Online algorithms for combinatorial problems

Ph.D. Thesis

by

Judit Nagy-György

Supervisor: Péter Hajnal
Associate Professor

Doctoral School in Mathematics and Computer Science
University of Szeged
Bolyai Institute

2009

Contents

List of Figures

Chapter 1

Introduction: competitive analysis

Online algorithms have been investigated for approximately 30 years. One of the main methods to measure the performance of online algorithms is a worst case analysis called competitive analysis. Its roots can be found in combinatorial optimization theory, in particular Graham's work [23]. In an online problem the parts of the input sequence (input set equipped with an ordering) appear one by one and the *online algorithm* must produce a sequence of decisions about these parts that will have an impact on the final quality of its overall performance. Each of these decisions must be made based on the already appeared part of the input sequence without any information about the future (for more details see [11]).

To introduce some notions we begin with discussion of optimization problems which can be either cost minimization or profit maximization. Here we will consider the former case. An *optimization problem* of cost minimization consists of a set I of inputs and a cost function. Associated with every input I is a set of feasible outputs, and associated with each feasible output is a positive real representing the cost of the output with respect to I, among which the minimal is denoted by $\mathrm{opt}(I)$. Given any legal input I an algorithm $\mathcal{A}$ computes a feasible output. The cost associated with this output is denoted by $\mathcal{A}(I)$.

An online algorithm $\mathcal{A}$ is called *(strictly) c-competitive* for some $c > 0$ if for all finite input sequences

$$\mathcal{A}(I) \leq c \cdot \mathrm{opt}(I).$$

The competitive ratio of $\mathcal{A}$ is the smallest c such that $\mathcal{A}$ is c-competitive. An

algorithm $\mathcal{A}$ is *weakly c-competitive* if there is a constant a such that for all finite input sequence I,

$$\mathcal{A}(I) \leq c \cdot \mathrm{opt}(I) + a.$$

The weak competitive ratio of $\mathcal{A}$ is the smallest c such that $\mathcal{A}$ is c-competitive.

There are several ways to view online problems. One is to consider them as a game between an *online player* and a malicious *adversary*. The online player runs an online algorithm on an input that is created by the adversary. The adversary's goal is to construct the worst possible input maximizing the competitive ratio, based only on knowledge of the algorithm used by the online player. That is, the adversary tries to make the task expensive to the online player, but, at the same time, maintaining the optimal cost low. The adversary is often identified with an algorithm providing the best possible (optimal) cost.

For deterministic online algorithms the adversary knows what the online player's response will be to each input element. So it does not matter whether the adversary has to construct the whole input sequence in advance or has the possibility to construct it piecewise, after the response of the player. For randomized online algorithms it does not hold, therefore the nature of the adversaries can be defined several ways (see [9] for a comparison).

- The first and most frequently used variant is the *oblivious adversary*, which must construct the input sequence in advance based only on the description of the online algorithm and pays the optimal cost.
- A much stronger version is the *adaptive online adversary*: makes the next request based on the algorithm's answers to the previous ones, but also serves it immediately.
- The strongest one is the *adaptive offline adversary*: makes the next request based on the algorithm's answers to the previous ones, but serves them optimally at the end.

We say that a randomized online algorithm $\mathcal{A}$ is c-competitive against an adversary if there exists a constant a such that the expected difference of the cost of $\mathcal{A}$ and c times the adversary's cost is at most a for any finite input sequence. By the following theorem the adaptive offline adversary is so strong that randomization adds no power against it.

Theorem 1 (Ben-David et al. [9]) *If there is a randomized algorithm that is c-competitive against any adaptive offline adversary then there also exists an c-competitive deterministic algorithm.*

We will play against oblivious adversary so we call a randomized online algorithm *c-competitive* if it is c-competitive against the oblivious adversary (recall that the cost of the oblivious adversary is the optimal cost), i.e. if there exists a constant a such that for any input sequence I

$$\mathrm{E}[\mathcal{A}(I)] \leq c \cdot \mathrm{opt}(I) + a.$$

The competitive ratio of $\mathcal{A}$ is the smallest c for which $\mathcal{A}$ is c-competitive.

The first family of problems we will consider is the family of online scheduling, in which jobs are given with processing time and have to be assigned to machines handling them. Jobs arrive one by one, and they have to be scheduled immediately at their arrival. The cost is the processing time of the most loaded machine plus extra costs. In our model one may purchase machines and reject jobs by paying some penalty. We will construct a constant-competitive algorithm in this model.

The second family, the online graph coloring, is a well studied problem. The vertices of the input graph are revealed one by one and have to be colored immediately. The cost of an algorithm is obviously the number of colors it uses. We will consider two families of graphs with n vertices and generalize known algorithms. We will give upper bounds of their competitive ratio depending on n and the parameter of the family of graphs. Moreover, we will generalize the online graph coloring problem to a kind of hypergraph coloring and prove lower and upper bounds of competitive ratios in several families of hypergraphs.

One of the best-known online problems is the online k-server problem in which k servers are given, occupying different points of a metric space. The input is a request sequence, also consisting of the points of the metric space. We have to serve each request by moving a server there. The cost is the sum of the distances covered by the k servers. We will consider a randomized version. Usually a randomized online algorithm for the k-server problem is called c-competitive if it is c-competitive in the above sense with a depending (only) on the initial configuration of the servers (i.e., neither the metric space nor the initial configuration are parts of the input). We will present a randomized online algorithm on so-called decomposable spaces

which is $o(k)$-competitive on specific metric spaces called HSTs with small height. We also define another type of the problem in which one can reject some requests by paying some penalty. For this model we will present a (deterministic) online algorithm on uniform spaces (i.e., spaces where the distance is 1 between any two points) and prove that its competitive ratio is the best possible apart from some constant additive term a.

We will use the following well-known notations.

- $[n] = \{1, 2, \ldots, n\}$ stands for the set of the first n positive integers.
- $\varphi = \frac{1+\sqrt{5}}{2}$ is the golden ratio.
- $\mathrm{H}_k = \sum_{i=1}^{k} i^{-1}$ is the kth harmonic number.
- $\Pr(\mathcal{E})$ is the probability of event $\mathcal{E}$.
- $\mathrm{E}[\eta]$ is the expected value of the random variable η.
- C_n is a cycle (graph) of length n.
- $K_{n,n}$ is a complete bipartite graph with color classes of size n.
- o, O, Ω and Θ are the Bachmann–Landau notations.

Chapter 2

Scheduling with machine cost and rejection

2.1 Preliminaries

The area of scheduling theory has large literature and several models (see in [40]). In one of the most fundamentals and simplests we have a fixed number of machines and the jobs arrive from a list (list model). The ith job has *processing time* p_i. We consider the "parallel machines case" where m machines are given. To schedule a job we have to assign it to a machine. We have to schedule each job and no machine may simultaneously run two jobs. By the load of a machine we mean the sum of processing times of all jobs assigned to it and the *makespan* is the maximum of loads. The cost is the makespan, therefore our goal is to minimize it.

In the online version of the problem the jobs and their processing times are revealed one by one. When a job is revealed the online algorithm has to assign to a machine without any information about the further jobs. Algorithm $\mathcal{LIST}$ is the first algorithm in this model has been developed by Graham [23] in 1966.

Algorithm $\mathcal{LIST}$

When the ℓth job is revealed assign it to the machine where the actual load is minimal.

In fact, Algorithm $\mathcal{LIST}$ tries to balance the loads of the machines.

Theorem 2 (Graham [23]) *The competitive ratio of Algorithm $\mathcal{LIST}$ is $2 - 1/m$.*

2.1.1 Scheduling with machine cost

In machine scheduling usually there is a fixed set of machines and a given set of jobs must be scheduled on the machines. In the last few years some generalized models were investigated where it is allowed to change the set of machines.

The problem of scheduling with machine cost is defined in [25]. In this model the number of machines is not a given parameter of the problem: the algorithm has to purchase the machines, and the goal is to minimize the cost spent for purchasing the machines plus the makespan. In [25] the problem where each machine has cost 1 is investigated. It can be supposed without loss of generality that the machines have cost 1, any constant cost can be reduced to this problem by scaling the processing times. The jobs arrive one by one and the decision maker has to decide in each step whether to buy new machines and then schedule the job on one of the already purchased machines without any information about the further jobs. Imreh and Noga [25] presented the following algorithm. $\rho = (0 = \rho_1, \rho_2, \dots, \rho_i, \dots)$ is an increasing sequence.

Algorithm $\mathcal{A}_\rho$

(i) When the ℓth job is revealed $\mathcal{A}_\rho$ purchases machines (if necessary) so that the current number of machines i satisfies $\rho_i \leq \sum_{j=1}^{\ell} p_j < \rho_{i+1}$.

(ii) We schedule the jth job on a least loaded machine, according to the $\mathcal{LIST}$ algorithm.

Theorem 3 (Imreh, Noga [25]) *The competitive ratio of Algorithm $\mathcal{A}_\rho$ is φ for $\rho = (0, 4, 9, 16, \dots, i^2, \dots)$.*

They showed also that no online algorithm can have smaller competitive ratio than 4/3. In [16] the problem is further investigated and a 1.5798-competitive algorithm is presented.

2.1.2 Scheduling with rejection

In the original scheduling problem the algorithm has to schedule each job. The problem of scheduling with rejection is defined in [6]. In this model, it is possible to reject the jobs. The jobs are characterized by a *processing time* and a *penalty*. In this problem the cost is the sum of the makespan and the penalties of all rejected jobs.

Denote p_i the processing time of ith job and w_j its penalty. Bartal et al. [6] developed algorithm Reject-Total-Penalty(α). It has a parameter α which plays the role of a threshold. In the j-th step R_j denotes the set of the indices of the rejected jobs, moreover $R_{j,m} = \{i \mid i \in R_j, w_i > p_i/m\}$.

Algorithm $\mathcal{RTP}(\alpha)$

jth *step:*

(i) If $w_j \leq p_j/m$, we reject the jth job.

(ii) If $w_j > p_j/m$, and $w_j + \sum_{i \in R_{j-1,m}} w_i \leq \alpha p_j$, we reject the jth job.

(iii) Otherwise, we schedule it on a least loaded machine, according to the $\mathcal{LIST}$ algorithm.

Theorem 4 (Bartal et al. [6]) *The Algorithm* $\mathcal{RTP}(\varphi - 1)$ *is* $(1 + \varphi)$*-competitive.*

Bartal et al. [6] also proved that there is no online algorithm that is c-competitive for some constant $c < 1 + \varphi$ and all m.

2.1.3 Scheduling with machine cost and rejection

The results of the rest of the chapter can be found in [34].

Hereinafter we consider a more general model MCR combining the above two approaches. Here the machines are not given to the algorithm in advance but the algorithm must purchase them, and the jobs can be rejected. Here the cost is the makespan plus the cost of purchasing the machines plus the sum of the penalties of the rejected jobs so our goal is to minimize it. We suppose that each machine has cost 1. We call the total cost of purchasing the machines *machine purchasing cost*.

Consider an arbitrary list of jobs and denote the set of its indices by J. Let $J' \subseteq J$. For the sake of convenience we denote by $\mathcal{A}(J')$ the cost of the

schedule produced by algorithm $\mathcal{A}$ on input list generated by J', the cost of the optimal schedule is denoted by $\text{opt}(J')$.

In the problem the jth job has a processing time p_j and a penalty which is the cost of rejecting it, denoted by w_j. For a set $H \subseteq J$ we make use of the notations $P_H = \sum_{j \in H} p_j$ and $W_H = \sum_{j \in H} w_j$. As a shorthand we denote $P_{\{1,\ldots,\ell\}}$ by simply writing P_ℓ.

