J. Sethuraman. Variable Sized Bin Packing With Color Constraints. *Electronic Notes in Discrete Mathematics (ENDM)*, 7:154–157, 2001.

W. Fernandez de la Vega and G. S. Lueker. Bin Packing can be Solved within $1+\epsilon$ in Linear Time. *Combinatorica*, 1(4):349–355, 1981.

G. Dósa and L. Epstein. Colorful bin packing. In *Proceedings of the 14th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, pages 170–181, 2014.

G. Dósa and J. Sgall. First Fit bin packing: A tight analysis. In *Proceedings of the 30th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 538–549, 2013.

M. Englert, N. Matsakis, and M. Mucha. New Bounds for Online Packing LPs. In *Proceedings of the 11th Latin American Theoretical Informatics Symposium (LATIN)*, pages 318–329, 2014.

L. Epstein. Bin packing with rejection revisited. In *Proceedings of the 4th Workshop on Approximation and Online Algorithms (WAOA)*, pages 146–159, 2006.

L. Epstein and A. Levin. AFPTAS results for common variants of bin packing: A new method for handling the small items. *SIAM Journal on Optimization (SIOPT)*, 20(6):3121–3145, 2010.

P. Eugster, K. Kogan, S. Nikolenko, and A. Sirotkin. Shared Memory Buffer Management for Heterogeneous Packet Processing. In *Proceedings of the 34th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 471–480, 2014.

P. Eugster, A. Kesselman, K. Kogan, S. Nikolenko, and A. Sirotkin. Essential Traffic Parameters for Shared Memory Switch Performance. In *Proceedings of the 22nd International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 61–75, 2015.

A. Fiat and G. J. Woeginger. *Online Algorithms, The State of the Art*. Springer, 1998.

M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

M. H. Goldwasser. A Survey of Buffer Management Policies for Packet Switches. *Special Interest Group on Algorithms and Computation Theory (SIGACT) News*, 41(1):100–128, 2010.

R. L. Graham. Bounds on Multiprocessing Timing Anomalies. *SIAM Journal of Applied Mathematics (SIAP)*, 17(2):416–429, 1969.

E. L. Hahne, A. Kesselman, and Y. Mansour. Competitive Buffer Management for Shared-Memory Switches. In *Proceedings of the 13th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 53–58, 2001.

K. Jansen and S. R. Öhring. Approximation Algorithms for Time Constrained Scheduling. *Information and Computation*, 132(2):85–108, 1997.

D. S. Johnson. *Near-Optimal Bin Packing Algorithms.* PhD thesis, Massachusetts Institute of Technology, 1973.

A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator. Competitive Snoopy Caching. *Algorithmica*, 3:77–119, 1988.

N. Karmarkar and R. M. Karp. An Efficient Approximation Scheme For The One-Dimensional Bin-Packing Problem. In *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 312–320, 1982.

R. M. Karp. Reducibility Among Combinatorial Problems. In *Proceedings of a Symposium on the Complexity of Computer Computations at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York*, pages 85–103, 1972.

A. Kesselman and Y. Mansour. Harmonic buffer management policy for shared memory switches. *Theoretical Computer Science*, 324(2-3):161–182, 2004.

A. Kesselman, Z. Lotker, Y. Mansour, B. Patt-Shamir, B. Schieber, and M. Sviridenko. Buffer Overflow Management in QoS Switches. In *Proceedings of the 33rd ACM Symposium on Theory of Computing (STOC)*, pages 520–529, 2001.

K. M. Kobayashi, S. Miyazaki, and Y. Okabe. A Tight Bound on Online Buffer Management for Two-port Shared-Memory Switches. In *Proceedings of the 19th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 358–364, 2007.

S. O. Krumke. Online Optimization. Optimization and its Applications in Learning and Industry (OptALI) notes, Auckland, 2011.

C. C. Lee and D. T. Lee. A Simple On-Line Bin-Packing Algorithm. *Journal of the ACM (JACM)*, 32(3):562–572, 1985.

F. M. Liang. A Lower Bound for On-line Bin Packing. *Information Processing Letters (IPL)*, 10(2):76–79, 1980.

Z. Lotker and B. Patt-Shamir. Nearly optimal FIFO buffer management for Diff-Serv. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC)*, pages 134–142, 2002.

S. Martello, D. Pisinger, and D. Vigo. The Three-Dimensional Bin Packing Problem. *Operations Research*, 48(2):256–267, 2000.

N. Matsakis. LQD is 1.5-competitive for 3-port Shared-Memory Switches. In *Proceedings of the 42nd International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM) - Student Research Forum*, 2016.

M. Ochel, K. Radke, and B. Vöcking. Online Packing with Gradually Improving Capacity Estimations and Applications to Network Lifetime Maximization. In *Proceedings of the 39th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 648–659, 2012.

P. Raghavan and M. Snir. Memory Versus Randomization in On-line Algorithms (extended abstract). In *Proceedings of the 16th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 687–703, 1989.

P. V. Ramanan, D. J. Brown, C. C. Lee, and D. T. Lee. On-Line Bin Packing in Linear Time. *Journal of Algorithms (JALG)*, 10(3):305–326, 1989.

T. Rothvoß. Approximating Bin Packing within $\mathcal{O}(\log \mathsf{OPT} \cdot \log\log \mathsf{OPT})$ bins. In *Proceedings of the 54th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 20–29, 2013.

S. S. Seiden. On the Online Bin Packing Problem. *Journal of the ACM (JACM)*, 49(5):640–671, 2002.

D. Simchi-Levi. New Worst-Case Results for the Bin-Packing Problem. *Naval Research Logistics (NRL)*, 41(4):579–585, 1994.

D. D. Sleator and R. E. Tarjan. Amortized Efficiency of List Update and Paging Rules. *Communications of the ACM (CACM)*, 28(2):202–208, 1985.

J. D. Ullman. The Performance of a Memory Allocation Algorithm. *Technical Report 100, Princeton University*, page 78, 1971.

A. van Vliet. An Improved Lower Bound for On-line Bin Packing Algorithms. *Information Processing Letters (IPL)*, 43(5):277–284, 1992.

V. V. Vazirani. *Approximation Algorithms*. Springer, 2001.

V. G. Vizing. On an Estimate of the Chromatic Class of a $p$-Graph. *Diskret. Analiz.*, 3:25–30, 1964.

J. von Neumann. Zur Theorie der Gesellschaftsspiele (*in German)*. *Mathematische Annalen*, 100:295–300, 1928.

I. Wegener. *Complexity Theory - Exploring the Limits of Efficient Algorithms.* Springer, 2005.

S. X. Wei, E. J. Coyle, and M. T. Hsiao. An Optimal Buffer Management Policy for High-Performance Packet Switching. In *Proceedings of the Global Communication Conference (GLOBECOM)*, pages 924–928, 1991.

D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms.* Cambridge University Press, 2011.

A. C. Yao. Probabilistic Computations: Toward a Unified Measure of Complexity. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 222–227, 1977.

A. C. Yao. New Algorithms for Bin Packing. *Journal of ACM (JACM)*, 27(2): 207–227, 1980.