In the following we will introduce several algorithms. We note that in [18] the authors observe that the problem is a generalization of the Ski-Rental Problem [37] which can be described as follows: A sportsman can either rent a pair of skis, in this case he must pay 1 unit of money by each occasion, or he can buy a pair of skis for N unit of money. When should he buy the pair of skis to pay totally the least money? This problem is equivalent to the very special case of the MCR problem, where all jobs have size 0, and penalty $1/N$. It is well known that no algorithm with smaller competitive ratio than 2 exists for the solution of the Ski-Rental Problem and therefore neither for MCR problem.

2.2 Results on MCR

In this section we develop and analyze some algorithms for the solution of the problem. Since we have rules for purchasing the machines and for the rejection and scheduling of the jobs it is a straightforward idea to combine these rules and build algorithms for the complex problem. In the first part we show the surprising result that the simple combinations of these rules are not constant competitive.

2.2.1 Mixed algorithms

In the following algorithms, α is a given constant, $\rho = (0 = \rho_1, \rho_2, \ldots, \rho_i, \ldots)$ is an increasing sequence. In the j-th step A_j denotes the set of the indices of accepted jobs, R_j denotes the set of the indices of the rejected ones, moreover $R_{j,m} = \{i \,|\, i \in R_j, w_i > p_i/m\}$ and $R_{j,0} = \{i \,|\, i \in R_j, w_i > p_i\}$. In the j-th step A_j denotes the set of the indices of accepted jobs and R_j the set of the rejected ones. In all cases we start with 0 machines.

1st combined algorithm ($\mathcal{CA}1$).

jth *step:*

(i) When the jth job appears, we purchase machines (if necessary) so that the current number of machines m satisfies $\rho_m \leq P_{A_{j-1}\cup\{j\}} < \rho_{m+1}$.
(ii) If $w_j \leq p_m/m$, we reject the jth job.
(iii) If $w_j > p_m/m$, and $W_{R_{j-1,m}} + w_j \leq \alpha p_j$, we also reject it.
(iv) Otherwise, we schedule it on a least loaded machine, according to Algorithm $\mathcal{LIST}$.

Proposition 5 *There is no such c for which algorithm $\mathcal{CA}1$ is c-competitive.*

Proof. Assume that $\mathcal{CA}1$ is c-competitive for some $c > 0$. Let $n > c$, $J = \{1\}$, $p_1 = \rho_{n+1}$ and $w_1 = 1$. For this job, the optimal schedule rejects it and $\text{opt}(J) = 1$ holds. Algorithm $\mathcal{CA}1$ also rejects it, but it purchases $n + 1$ machines; so its cost $\mathcal{CA}1(J) = n + 2 > n > c \cdot \text{opt}(J)$, from the constraint $n > c$. From this contradiction follows that $\mathcal{CA}1$ is not c-competitive. □

We also investigate the following similar algorithm which can handle the counterexample given above.

2th combined algorithm ($\mathcal{CA}2$).

jth *step:*

(i) When the jth job appears, we compute the number m such that $\rho_m \leq P_{A_{j-1}\cup\{j\}} < \rho_{m+1}$ holds.
(ii) If $w_j \leq p_m/m$, we reject the jth job.
(iii) If $w_j > p_m/m$, and $W_{R_{j-1,m}} + w_j \leq \alpha p_j$, we also reject it.
(iv) Otherwise if necessary, we purchase machines so that the current number of them reaches m; after that, we schedule it on a least loaded machine, according to Algorithm $\mathcal{LIST}$.

Proposition 6 *There is no such c for which algorithm $\mathcal{CA}2$ is c-competitive.*

Proof. Assume that $\mathcal{CA}2$ is c-competitive for some $c > 0$. Let n and k be two integers such that $n > 2c$ and $\rho_2/2 \leq n/k < \rho_2$. Furthermore, let $|J| = kn$, and for all $j \in J$ let $p_j = w_j = n/k$. If we purchase n machines and schedule k jobs on each of them, the cost will be $n + k(n/k) = 2n$. From this we can conclude $\text{opt}(J) \leq 2n$. Since algorithm $\mathcal{CA}2$ rejects all the jobs, its cost is $\mathcal{CA}2(J) = kn(n/k) = n^2$. From the constraint $n > 2c$, $n^2 > 2cn$ holds, so $\mathcal{CA}2$ is not c-competitive. □

3rd combined algorithm ($\mathcal{CA}3$).

jth *step:*

(i) Let m be the actual number of the machines. If $w_j \leq p_m/m$, we reject the jth job.

(ii) If $w_j > p_m/m$, and $W_{R_{j-1,m}} + w_j \leq \alpha p_j$, we also reject it.

(iii) Otherwise if necessary, we purchase machines so that the number of them m satisfies $\rho_m \leq P_{A_{j-1} \cup \{j\}} < \rho_{m+1}$. After that, we schedule it on a least loaded machine, according to Algorithm $\mathcal{LIST}$.

Proposition 7 *There is no such c for which algorithm $\mathcal{CA}3$ is c-competitive.*

Proof of Proposition 6 can also be applied to this case.

2.2.2 Algorithm Optcopy

In this section we present a more sophisticated algorithm. The basic idea is that instead of the original problem we consider a relaxed version, where we replace part of the cost of the schedule (purchasing cost of machines plus the makespan) with a lower bound of it.

Suppose that we accepted a set of jobs, denote the set of their indices by A, furthermore m machines were purchased, and the current makespan is M. Then $Mm \geq P_A$, thus $m \geq P_A/M$. So we obtain that for the cost of the schedule $M + m \geq M + P_A/M$ is valid. Let l_A denote the greatest processing time that belongs to a job with indices in A. We define the following expression:

$$M_A := \begin{cases} \max\{\sqrt{P_A}, l_A\}, & \text{if } P_A > 1 \\ 1 & \text{otherwise} \end{cases}$$

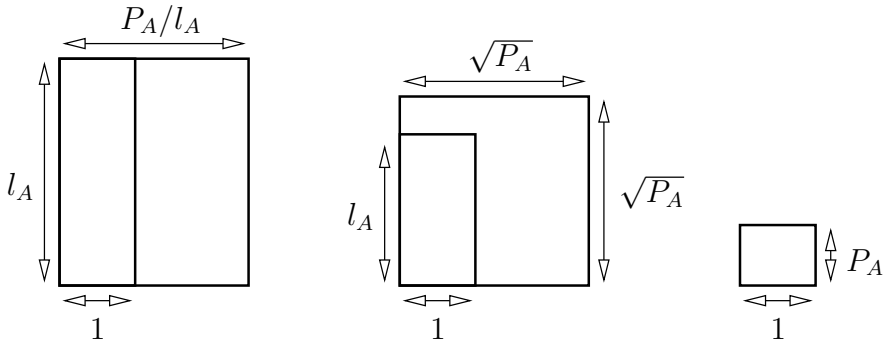
Concerning the value of M_A the following statement comes immediately by the definition.

Lemma 8 *For two arbitrary sets A_1 and A_2 of indices, if $A_1 \subseteq A_2$ then $M_{A_1} \leq M_{A_2}$.*

Now for an arbitrary set A of indices let

$$T_A := \begin{cases} M_A + \dfrac{P_A}{M_A} & \text{if } A \neq \emptyset \\ 0 & \text{if } A = \emptyset \end{cases}$$

The geometrical meaning of T_A is the following: if we consider the jobs as rectangles with sides 1 and p_i, then $2T_A$ is the smallest possible perimeter of the rectangles which can be used to pack the rectangles assigned to the jobs. Figure 2.1 shows the possible such rectangles.



(a) $1 \leq \sqrt{P_A} < l_A$ (b) $1, l_A \leq \sqrt{P_A}$ (c) $0 < P_A < 1$

Figure 2.1: Possible rectangles to pack

By this interpretation we can prove easily the following statements.

Lemma 9 *For two arbitrary sets A_1 and A_2 of indices, if $A_1 \subseteq A_2$ then $T_{A_1} \leq T_{A_2}$.*

Lemma 10 *Let A be an arbitrary nonempty set and $x \geq \max\{1, l_A\}$ an arbitrary positive number. Then*

$$x + \frac{P_A}{x} \geq T_A.$$

Using Lemma 10 we immediately obtain the following statement for the case where rejection is not allowed (also proven in [25]).

Lemma 11 *[25] The cost of an optimal schedule with machine cost of the jobs with indices from set A when no rejection is allowed is at least T_A.*

In [25], Theorem 2 proves that Algorithm $\mathcal{A}_\rho$ with the sequence $\rho = (0, 4, \dots, i^2, \dots)$ is φ competitive in the model where the rejection of the jobs is not allowed. In the proof the authors show that for an arbitrary set A of indices $\mathcal{A}_\rho(A)/\mathrm{opt}(A) \leq \varphi$. This is shown by case analysis, in each case the inequality $\mathcal{A}_\rho(A)/T_A \leq \varphi$ is proven and by Lemma 11 this shows the required statement. Therefore the same proof proves the following statement:

Lemma 12 *[25] For Algorithm $\mathcal{A}_\rho$ with the sequence $\rho(0, 4, \dots, i^2, \dots)$ and an arbitrary input set A of indices when no rejection is allowed,*

$$\mathcal{A}_\rho(A) \ \leq \ \varphi T_A.$$

Now we can define the relaxed problem. Jobs arrive, each job has a processing time and a penalty. We have to find a solution where the total penalty paid for the rejected jobs plus the value T_A for the set A of indices of accepted jobs is minimal. We call this problem *relaxed*. For a set J of indices of jobs the cost of the optimal solution of the relaxed problem is denoted by $\mathrm{ropt}(J)$. From Lemma 11 the following statement follows.

Corollary 13 *For an arbitrary set J of indices of jobs*

$$\mathrm{ropt}(J) \leq \mathrm{opt}(J).$$

Proof. Consider an optimal solution of the original problem on input J. Let A be the set of the indices of the accepted jobs, R be the set of the indices of the rejected jobs. Then by Lemma 11 we obtain that

$$\mathrm{opt}(J) \geq \sum_{j \in R} w_j + T_A.$$

On the other hand, using the sets R and A in the case of the relaxed problem the value of the objective function is $\sum_{j \in R} w_j + T_A$. Therefore we obtain a feasible solution of the relaxed problem with not larger objective function value than $\mathrm{opt}(J)$, thus the statement of the corollary follows. □

To develop algorithm Optcopy we have to examine the structure of the optimal solutions of the relaxed problem. For a set of indices J denote J_k the set of the first k indices of J. Then the following statement is valid.

Lemma 14 *Suppose that A^*_{k-1} is the set which belongs to an optimal solution of the relaxed problem on set J_{k-1}. Then the relaxed problem on set J_k has an optimal solution such that A^*_{k-1} is a subset of the set of the indices of the accepted jobs.*

Proof. Assume that there is no such optimal solution. Let A_k be the set of the indices of the accepted jobs and R_k the set of the indices of rejected jobs in an optimal solution of the relaxed problem on set J_k. As we assumed, $A^*_{k-1} \not\subseteq A_k$. Therefore $A^*_{k-1} \neq \emptyset$. We have to deal with the following two cases: when $k \in R_k$ and when $k \in A_k$.

Case 1. $k \in R_k$
If we use A_k as the index set of accepted jobs we receive a feasible solution of the relaxed problem on set J_{k-1}, therefore we obtain that

$$\text{ropt}(J_{k-1}) \leq W_{R_k \setminus \{k\}} + T_{A_k}.$$

If we substitute the definition of $\text{ropt}(J_{k-1})$ and we increase both side by w_k then we get that

$$W_{R^*_{k-1}} + w_k + T_{A^*_{k-1}} \leq W_{R_k} + T_{A_k},$$

where $R^*_{k-1} = J_{k-1} \setminus A^*_{k-1}$. On the other hand, the right side is $\text{ropt}(J_k)$ thus we obtained that

$$W_{R^*_{k-1} \cup \{k\}} + T_{A^*_{k-1}} \leq \text{ropt}(J_k).$$

Let $A^*_k := A^*_{k-1}$, that is an optimal solution naturally satisfying the property $A^*_{k-1} \subseteq A^*_k$. This is a contradiction.

Case 2. $k \in A_k$
Case 2 has two subcases: *(a)* $M_{A^*_{k-1}} > M_{A_k}$ and *(b)* $M_{A^*_{k-1}} \leq M_{A_k}$.

(a) $M_{A^*_{k-1}} > M_{A_k}$
We obtain by Lemma 8 that $M_{A^*_{k-1} \cup \{k\}} \geq M_{A^*_{k-1}}$. Then using Lemma 10 with the values $x = M_{A^*_{k-1}}$ and $A = A^*_{k-1} \cup \{k\}$ (the conditions of the lemma are satisfied since $M_{A^*_{k-1}} > M_{A_k} \geq p_k$) we obtain that

$$T_{A^*_{k-1} \cup \{k\}} \leq M_{A^*_{k-1}} + \frac{P_{A^*_{k-1} \cup \{k\}}}{M_{A^*_{k-1}}} = T_{A^*_{k-1}} + \frac{p_k}{M_{A^*_{k-1}}}.$$

On the other hand, if we use the sets R_k and $A_k \setminus \{k\}$ we have a feasible solution of the relaxed problem on set J_{k-1}, thus

$$W_{R^*_{k-1}} + T_{A^*_{k-1}} + \frac{p_k}{M_{A^*_{k-1}}} = \text{ropt}(J_{k-1}) + \frac{p_k}{M_{A^*_{k-1}}} \leq W_{R_k} + T_{A_k \setminus \{k\}} + \frac{p_k}{M_{A^*_{k-1}}}$$

Furthermore $\frac{p_k}{M_{A^*_{k-1}}} < \frac{p_k}{M_{A_k}}$ is valid and by Lemma 10 (with values $x = M_{A_k}$ and $A = A_k \setminus \{k\}$)

$$T_{A_k\setminus\{k\}} \leq M_{A_k} + \frac{P_{A_k\setminus\{k\}}}{M_{A_k}}$$

follows. Therefore we obtain that

$$W_{R_k} + T_{A_k\setminus\{k\}} + \frac{p_k}{M_{A^*_{k-1}}} < W_{R_k} + M_{A_k} + \frac{P_{A_k\setminus\{k\}}}{M_{A_k}} + \frac{p_k}{M_{A_k}} = \text{ ropt}(J_k).$$

Using the chain of inequalities proven above we obtain that

$$W_{R^*_{k-1}} + T_{A^*_{k-1}\cup\{k\}} < \text{ ropt}(J_k),$$

which is a contradiction, thus this case is not possible.

(b) $M_{A^*_{k-1}} \leq M_{A_k}$
If we use the sets $R^*_{k-1} \cup (R_k \cap A^*_{k-1})$ and $A_k \cap A^*_{k-1}$ we have a feasible solution of the relaxed problem on set J_{k-1}, thus

$$\text{ropt}(J_{k-1}) \leq W_{R^*_{k-1}} + W_{R_k\cap A^*_{k-1}} + T_{A_k\cap A^*_{k-1}}.$$

Then we apply Lemma 10 with the values $x = M_{A^*_{k-1}}$ and $A = A_k \cap A^*_{k-1}$ (the conditions hold since $M_{A^*_{k-1}} \geq M_{A_k\cap A^*_{k-1}}$ by Lemma 8), and we obtain that

$$T_{A_k\cap A^*_{k-1}} \leq M_{A^*_{k-1}} + \frac{P_{A_k\cap A^*_{k-1}}}{M_{A^*_{k-1}}},$$

therefore

$$\text{ropt}(J_{k-1}) \leq W_{R^*_{k-1}} + W_{R_k\cap A^*_{k-1}} + M_{A^*_{k-1}} + \frac{P_{A_k\cap A^*_{k-1}}}{M_{A^*_{k-1}}} \tag{2.1}$$

Using $\text{ropt}(J_{k-1}) = W_{R^*_{k-1}} + M_{A^*_{k-1}} + P_{A^*_{k-1}}/M_{A^*_{k-1}}$ and $P_{A^*_{k-1}} = P_{A_k\cap A^*_{k-1}} + P_{R_k\cap A^*_{k-1}}$, by inequality (2.1) and by the constraint of the subcase it follows

that

$$\frac{P_{R_k\cap A^*_{k-1}}}{M_{A_k}} \;\le\; \frac{P_{R_k\cap A^*_{k-1}}}{M_{A^*_{k-1}}} \;\le\; W_{R_k\cap A^*_{k-1}} \tag{2.2}$$

If we use Lemma 10 with the values $x = M_{A_k}$ and $A = A_k \cup A^*_{k-1}$ (the conditions of the lemma hold since $M_{A_k} \ge l_{A_k}$, $M_{A_k} \ge M_{A^*_{k-1}} \ge l_{A^*_{k-1}}$) then we obtain

$$W_{R_k\cap R^*_{k-1}} + T_{A_k\cup A^*_{k-1}} \le W_{R_k\cap R^*_{k-1}} + M_{A_k} + \frac{P_{A_k\cup A^*_{k-1}}}{M_{A_k}} \tag{2.3}$$

From inequality (2.2) we get

$$\begin{aligned} &W_{R_k\cap R^*_{k-1}} + M_{A_k} + \frac{P_{A_k\cup A^*_{k-1}}}{M_{A_k}} \;\; = \\ &W_{R_k\cap R^*_{k-1}} + M_{A_k} + \frac{P_{A_k\cap A^*_{k-1}} + P_{A_k\cap R^*_{k-1}} + P_{R_k\cap A^*_{k-1}} + p_k}{M_{A_k}} \;\le \\ &W_{R_k\cap R^*_{k-1}} + W_{R_k\cap A^*_{k-1}} + T_{A_k} \;\; = \;\; \mathrm{ropt}(J_k) \end{aligned} \tag{2.4}$$

Let $A^*_k := A_k \cup A^*_{k-1}$.
Using inequalities (2.3) and (2.4) it follows that A^*_k provides an optimal solution and $A^*_{k-1} \subseteq A^*_k$, what is again a contradiction. □

The relaxed problem can be solved in polynomial time. The algorithm which solves the problem is based on the following structural property.

Lemma 15 *For the jth job we consider the problem* REL(j) *which is the restricted relaxed problem where it is given that j is the index of the largest accepted job. Order the set of jobs whose indices are not larger than j by the value p_i/w_i into an increasing sequence. Then* REL(j) *has an optimal solution which is a prefix of this sequence.*

Proof. Consider the problem REL(j) for the jth job and let A and R be the sets of the indices of the accepted and rejected jobs in an optimal solution. Let $i \ne j$ be the index of the accepted job maximizing the value p_i/w_i. Since A and R are the optimal sets we obtain that

$$W_R + T_A \le W_{R\cup\{i\}} + T_{A\setminus\{i\}}$$

On the other hand, $M_A \ge M_{A\setminus\{i\}}$ thus by Lemma 10 we obtain that $M_A + (P_A - p_i)/M_A \ge T_{A\setminus\{i\}}$. Therefore

$$W_R + T_A \leq W_R + w_i + M_A + \frac{P_A - p_i}{M_A} = W_R + T_A + w_i - \frac{p_i}{M_A}.$$

Thus we obtained that $p_i/w_i \leq M_A$.

Now suppose that the solution does not satisfy the property stated in the lemma. Then there exists a job with index $k \neq j$ and with properties $p_k \leq p_j$ and $p_k/w_k \leq p_i/w_i$ which is rejected. Consider the feasible solution which also accepts this job. Then the value of the objective function is $W_{R\setminus\{k\}} + T_{A\cup\{k\}}$ and by Lemma 10 we obtain that

$$W_{R\setminus\{k\}} + T_{A\cup\{k\}} \leq W_R - w_k + M_A + \frac{P_A + p_k}{M_A}.$$

On the other hand, $p_k/w_k \leq p_i/w_i \leq M_A$ thus $P_k/M_A \leq w_k$ which yields that $W_{R\setminus\{k\}} + T_{A\cup\{k\}} \leq W_R + T_A$. Therefore accepting job with index k does not increase the value of the objective function and this proves the statement of the lemma. □

By Lemma 15 we can find a polynomial time algorithm which solves the relaxed problem. (We consider the restricted problem REL(j) for each j and we investigate the possible prefixes of the ordered sequences and choose the best solution.) Furthermore by Lemma 14 we can find in each step such an optimal solution of the relaxed problems for the set J_k where the size of the maximal accepted job is increasing. Using such maximal jobs and the prefixes of the ordered sequences in each steps we have a polynomial time algorithm giving optimal solutions which satisfy Lemma 14. We call this algorithm *Relopt*. Denote the sets of the indices of the accepted jobs from J_k by A_k^* and the set of the indices of rejected jobs by R_k^*. Therefore $A_i^* \subseteq A_k^*$ if $i \leq k$. Then the following statement holds.

Lemma 16 *For the above defined sets, the following inequality is valid:*

$$\sum_{j=1}^{n} W_{R_{j-1}^* \cap A_j^*} \ \leq \ T_{A_n^*}.$$

Proof. We have $R_j^* \setminus \{j\} \subseteq R_{j-1}^*$ by $A_{j-1}^* \subseteq A_j^*$. Therefore

$$\mathrm{ropt}(J_{j-1}) = W_{R_j^*\setminus\{j\}} + W_{R_{j-1}^* \cap A_j^*} + T_{A_{j-1}^*}.$$

On the other hand, using the sets $R_j^* \setminus \{j\}$ and $A_j^* \setminus \{j\}$ we get a feasible solution of the relaxed problem on set J_{j-1}, thus

$$\mathrm{ropt}(J_{j-1}) \leq W_{R_j^* \setminus \{j\}} + T_{A_j^* \setminus \{j\}},$$

so substituting the definition of $\mathrm{ropt}(J_{j-1})$ we obtain that

$$W_{R_{j-1}^* \cap A_j^*} \leq T_{A_j^* \setminus \{j\}} - T_{A_{j-1}^*}.$$

Therefore

$$\sum_{j=1}^{n} W_{R_{j-1}^* \cap A_j^*} \leq \sum_{j=1}^{n} (T_{A_j^* \setminus \{j\}} - T_{A_{j-1}^*}).$$

On the other hand by Lemma 9 we obtain $T_{A_j^* \setminus \{j\}} \leq T_{A_j^*}$, thus

$$\sum_{j=1}^{n} W_{R_{j-1}^* \cap A_j^*} \leq \sum_{j=1}^{n} (T_{A_j^*} - T_{A_{j-1}^*}) = T_{A_n^*},$$

and this is what we have to prove. □

Now we are ready to define the class $\mathrm{Optcopy}_\rho$ of algorithms. Each algorithm form this class rejects all the jobs rejected by Relopt, therefore it does not accept more jobs than the optimal solution of the relaxed problem. On the other hand, it may reject more jobs than an optimal solution, but we can prove some bounds on the amount of the accepted jobs.

Algorithm $\mathcal{OC}_\rho$

At the arrival of a new job perform the following steps.

(i) If it is rejected by Relopt, reject it, otherwise go to step (ii)

(ii) Schedule the job by Algorithm $\mathcal{A}_\rho$, where in the machine purchasing rule only the accepted jobs are taken into account.

We have the following result.

Theorem 17 *Algorithm $\mathcal{OC}_\rho$ with the sequence $\rho = (0, 4, 9, 16, \ldots, i^2, \ldots)$ is $(\varphi + 1)$-competitive.*

Proof. Denote A_n the set of the indices of jobs scheduled by Algorithm $\mathcal{OC}_\rho$ and A_n^* the set of indices of jobs accepted by Relopt. Since $A_n \subseteq A_n^*$ and because of Lemma 12

$$\mathcal{OC}_\rho(J) \;=\; W_{R_n} + \mathcal{A}_\rho(A_n) \;\leq\; W_{R_n} + \varphi T_{A_n^*}, \tag{2.5}$$

furthermore by the definition of the algorithms Optcopy and Relopt we obtain that

$$R_n \;=\; \bigcup_{j=1}^{n} R_j^* \;=\; \bigcup_{j=1}^{n-1} \left(R_j^* \setminus R_{j+1}^*\right) \cup R_n^* \;=\; \bigcup_{j=1}^{n-1} \left(R_j^* \cap A_{j+1}^*\right) \cup R_n^*,$$

so applying Lemma 16

$$W_{R_n} \;=\; W_{R_n^*} + \sum_{j=1}^{n-1} W_{R_j^* \cap A_{j+1}^*} \;\leq\; W_{R_n^*} + T_{A_n^*}. \tag{2.6}$$

Finally applying inequalities (2.5) and (2.6), we get

$$\mathcal{OC}_\rho(J) \;\leq\; W_{R_n^*} + (1+\varphi)\, T_{A_n^*} \leq (1+\varphi)\mathrm{opt}(J)$$

and this is exactly what we have to prove. □

We note that we could not determine the competitive ratio of the algorithm: we just proved an upper bound on it. On the other hand, it is easy to see that the competitive ratio of the algorithm is at least $\frac{2+2\varphi}{\varphi+1/\varphi} \approx 2,34$. Consider the following sequence of jobs: the first job is $(\varphi N, \varphi N)$, and then N^3 jobs of size $(1/N, \infty)$ followed by one job of size $(\varphi N, \infty)$ follows. (The second part of the example is the same which was used in [25].) Then Algorithm $\mathcal{OC}_\rho$ will reject the first job and accept the others, it will schedule the first N^3 by purchasing N machines and putting N^2 jobs on each machine. The final job will be placed on an arbitrary machine. Therefore, the cost of Algorithm $\mathcal{OC}_\rho$ will be $N + N + 2\varphi N$. The optimal cost is no more than $\varphi N + \lceil (N+2\varphi)/\varphi \rceil$. So, the competitive ratio of Algorithm $\mathcal{OC}_\rho$ is at least

$$\frac{(2+2\varphi)N}{\varphi N + \lceil (N+2\varphi)/\varphi \rceil} \quad \xrightarrow{N\to\infty} \quad \frac{2+2\varphi}{\varphi + 1/\varphi}.$$

Chapter 3

Coloring graphs and hypergraphs

3.1 Preliminaries

In this chapter on hypergraph we mean a structure $H = (V, E)$ where V is the finite set of the vertices of the hypergraph and E is a subset of the nonempty subsets of V called the set of the edges. If each edge has at most two elements than H is a graph (in this case we will write G instead of H). We suppose that each edge has at least two elements.

We can define coloring of a hypergraph many ways (see in [10]). Here we consider the one which is an assignment of positive integers (called colors) to the vertices of the hypergraph so that every edge contains vertices having different colors. For a hypergraph H the minimum number of colors which is enough to color the hypergraph is called the *chromatic number* of the hypergraph and is denoted by $\chi(H)$.

The notion of online graph appeared in [29], followed by that of online hypergraph which is the generalization of online graph defined first in [2]. An *online hypergraph* is a structure $H^{\prec} = (H, \prec)$ where $H = (V, E)$ is a hypergraph and $\prec$ is a linear order of its vertices. We call a vertex the first, second, ..., and ending vertex of an edge according to the ordering $\prec$. Let H_i denote the online hypergraph induced by the $\prec$-first i elements V_i of V.

An online hypergraph coloring algorithm colors the i-th vertex of the hypergraph by only looking at the subhypergraph H_i. For an online algorithm $\mathcal{A}$ and an online hypergraph $H^{\prec}$, the cost is the number of colors used by $\mathcal{A}$

to color $H^{\prec}$ which is denoted by $\chi_{\mathcal{A}}(H^{\prec})$. Clearly, $\mathrm{opt}(H^{\prec}) = \chi(H)$. For a hypergraph H, $\chi_{\mathcal{A}}(H)$ denotes the maximum of the $\chi_{\mathcal{A}}(H^{\prec})$ values over all orderings $\prec$. Clearly, the competitive ratio of an algorithm $\mathcal{A}$ on a class Γ of hypergraphs is $\sup_{H\in\Gamma} \chi_{\mathcal{A}}(H)/\chi(H)$.

Online graph coloring has been investigated in several papers, one can find many details on the problem in the survey paper [26]. Several results are proved about the following straightforward online graph coloring algorithm First Fit ($\mathcal{FF}$).

Algorithm $\mathcal{FF}$

When a vertex arrived assign to it the least color which does not make a monochromatic edge.

In [24] it is shown that this algorithm is the best possible for the trees, where the authors constructed a tree T_k on 2^{k-1} vertices such that for every online algorithm $\mathcal{A}$, $\chi_{\mathcal{A}}(T_k) \geq k$. But more general Algorithm $\mathcal{FF}$ is not effective: for every positive integer there exists a 2-colorable online graph $G^{\prec}$ on $2n$ vertices such that $\chi_{\mathcal{FF}}(G^{\prec}) = n$. To verify this consider $G = K_{n,n} - \{a_ib_i : i \in [n]\}$, where $\{a_ib_i : i \in [n]\}$ is a perfect matching in $K_{n,n}$. Let $\prec$ be the input sequence $a_1 \prec b_1 \prec a_2 \prec b_2 \prec \ldots, \prec a_n \prec b_n$. Then Algorithm $\mathcal{FF}$ colors each a_i and b_i with color i.

The following online algorithm is developed by Lovász (posed as "an easy exercise" in [29], see [26] for details)

Algorithm $\mathcal{AA}$

Consider the input sequence $v_1 \prec \ldots \prec v_n$ of an online 2-colorable graph $G^{\prec}$.
When v_i is presented there is a unique partition (I_1, I_2) of the connected component of $G_i^{\prec}$ to which v_i belongs, into independent sets such that $v_i \in I_1$. Assign v_i the least color not already assigned to some vertex of I_2.

Obviously Algorithm $\mathcal{AA}$ produces a coloring.

Theorem 18 (Lovász, Saks, Trotter [29]) *For every 2-colorable graph G on n vertices $\chi_{\mathcal{AA}}(G) \leq 2\log_2 n$.*

The best known lower bound [41] states that no online algorithm can color every k-colorable graph on n vertices with less then $\Omega(\log^{k-1} n)$ colors. In [27] an online algorithm is presented which colors k-colorable graphs on n vertices with at most $O(n^{1-1/k!})$ colors. As a part of this algorithm the author introduces Algorithm $\mathcal{B}_n$ which uses less than $2n^{1/2}$ colors to color any online graph on n vertices that induces neither C_3 nor C_5. Let v be a vertex of a graph G. Denote $N(v)$ the set of the neighbors of v in G. For any $S \subset V(G)$ let us define $N(S) = \bigcup_{v \in S} N(v) \setminus S$.

Algorithm $\mathcal{B}_n$

Consider the input sequence $v_1 \prec \ldots \prec v_n$ of an online graph $G^{\prec}$ containing neither C_3 nor C_5. Initialize by setting $U_i = \emptyset$ for all $i > n^{1/2}$. At the s-th stage the algorithm processes the vertex v_s as follows.

(i) If there exists $i \in [n^{1/2}]$ such that v_s is not adjacent to any vertex colored i then color v_s by the least such i. (Coloring by Algorithm $\mathcal{FF}$ with $n^{1/2}$ colors.)

(ii) Otherwise, if there exists $i > n^{1/2}$ such that $v_s \in N(U_i)$ then color v_s by the least such i.

(iii) Otherwise, let j be the least integer $i > n^{1/2}$ with $U_i = \emptyset$. Set $U_j = \{v \in N^{\prec}(v_s) : \text{the color of } v \text{ is at most } n^{1/2}\}$ and color v_s with j.

The following lemma holds for Algorithm $\mathcal{B}_n$.

Lemma 19 (Kierstead [26, 27]) *Algorithm $\mathcal{B}_n$ produces a coloring of any graph $G^{\prec}$ on n vertices that induces neither C_3 nor C_5 with fewer than $2n^{1/2}$ colors.*

The girth of a graph G denoted $g(G)$ is the length of its shortest cycle and the oddgirth of G denoted $g_o(G)$ is the length of its shortest odd cycle. In the next two sections we will generalize Kierstead's result above for graphs with high girth and graphs with high oddgirth. In the latter case we will apply Algorithm $\mathcal{AA}$.

Concerning the online hypergraph coloring problem we are not aware of any results about this area. The only online problem for hypergraphs which has been investigated is the problem of finding independent sets. This problem has been investigated in [2], where an $\Theta(n/k)$ lower and upper bound is

presented for the competitive ratio of deterministic algorithms. Some results from online graph coloring which belong to particular subclasses can be extended easily: the algorithms and their competitive analysis for the trees, the graphs with bounded degree, k-claw free graphs, graphs without induced C_3 and C_5 (see [19] and [27] for the results on graphs). On the other hand, the results for general graphs cannot be extended. In this chapter we will show that in contrast to the online graph coloring there is no online algorithm with sublinear competitive ratio for the general online hypergraph coloring problem. Moreover the hypergraph which proves the lower bound is 2-colorable (bipartite graphs can be colored by $2\log n$ colors online).

We also investigate some particular hypergraph classes. We give the performance of Algorithm $\mathcal{FF}$ and we present matching lower bounds. We show that in the case of 2-colorable k-uniform hypergraphs no online algorithm exists which can color every such hypergraph with less than $\lceil n/(k-1)\rceil$ colors and we show that Algorithm $\mathcal{FF}$ colors these hypergraphs with this much colors. Furthermore we consider hypergraphs with matching number k, showing that Algorithm $\mathcal{FF}$ colors these hypergraphs with $2k+1$ colors. As a consequence we obtain that this algorithm colors the projective planes with 3 colors. We show that this bound is the best possible, we prove that there exists no online algorithm which can color a projective plane with less then 3 colors. (We note that the projective planes of order $q > 2$ are 2-colorable).

We use the following notions. A hypergraph is called *k-uniform* if each edge contains k vertices. The *degree* of a vertex is the number of edges containing it, the maximal degree of a hypergraph is the maximum of the degrees of the vertices. By the *matching number* $\nu(H)$ of a hypergraph H we mean the maximal number of pairwise disjoint edges of H.
Recall that a *finite projective plane* of order q is a $q+1$-uniform hypergraph $H = (V, E)$ satisfying the following properties:

1. Given any two distinct vertices, there is exactly one edge that contains both vertices.

2. The intersection of any two distinct edges contains exactly one vertex.

3. There exists a set of four vertices, no three of which belong to the same edges.

For more information on projective planes the reader is referred to [12]. The Let us note that by the definitions it follows immediately that the matching number of the finite projective planes is 1.

The distance $\mathrm{dist}(u,v)$ of vertices u and v is the length of the shortest uv path. For a positive integer d, let $N_d(v)$ be the set of vertices with positive distance at most d from vertex v:

$$N_d(v) = \{u \in V(H) : 1 \leq \mathrm{dist}(u,v) \leq d\}.$$

$N_{d,\mathrm{odd}}(v)$ is the set of vertices with a positive *odd* distance at most d from vertex v:

$$N_{d,\mathrm{odd}}(v) = \{u \in V(H) : 1 \leq \mathrm{dist}(u,v) \leq d \text{ and } \mathrm{dist}(u,v) \text{ is odd}\}.$$

For any $S \subset V(H)$ let us define

$$N_d(S) = \bigcup_{v \in S} N_d(v) \setminus S \quad \text{and} \quad N_{d,\mathrm{odd}}(S) = \bigcup_{v \in S} N_{d,\mathrm{odd}}(v) \setminus S.$$

Let $N_d^{\prec}(v)$ the set of vertices preceding v with a positive distance at most d from vertex v:

$$N_d^{\prec}(v) = \{u \in V(H^{\prec}) : u \prec v,\ 1 \leq \mathrm{dist}(u,v) \leq d\}.$$

$N_{d,\mathrm{odd}}^{\prec}(v)$ is the set of vertices preceding v with a positive odd distance at most d from vertex v:

$$N_{d,\mathrm{odd}}^{\prec}(v) = \{u \in V(H^{\prec}) : u \prec v,\ 1 \leq \mathrm{dist}(u,v) \leq d \text{ and } \mathrm{dist}(u,v) \text{ is odd}\}.$$

Note that $N_1(v) = N(v)$ and $N_1^{\prec}(v) = N^{\prec}(v)$ are just the neighbors and preceding neighbors of v, furthermore $N_1(S) = N(S)$.

We can assume that our graph coloring online algorithms know the number of vertices of graph G by the following Lemma [26].

Lemma 20 (Kierstead [26]) *Let Γ be a class of graphs and f be an integer valued function on the positive integers such that $f(x) \leq f(x+1) \leq f(x)+1$, for all x. If for every n, there exist an online coloring algorithm $\mathcal{A}_n$ such that for every graph $G \in \Gamma$ on n vertices, $\chi_{\mathcal{A}_n}(G) \leq f(n)$ then there exists a fixed online coloring algorithm $\mathcal{A}$ such that for every $G \in \Gamma$ on n vertices $\chi_{\mathcal{A}}(G) \leq 4f(n)$.*

3.2 Results on graphs with forbidden subgraphs

The results of this section can be found in [33].

3.2.1 Graphs with high girth

Now we generalize Kierstead's algorithm for graphs with high girth.

Algorithm $\mathcal{B}_{n,d}$

Consider the input sequence $v_1 \prec \ldots \prec v_n$ of an online graph $G^{\prec}$ with $g(G) > 4d+1$. Initialize by setting $U_i = \emptyset$ for all $i > dn^{1/(d+1)}$.

s-th stage.

(i) If there exists $i \in [dn^{1/(d+1)}]$ such that v_s is no adjacent to any vertex colored i then color v_s by the least such i.

(ii) Otherwise, if there exists $i > dn^{1/(d+1)}$ such that $v_s \in N_d(U_i)$ then then color v_s by the least such i.

(iii) Otherwise, let j be the least integer $i > dn^{1/(d+1)}$ with $U_i = \emptyset$. Set $U_j = N_d^{\prec}(v_s)$ and color v_s with j.

Theorem 21 *Algorithm $\mathcal{B}_{n,d}$ produces a coloring of any graph $G^{\prec}$ on n vertices with girth $g > 4d+1$ with less than $(d+1)n^{1/(d+1)}$ colors.*

Our proof goes along the same line as Kierstead's proof.

Proof. First we prove that $\mathcal{B}_{n,d}$ produces a coloring. Assume to the contrary that two adjacent vertices $x \prec y$ have the same color j. Clearly y is not colored by Step (i). Thus $j > dn^{1/(d+1)}$ and hence x is not colored by Step (i). Since only the first vertex colored j can be colored by Step (iii), y must be colored by Step (ii). If x is colored by Step (iii) then $U_j \subset N_d(x)$ and $y \in N_d(U_j)$ so there exists $z \in U_j$ that both $\mathrm{dist}(x,z) \leq d$ and $\mathrm{dist}(y,z) \leq d$ hold. But then $G^{\prec}$ contains a cycle of length at least $2d+1$, a contradiction (Figure 3.1 shows this case, the waves denote paths with length at most d).

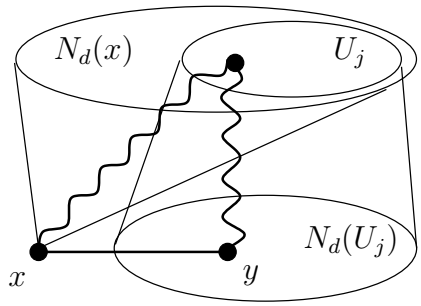


Figure 3.1: Contradiction: cycle with length at most $2d + 1$

If x is colored by Step (ii) then both x and y are in $N_d(U_j)$ so there exist (not necessarily distinct) $x', y' \in U_j$ with $\text{dist}(x', x) \leq d$, $\text{dist}(y', y) \leq d$, $\text{dist}(x', z) \leq d$ and $\text{dist}(y', z) \leq d$ where z is the first vertex colored with j. In this case $G^{\prec}$ contains a cycle with length at most $4d + 1$, a contradiction (Figure 3.2 shows this case, the waves denote paths with length at most d). So $\mathcal{B}_{n,d}$ produces a coloring.

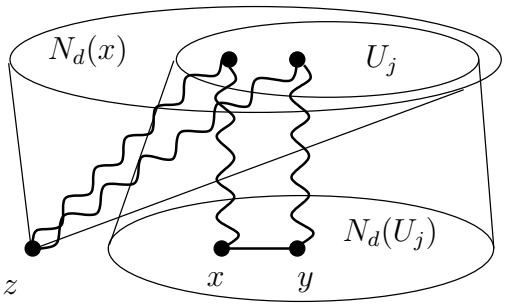


Figure 3.2: Contradiction: cycle with length at most $4d + 1$

Now we give an upper bound for the number of colors used by $\mathcal{B}_{n,d}$. At most $dn^{1/(d+1)}$ colors are used in Step (i). Let $j > dn^{1/(d+1)}$ and z_j be the first vertex colored j. From the assumption on $g(G)$ it follows that the subgraph induced by $U_j \cup \{j\}$ is a tree. Since z_j is not colored by Step (i) it has neighbors colored $1, 2, \ldots, dn^{1/(d+1)}$ in U_j and each vertex $x \in N_{d-1}(z_j)$ colored $i \leq dn^{1/(d+1)}$ have neighbors colored $1, 2, \ldots, i-1$ in U_j. Thus, for each $S \subset [dn^{1/(d+1)}]$, $|S| \leq d$ there exists $x \in U_j$ such that the colors occurring on the (unique) z_j–x path are exactly the elements of $S \cup \{j\}$. So counting

the z_j–x paths with length at most d over all possible x we get that

$$|U_j| \geq \sum_{\ell=1}^{d} \binom{dn^{1/(d+1)}}{\ell} > \binom{dn^{1/(d+1)}}{d} > n^{d/(d+1)}.$$

Since $z_j \notin N(U_i)$ if $i \neq j$ that is $U_i \cap U_j = \emptyset$, at most $n/(n^{d/(d+1)}) = n^{1/(d+1)}$ colors are used in Steps 2 and 3. Thus

$$\chi_{\mathcal{B}_{n,d}}(G^{\prec}) \leq (d+1)n^{1/(d+1)}.$$

□

If $G^{\prec}$ an online graph $G^{\prec}$ on n vertices with $g(G) > g = 4d+1$ then $\chi_{\mathcal{B}_{n,d}} \leq \frac{g+3}{4} n^{4/(g+3)}$. We note that Erdős [20] proved that for any $g > 0$ and sufficiently large n there exists a graph G on n vertices with girth greater than g and with $\chi(G) > n^{1/(2g)}$.

3.2.2 Graphs with high oddgirth

We will use Algorithm $\mathcal{AA}$ as an auxiliary algorithm and we need a slight improvement of Theorem 18.

Lemma 22 *Suppose that $G^{\prec}$ is a 2-colorable online graph and $\mathcal{AA}$ uses at least k colors on a connected component C of $G_i^{\prec}$. Let $v \in C$ be a vertex colored k by Algorithm $\mathcal{AA}$. Then both color classes of C contain at least $2^{k/2-1}$ vertices having distance at most k from v.*

Proof. We argue by induction on k and note that the base step is trivial. For the induction step observe that if $\mathcal{AA}$ assigns color $k+2$ to v_i then $\mathcal{AA}$ must have already assigned color k to some vertex $v_p \in I_2 \cap N_1(v_i)$ and color $k+1$ to some other vertex in $I_2 \cap N_1(v_i)$. Thus $\mathcal{AA}$ must have assigned color k to some vertex $v_q \in I_1 \cap N_2(v_i)$. $\mathcal{AA}$ assigned the same color to v_p and v_q, hence v_p and v_q must be in separate components of $G_t^{\prec}$ where $t = \max\{p, q\}$ (see Figure 3.3). Thus by the induction hypothesis each of the color classes of these connected components must have at least $2^{k/2-1}$ vertices with distance at most k from v_p or v_q and so the color classes of the components of v_i have at least $2^{(k+2)/2-1}$ vertices with distance at most $k+2$ from v_i. □

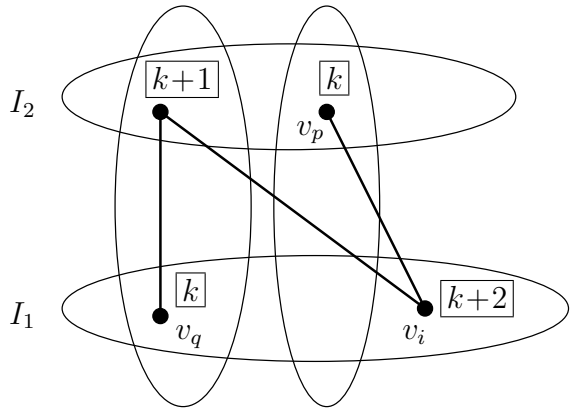


Figure 3.3: Online coloring bipartite graph

Now we generalize Algorithm $\mathcal{B}_n$ for graphs with high oddgirth.

Algorithm $\mathcal{BO}_{n,d}$

Consider the input sequence $v_1 \prec \ldots \prec v_n$ of online graph $G^{\prec}$ with $g_o(G) > 4d+1$. Set $r = (n/(2d\log_2 2d))^{1/2}$. Initialize by setting $S_i = \emptyset$ for all $i \in [r]$ and $U_i = \emptyset$ for all $i > r$. At the s-th stage the algorithm processes the vertex v_s as follows.

(i) If there exists $i \in [r]$ such that the subgraph induced by $S_i \cup \{v_s\}$ is 2-colorable and Algorithm $\mathcal{AA}$ uses at most $2\log_2 2d$ colors to color $S_i \cup \{v_s\}$, then let j be the least such i. Set $S_j = S_j \cup \{v_s\}$ and color v_s (in the subgraph induced by S_j) by Algorithm $\mathcal{AA}$ using colors $2(j-1)\log_2 2d+1, \ldots, 2j\log_2 2d$.

(ii) Otherwise, if there exists $i > r$ such that $v_s \in N_{d,\mathrm{odd}}(U_i)$ then color v_s with the least such i.

(iii) Otherwise, let j be the least integer $i > r$ such that $U_i = \emptyset$. Set $U_j = N^{\prec}_{d,\mathrm{odd}}(v_s) \cap (\bigcup_{\ell=1}^{r} S_\ell)$ and color v_s with j.

Theorem 23 *Algorithm $\mathcal{BO}_{n,d}$ produces a coloring of any graph $G^{\prec}$ on n vertices having oddgirth greater than $4d+1$ with at most $2(2n\log_2 2d/d)^{1/2}$ colors.*

Proof. First we prove that $\mathcal{BO}_{n,d}$ produces a coloring. Assume to the contrary that two adjacent vertices $x \prec y$ have the same color j. Clearly y is not colored

by Step (i). Thus $j > r$ and hence x is not colored by Step (i). Since only the first vertex colored j can be colored by Step (iii), y must be colored by Step (ii). If x is colored by Step (iii) then $U_j \subset N_{d,\text{odd}}(x)$ and $y \in N_{d,\text{odd}}(U_j)$ so there exists $z \in U_j$ that both $\text{dist}(x,z) \leq d$ and $\text{dist}(y,z) \leq d$ are odd. But then $G^{\prec}$ contains a closed walk with odd length at most $2d+1$ so it contains an odd cycle with length at most $2d+1$, a contradiction (Figure 3.4 shows this case, the waves denote paths with odd length at most d).

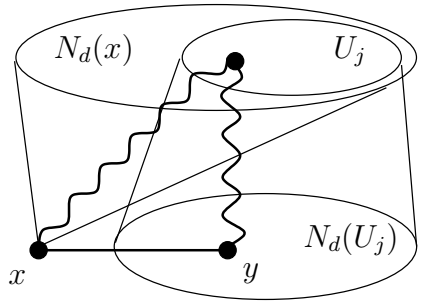


Figure 3.4: Contradiction: odd cycle with length at most $2d+1$

If x is colored by Step (ii) then both x and y are in $N_{d,\text{odd}}(U_j)$ so there exist (not necessarily distinct) $x', y' \in U_j$ with odd distances $\text{dist}(x',x) \leq d$, $\text{dist}(y',y) \leq d$, $\text{dist}(x',z) \leq d$ and $\text{dist}(y',z) \leq d$ where z is the first vertex colored with j. In this case $G^{\prec}$ contains a closed walk with odd length at most $4d+1$, a contradiction (Figure 3.5 shows this case, the waves denote paths with odd length at most d). So $\mathcal{BO}_{n,d}$ produces a coloring.

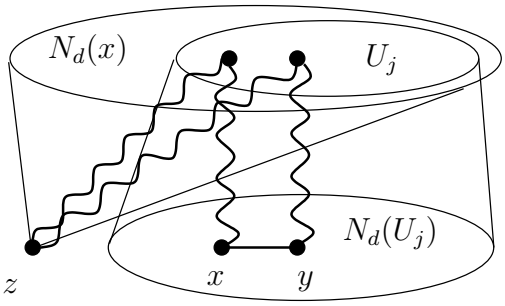


Figure 3.5: Contradiction: odd cycle with length at most $4d+1$

Now we give an upper bound for the number of colors used by $\mathcal{BO}_{n,d}$. At most $2r \log_2 2d$ colors are used in Step (i). Let $j > r$ and z_j be the first vertex colored j. Now we show that $|U_j \cap S_k| \geq d$ for all $k \leq r$. Since z_j is not

colored by Step (i), we have two cases. In the first case the subgraph induced by $S_k \cup \{z_j\}$ contains an odd cycle containing z_j with length at least $4d+1$ so $|U_j \cap S_k| \geq d$ by the definition of U_j. In the second case the subgraph induced by $S_k \cup \{z_j\}$ has is 2-colorable but Algorithm $\mathcal{AA}$ uses at least $2\log_2 2d + 1$ colors to color this subgraph. From the proof of Lemma 22 it follows that $|U_j \cap S_k| \geq 2^{(2\log_2 2d)/2-1} = d$. Since $S_k \cap S_\ell = \emptyset$ if $k \neq \ell$, we get that $|U_j| \geq rd$. Since $U_i \cap U_j = \emptyset$ if $i \neq j$, at most $n/(rd)$ colors are used in Steps 2 and 3. Thus

$$\begin{aligned}
\chi_{GB}(G^{\prec}) &\leq 2r\log_2 2d + \frac{n}{rd} \\
&= 2\left(\frac{n}{2d\log_2 2d}\right)^{1/2}\log_2 2d + \frac{n}{d\left(\frac{n}{2d\log_2 2d}\right)^{1/2}} \\
&= 2\left(\frac{2n\log_2 2d}{d}\right)^{1/2}.
\end{aligned}$$

□

We note that running $\mathcal{BO}_{n,d}$ on G with oddgirth $4d+1$ choosing $r = (n/(2d))^{1/2}$ and exploiting that the subgraphs induced by S_j ($j \in [r]$) are 2-colorable we get that $\chi(G) \leq 2r + \frac{n}{rd} = 2\left(\frac{2n}{d}\right)^{1/2}$.

3.3 Results on hypergraphs

The results of this section can be found in [35].

3.3.1 2-colorable k-uniform hypergraphs

In this part we consider the case of 2-colorable k-uniform hypergraphs with $k \geq 3$. We prove the following result.

Theorem 24 *Let $k \geq 3$. For every online hypergraph coloring algorithm $\mathcal{A}$ there exists a 2-colorable k-uniform hypergraph H on n vertices with $\chi_{\mathcal{A}}(H) \geq \lceil n/(k-1)\rceil$. If H is a k-uniform hypergraph then $\chi_{\mathcal{FF}}(H) \leq \lceil n/(k-1)\rceil$.*

Proof. Let $\mathcal{A}$ be an arbitrary online hypergraph coloring algorithm, define the online hypergraph $H_{n,\mathcal{A}}$ on n vertices as follows. Any vertex v_i is the ending vertex of the following edges: for each color c which is used by $\mathcal{A}$

for at least $k-1$ vertices we have and edge $e_{c,i}$ which contains the vertices colored with c by $\mathcal{A}$ and v_i.

Note that if $k-1$ vertices are colored with a color c by $\mathcal{A}$, then each of the following vertices is contained in an edge together with these c-colored vertices. Therefore no more vertex can get the color c. This means that each color is used at most $k-1$ times. Consequently we obtained that $\chi_{\mathcal{A}}(H_{n,\mathcal{A}}) \geq \lceil n/(k-1) \rceil$.

Now we show that this hypergraph is k-uniform and 2-colorable. Let us observe that each edge in $H_{n,\mathcal{A}}$ contains k vertices, where the first $k-1$ are colored with the same color by $\mathcal{A}$. Consider the following 2-coloring of $H_{n,\mathcal{A}}$. Every vertex which is colored with a new color by $\mathcal{A}$ gets the color 1, the other vertices get the color 2. By the above observation on the first $k-1$ vertices of the edges it follows that the first vertex gets the color 1, the second vertex gets the color 2 (since $k \geq 3$ it has the same online color) which yields that there is no monochromatic edge in this coloring. Therefore we showed that the hypergraph is 2-colorable.

Now consider Algorithm $\mathcal{FF}$. If less than $k-1$ vertices are colored by Algorithm $\mathcal{FF}$ with some color c then Algorithm $\mathcal{FF}$ does not use a further color for the next vertex because the graph is k-uniform. Therefore the number of colors used by Algorithm $\mathcal{FF}$ is not more than $\lceil n/(k-1) \rceil$. □

The theorem shows that the competitive ratio of Algorithm $\mathcal{FF}$ is $\lceil n/(k-1) \rceil/2$ on this class and no better algorithm can be defined for this class. Therefore we obtained the following result.

Corollary 25 *Algorithm $\mathcal{FF}$ is an optimal online algorithm for the class of 2-colorable k-uniform hypergraphs.*

We must note that in this case not only Algorithm $\mathcal{FF}$ but many reasonable coloring algorithms can color the above hypergraph with $\lceil n/(k-1) \rceil$ colors. Any algorithm which uses each color at most $k-1$ times gives a coloring of the hypergraph.

Moreover this theorem also proves (with $k=3$) that contrary to the case of the online graph coloring in the case of hypergraphs no online algorithm with sublinear competitive ratio exists.

Corollary 26 *For every online hypergraph coloring algorithm $\mathcal{A}$ there exists a 2-colorable hypergraph H on n vertices with $\chi_{\mathcal{A}}(H) \geq n/2$.*

3.3.2 Hypergraphs with maximal degree k

In this part we consider the case of 2-colorable hypergraphs with maximal degree k. It is easy to see that any hypergraph with maximal degree k is $k+1$ colorable, since Algorithm $\mathcal{FF}$ colors them with at most $k+1$ colors.

Extending the result of [24] on trees, for each online algorithm $\mathcal{A}$ a d-uniform online hypergraph can be constructed on d^k vertices with maximal degree k such that $\mathcal{A}$ uses at least $k+1$ colors to color it in the following way: we can build hypergraph T recursively from disjoint online hypergraphs $T_1, \ldots, T_{(d-1)k}$ in this order and a new vertex v. We can achieve that $\mathcal{A}$ uses $\lceil j/(d-1) \rceil$ colors to color hypergraph T_j on $d^{\lceil j/(d-1) \rceil}$ vertices by induction. It is easy to see that we can choose a vertex set V such that each hypergraph T_j contains exactly one vertex in V and for each color i there are at most $d-1$ vertices in V colored i by $\mathcal{A}$. Partitioning V into classes $V_1, \ldots, V_k$ with sizes $d-1$ such that if each same-colored $d-1$-tuple in V assigned to same class and adding new edges $V_j \cup \{v\}$ we get the required hypergraph on $(d-1)(1+d+d^2+\ldots+d^{k-1})+1 = d^k$ vertices, since if there are at most k colors used by $\mathcal{A}$ coloring $T_1, \ldots, T_{(d-1)k}$ then $V_1, \ldots, V_k$ are monochromatic so v must get color $k+1$ by $\mathcal{A}$.

Treating the problem more carefully we can give a stronger result for 2-colorable hypergraphs.

Theorem 27 *For every online hypergraph coloring algorithm $\mathcal{A}$ and integer $d > 2$ there exists a 2-colorable d-uniform hypergraph H on at most $\frac{(d-1)^k-1}{d-2}$ vertices with maximal degree k such that $\chi_{\mathcal{A}}(H) \geq k+1$.*

Proof. For an arbitrary online algorithm $\mathcal{ONL}$ we use the following recursive algorithm to construct the hypergraph H. H_0 is a hypergraph which contains one vertex and no edge. Then H_0 is colored with one color by any online algorithm, and the maximal degree of H_0 is 0.

Suppose that for every online algorithm $\mathcal{A}$ we have a d-uniform hypergraph $H_k(\mathcal{A})$ on $\frac{(d-1)^k-1}{d-2}$ vertices having maximal degree at most k and for which $\mathcal{A}$ uses at least $k+1$ colors. Let $\mathcal{ONL}_1 = \mathcal{ONL}$ and let $\mathcal{ONL}_2$ be the online algorithm which colors the vertices in the same way as $\mathcal{ONL}_1$ colors them after coloring a disjoint copy of $H_k(\mathcal{ONL}_1)$. In general, let $\mathcal{ONL}_j$ be the online algorithm which colors the vertices in the same way as $\mathcal{ONL}_1$ colors them after coloring disjoint copies of $H_k(\mathcal{ONL}_1) \ldots, H_k(\mathcal{ONL}_{j-1})$. We build $H_{k+1}(\mathcal{ONL})$ as follows. We give the disjoint d-uniform online hypergraphs $H_k(\mathcal{ONL}_1), \ldots, H_k(\mathcal{ONL}_{d-1})$ in this order to the algorithm. We

distinguish the following two cases.
If $\mathcal{ONL}$ does not use the same $k+1$ colors for hypergraphs $H_k(\mathcal{ONL}_1), \ldots$ and $H_k(\mathcal{ONL}_{d-1})$ then the sequence is finished, the hypergraph which we obtained can be chosen to $H_{k+1}(\mathcal{ONL})$: $\mathcal{ONL}$ uses at least $k+2$ colors for it, and the maximal degree is $k \leq k+1$. The number of vertices of $H_{k+1}(\mathcal{ONL})$ is at most

$$(d-1)\frac{(d-1)^k - 1}{d-2} < \frac{(d-1)^{k+1} - 1}{d-2}.$$

Otherwise, $\mathcal{ONL}$ uses the same colors for $H_k(\mathcal{ONL}_1), \ldots$ and $H_k(\mathcal{ONL}_{d-1})$ then we obtain H_{k+1} as follows. We give an extra vertex v at the end of the algorithm which closes the following vertices. For each color i we define one edge containing the first vertex from each $H_k(\mathcal{ONL}_j)$ with color i, and vertex v. Then $\mathcal{ONL}$ has to use a new color for vertex v, thus it uses $k+2$ colors. Concerning the degrees of the vertices, v has degree $k+1$, the degree of the other vertices is increased by at most 1, therefore the maximum degree of $H_{k+1}(\mathcal{ONL})$ is at most $k+1$. The number of vertices of $H_{k+1}(\mathcal{ONL})$ is at most

$$(d-1)\frac{(d-1)^k - 1}{d-2} + 1 = \frac{(d-1)^{k+1} - 1}{d-2}.$$

To prove the theorem we have to show that $H_k(\mathcal{ONL})$ is 2-colorable for each k and for each online algorithm $\mathcal{ONL}$. We prove by induction that for each k and for each online algorithm $\mathcal{ONL}$ there exists such a 2-coloring of $H_k(\mathcal{ONL})$ in which each vertex obtaining a new color from algorithm $\mathcal{ONL}$ has color 1. For H_0 the statement is trivial. Now suppose that the statement holds for k, we prove it for $k+1$. Let $\mathcal{ONL}$ be an arbitrary online algorithm. If $H_{k+1}(\mathcal{ONL})$ consists of the $d-1$ disjoint hypergraphs $H_k(\mathcal{ONL}_1), \ldots, H_k(\mathcal{ONL}_{d-1})$ then the statement is trivial, we can use the 2-colorings of $H_k(\mathcal{ONL}_1), \ldots, H_k(\mathcal{ONL}_{d-1})$, and the statement follows. Now suppose that $H_{k+1}(\mathcal{ONL})$ is given by the $d-1$ hypergraphs and the extra vertex v. Then we can give the following coloring. Color $H_k(\mathcal{ONL}_1)$ by induction, color $H_k(\mathcal{ONL}_2), \ldots, H_k(\mathcal{ONL}_{d-1})$ also by induction but swapping the colors, finally give color 1 to vertex v. By the induction hypothesis it follows that the edges which are included in one of $H_k(\mathcal{ONL}_1), \ldots, H_k(\mathcal{ONL}_{d-1})$ are not monochromatic. The edges which intersect each of $H_k(\mathcal{ONL}_1), \ldots, H_k(\mathcal{ONL}_{d-1})$ contain the first occurrence of the online color i in $H_k(\mathcal{ONL}_1)$ (this gets color 1 in the 2-coloring), the first occurrence of the online color i in $H_k(\mathcal{ONL}_2), \ldots, H_k(\mathcal{ONL}_{d-1})$ (these get color 2 in the 2-coloring), thus these edges are also not monochromatic (Figure 3.6 shows this case).

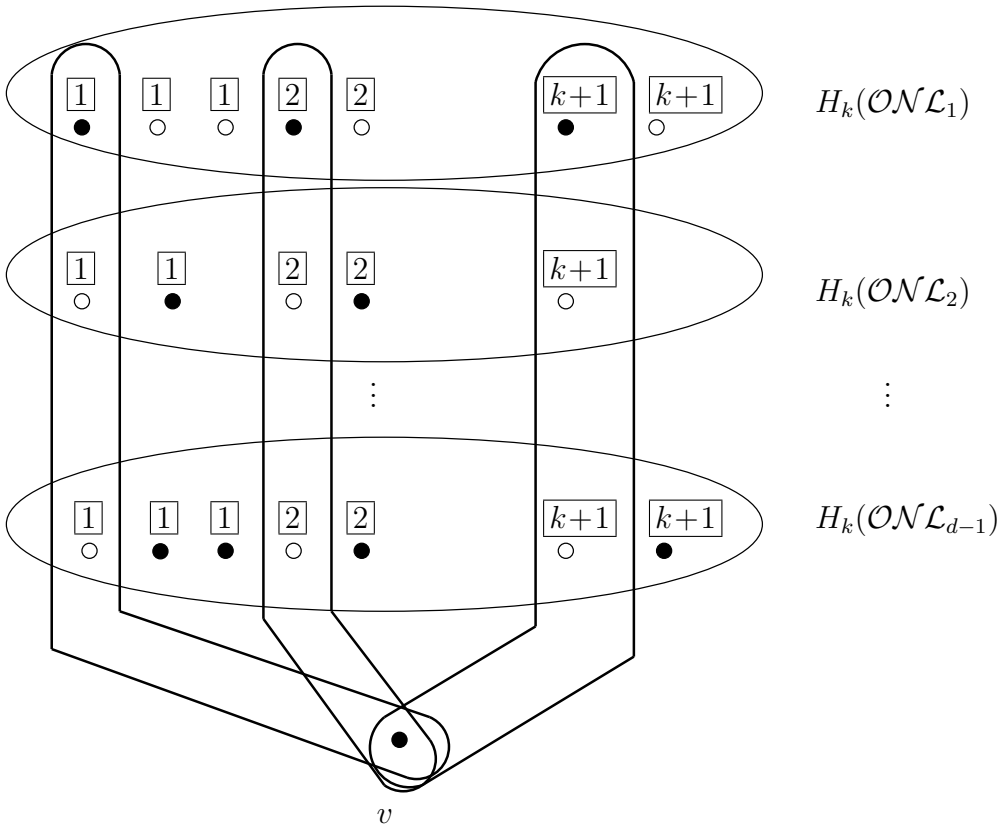


Figure 3.6: 2-coloring of $H_{k+1}(\mathcal{ONL})$

There are no other edges, therefore we gave a 2-coloring. Furthermore the extra property of the coloring also remains valid, the first occurrences of the online colors used for $H_{k+1}(\mathcal{ONL})$ get color 1 in the 2-coloring by the induction hypothesis (observe that all these vertices are in $H_k(\mathcal{ONL}_1)$ in which we did not swap colors) and the first occurrence of the fresh color (vertex v) again obtains color 1 in the 2-coloring. □

3.3.3 Hypergraphs with bounded matching number

Considering this class of hypergraphs Algorithm $\mathcal{FF}$ can achieve the following performance.

Theorem 28 *For any hypergraph H Algorithm $\mathcal{FF}$ gives a coloring of H with at most $2 \cdot \nu(H) + 1$ colors.*

Proof. Consider an arbitrary hypergraph H and suppose that Algorithm $\mathcal{FF}$ used k colors to color it. For every $i \leq k/2$ consider a vertex v_i which gets the color $2i$. By the definition of Algorithm $\mathcal{FF}$ it follows that there exists an edge E_i whose last vertex is v_i and the other vertices get the color $2i-1$ by Algorithm $\mathcal{FF}$. Then considering the edges E_i, $i = 1, \ldots, \lfloor k/2 \rfloor$ we obtain pairwise disjoint edges, and $\nu(H) \geq \lfloor k/2 \rfloor$ follows, which proves the theorem. □

Since any two edges of the finite projective planes are intersecting (the matching number is 1) we also obtained the following result.

Corollary 29 *Algorithm $\mathcal{FF}$ colors the finite projective planes with at most 3 colors.*

It is easy to see that the finite projective planes with order greater than 2 are two colorable. (Consider three points which are not collinear. Color these points with color 1, the points of the lines which connect these points get the color 2, the other points of the plane gets color 1.) On the other hand as the following statement shows there exists no online algorithm which can use less colors than Algorithm $\mathcal{FF}$ in this cases.

Theorem 30 *No online algorithm exists which can color a finite projective plane with less than 3 colors.*

Proof. Consider an arbitrary online algorithm $\mathcal{A}$ and a finite projective plane with order q. Give the points of the finite projective plane to the algorithm in the following order. The first q^2 points arrive such that none of the lines are completed (the remaining $q+1$ points will give one line of the plane furthermore they will finish all the other lines). Now distinguish the following two cases.

If $\mathcal{A}$ uses more than two colors then the statement is obvious. Suppose that $\mathcal{A}$ uses at most two colors 1 and 2. Without loss of generality we can suppose that the number of points colored by 1 is not less than the number of points colored by 2.

First suppose that at least q points is colored by 2. Then the next point which arrives will finish a line which contains q points of color 1, and a line which contains q points of color 2. Since $\mathcal{A}$ cannot construct a monochromatic line it has to use a new color for this point and the statement of the theorem follows.

Now suppose that there are at most $q-1$ points of color 2. Then each of the $q+1$ uncolored points has a line through it with q points of color 1. So now on we cannot use color 1. Since the uncolored points are on one line, it is impossible to use only color 2. This completes the proof. □

Chapter 4

The k-server problem

4.1 Preliminaries

In the theory of designing efficient virtual memory-management algorithms, the well studied paging problem plays a central role. Even the earliest operation systems contained some heuristics to minimize the amount of copying memory pages, which is an expensive operation. A generalization of the paging problem, called the k-server problem was introduced by Manasse, McGeoch and Sleator in [31], where the first important results were also achieved.

To give the definition of the general problem we need the notion of metric spaces. A *metric space* is a pair $\mathcal{M} = (M, \text{dist})$ where M is a set of points and $\text{dist} : M \times M \to \mathbb{R}$ is a metric distance function with the following properties:

- $\text{dist}(x, y) \geq 0$ for all $x, y \in M$,
- $\text{dist}(x, y) = \text{dist}(y, x)$ for all $x, y \in M$,
- $\text{dist}(x, y) + \text{dist}(y, z) \geq \text{dist}(x, z)$ for all $x, y, z \in M$,
- $\text{dist}(x, y) = 0$ holds if and only if $x = y$.

The *diameter* of $\mathcal{M}$ is $\max_{x,y \in M}\{\text{dist}(x, y)\}$.

The k-server problem can be formulated as follows. Given a metric space with k mobile servers that occupy distinct points of the space and a sequence of requests (points), each of the requests has to be served, by moving a server

from its current position to the requested point. The goal is to minimize the total cost, that is the sum of the distances covered by the k servers; the optimal cost for a given sequence ϱ is denoted $\mathrm{opt}(k, \varrho)$.

A k-server algorithm is online if it serves each request immediately when it arrives (without any prior knowledge about the future requests). The k-server conjecture (see [31]) states that there exists an algorithm that is k-competitive for any metric space. Manasse et al. proved that k is a lower bound [31], and Koutsoupias and Papadimitriou showed $2k - 1$ is an upper bound for any metric space [28].

In the randomized version there are more problems that are still open. The randomized k-server conjecture states that there exists a randomized algorithm with a competitive ratio $\Theta(\log k)$ in any metric space. The best known lower bound is $\Omega(\log k / \log\log k)$ which follows from the results of [8] (see also [5]). A natural upper bound is the bound $2k + 1$ given for the deterministic case.

By restricting our attention to metric spaces with a special structure, better bounds can be achieved: for *uniform* metric spaces where the distance is 1 between any two points, Fiat et al. [21] proved a lower bound $\mathrm{H}_k \approx \log k$. McGeoch and Sleator [30] constructed an algorithm called Partition, later Achlioptas et al. [1] presented another algorithm called Equitable, both being H_k-competitive. Although these algorithms are the best possible for the uniform spaces, yet we review the less effective algorithm Marking developed by Fiat et al. in [21] since our algorithms will be its extension in some sense. The algorithm maintains a set of *marked vertices*.

Algorithm $\mathcal{MARK}$

Initially the marked vertices are exactly those that are covered by servers. After each request, the marks are updated, followed by a server movement if necessary, as follows:

(i) *Marking:* Each time a vertex is requested, that vertex is marked. The moment $k + 1$ vertices are marked, all the marks except the one on the most recently requested vertex are erased.

(ii) *Serving:* If the requested vertex is already covered by a server, then no servers shall move. If the requested vertex is not covered, then a server is chosen uniformly at random from among the unmarked vertices, and this server is moved to cover the requested vertex.

Theorem 31 (Fiat et al. [21]) *Algorithm $\mathcal{MARK}$ is $2\mathrm{H}_k$-competitive on uniform spaces.*

In the next section we also consider a restriction of the problem, namely we seek for an efficient randomized online algorithm for metric spaces that are "μ-HST spaces" [5] and defined as follows:

Definition 32 *For $\mu \geq 1$, a μ-hierarchically well-separated tree (μ-HST) is a metric space defined on the leaves of a rooted tree T. To each vertex $u \in T$ there is associated a label $\Lambda(u) \geq 0$ such that $\Lambda(u) = 0$ if and only if u is a leaf of T. The labels are such that if a vertex u is a child of a vertex v then $\Lambda(u) \leq \Lambda(v)/\mu$. The distance between two leaves $x, y \in T$ is defined as $\Lambda(\mathrm{lca}(x, y))$, where $\mathrm{lca}(x, y)$ is the least common ancestor of x and y in T.*

The μ-HST spaces play an important role in the *probabilistic embedding technique* developed by Alon et al. [3] and Bartal [4]. Fakcharoenphol et al [22] proved that every metric space on n points can be β-probabilistically approximated by a set of μ-HSTs, for an arbitrary $\mu > 1$ where $\beta = O(\mu \log n / \log \mu)$.

In [39], μ-decomposable spaces have been introduced. We consider a special case of this notion as follows:

Definition 33 *Let $\mathcal{M}$ be a metric space. We call $\mathcal{M}$* uniformly μ-decomposable *for some $\mu > 1$ if its points can be partitioned into $t \geq 2$* blocks*, $B_1, \ldots, B_t$ such that the following conditions both hold:*

1. *whenever $x, y \in \mathcal{M}$ are belonging to different blocks, their distance is exactly Δ, the diameter of $\mathcal{M}$;*
2. *the diameter of each B_i is at most Δ/μ.*

For example, a μ-HST with at least two points is a uniformly μ-decomposable metric space.

Seiden [39] proved the existence of an $O(\mathrm{polylog}\, k)$-competitive algorithm for $\Omega(k \log k)$-decomposable spaces, where the space can be partitioned into $O(\log k)$ uniform blocks, each having diameter 1, and where the distance of any two blocks is at least $c{\cdot}k{\cdot}\log k$. In his work he also showed that for binary HST's (where each non-leaf node has exactly two children) there exists an $O(\log^3 k)$-competitive algorithm, provided the parameter μ of the HST is sufficiently large. We study decomposable spaces too, but unlike the above

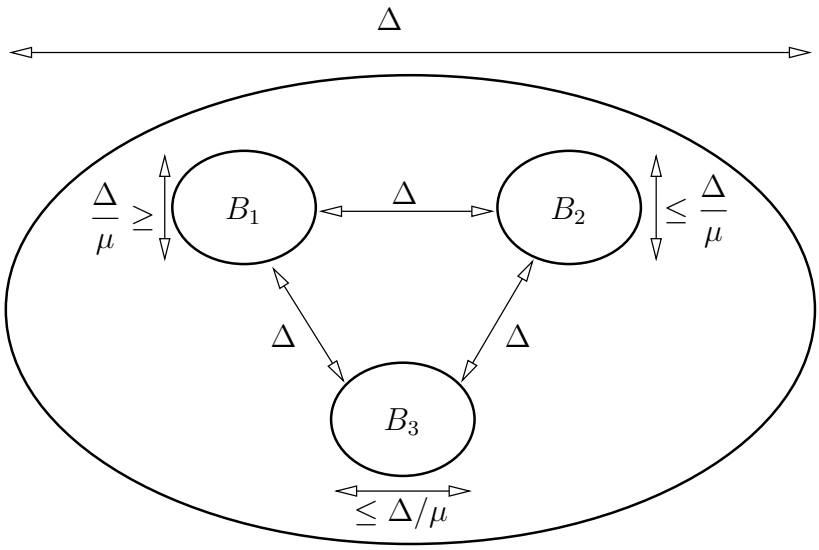


Figure 4.1: Uniformly μ-decomposable space with blocks B_1, B_2 and B_3. (Source: [39])

result our spaces consist of an arbitrary number of (not necessarily uniform) blocks with large distance between them. By slightly modifying the approach of Csaba and Lodha [13] and Bartal and Mendel [7][1] we show that there exists a polylog k-competitive algorithm for any μ-HST that has a small depth and *arbitrary* maximum degree t, given $\mu \geq k$. Our algorithm heavily relies on the technical notion of demand (Definition 34), which plays a central role in the description and the analysis of the algorithm.

For the rest of the section we fix a uniformly μ-decomposable metric space $\mathcal{M}$ having a diameter Δ, consisting of the blocks $B_1, \ldots, B_t$, with maximal diameter δ such that $\mu = \Delta/\delta$.

For a given request sequence ϱ we denote its ith member by ϱ_i, and the prefix of ϱ of length i by $\varrho_{\leq i}$. The length of the sequence is denoted $|\varrho|$.

The set of points where the servers are staying at a given time is a *configuration*. Given a block B_s, a request sequence ϱ and an initial configuration C in B_s, let $\mathcal{A}_s(C, \varrho)$ denote the cost computed by the algorithm $\mathcal{A}$ for the subsequence of ϱ consisting of the requests arriving to B_s. For any number ℓ of servers, let $\mathcal{A}_s(\ell, \varrho)$ stand for $\max_{|C|=\ell} \mathcal{A}_s(C, \varrho)$, where C runs over all the initial configurations in B_s consisting of ℓ servers. Also, let $\mathrm{opt}_s(C, \varrho)$ denote

[1] Although the publication has been withdrawn (see `http://arxiv.org/abs/cs.DS/0406033`), the approach itself is still valuable.

the optimal cost for the subsequence of ϱ consisting of the requests arriving to B_s, starting from configuration C and let

$$\mathrm{opt}_s(\ell, \varrho) = \min_{|C|=\ell} \mathrm{opt}_s(C, \varrho).$$

Thus, if ϱ is nonempty, $\mathrm{opt}_s(0, \varrho)$ is defined to be infinite.

Rejection in online problems first appeared in online scheduling [6], later in online bin packing [17], coloring graphs [19] and other problems. Hence it is a natural idea to investigate generalized model of k-server problem in which to each request a penalty is assigned which the algorithm has to pay if reject it. We give some results in this area on uniform spaces at the end of this chapter.

4.2 A randomized algorithm on decomposable spaces

The results of this section can be found in [32].

Our algorithm is based on the following notion.

Definition 34 *The* demand *of the block B_s for the request sequence ϱ is*

$$D_s(\varrho) := \min\{\ell \,|\, \mathrm{opt}_s(\ell, \varrho) + \ell\Delta = \min_j\{\mathrm{opt}_s(j, \varrho) + j\Delta\}\},$$

if ϱ is nonempty, otherwise it is 0.

Intuitively, $D_s(\varrho)$ denotes the least number of servers to be moved into the initially empty block B_s to achieve the optimal cost for the sequence ϱ. Observe that $D_s(\varrho)$ is finite since it is a nonnegative integer bounded by e.g. $|\varrho|$.

We note that in [15] a model has been investigated, where one does not have a fixed number of servers but they can be bought. The expression $\min_\ell\{\mathrm{opt}_s(\ell, \varrho) + \ell\Delta\}$ can be seen as the optimal cost in a model where one has to buy the servers, for a cost of Δ each.

In the rest of the section, the notion of demand of the blocks will play a crucial role. We now state a conjecture which would simplify the ensuing calculations, if it happened to be verified; however, we did not succeed to prove or disprove it yet.

Conjecture 35 *For any block B_s, request sequence ϱ inside B_s and index $0 < i < |\varrho|$, the difference $D_s(\varrho_{\leq i+1}) - D_s(\varrho_{\leq i})$ is either 0 or 1.*

A weaker, but still open question is that whether the sequence $(D_s(\varrho_{\leq i}))_{i=1}^{|\varrho|}$ is monotone for every ϱ and B_s.

We also introduce a technical notion.

Definition 36 *Suppose $\mathcal{N}$ is a metric space, $\mathcal{A}$ is a randomized online algorithm, f is a real function and $\mu > 0$ is a real number satisfying the following conditions:*

1. *$f(\ell)/\log \ell$ is monotone non-decreasing;*
2. *for any $0 < \ell \leq \mu$ and request sequence ϱ in $\mathcal{N}$,*

$$\mathrm{E}[\mathcal{A}(\ell, \varrho)] \;\leq\; f(\ell) \cdot \mathrm{opt}(\ell, \varrho) + \frac{f(\ell) \cdot \ell \cdot \mathrm{diam}(\mathcal{N})}{\log \ell}. \tag{4.1}$$

Then we call $\mathcal{A}$ an (f, μ)-efficient algorithm on $\mathcal{N}$.

Observe that if $\mathcal{A}$ is (f, μ)-efficient on $\mathcal{N}$, then $\mathcal{A}$ is $f(k)$-competitive for the k-server problem on $\mathcal{N}$ for any $0 < k < \mu$.

Our aim is to prove the following theorem:

Theorem 37 *Suppose $\mathcal{M}$ is a uniformly μ-decomposable space and $\mathcal{A}$ is an (f, μ)-efficient algorithm on each block of $\mathcal{M}$. Then there exists an (f', μ)-efficient algorithm on $\mathcal{M}$, where $f'(x)$ is defined as $c \cdot f(x) \log x$ for some absolute constant $c > 0$.*

For the rest of the section we now fix an algorithm $\mathcal{A}$ and a real function f such that $\mathcal{A}$ is an (f, μ)-efficient algorithm on each block of (the already fixed) $\mathcal{M}$. In the next subsection we define the algorithm which will be proven to be (f', μ)-efficient on $\mathcal{M}$. In the rest of the section we suppose that $k \leq \mu$ is an arbitrary integer.

4.2.1 Algorithm Shell

The algorithm uses $\mathcal{A}$ as a subroutine and it works in *phase*s. Let $\varrho^{(p)}$ denote the sequence of the pth phase. In this phase algorithm Shell works as follows:

Algorithm $\mathcal{SH}$

Initially we mark the blocks that contain no servers.

When $\varrho_i^{(p)}$, the ith request of this phase arrives to block B_s, we compute the demand $D_s(\varrho_{\leq i}^{(p)})$ and the maximal demand

$$D_s^*(\varrho_i^{(p)}) = \max\{D_s(\varrho_{\leq j}^{(p)}) | j \leq i\}$$

for this block (note that these values do not change in the other blocks).

(i) If $D_s^*(\varrho_i^{(p)})$ is less than the number of servers in B_s at that moment, then the request is served by algorithm $\mathcal{A}$, with respect to the block B_s.

(ii) If $D_s^*(\varrho_i^{(p)})$ becomes equal to the number of servers in B_s at that moment, then the request is served by algorithm $\mathcal{A}$, with respect to the block B_s and we mark the block B_s.

(iii) If $D_s^*(\varrho_i^{(p)})$ is greater than the number of servers in B_s at that moment, we mark the block B_s and perform the following subtask until we have $D_s^*(\varrho_i^{(p)})$ servers in that block or we cannot execute the steps (this happens when all the blocks become marked):

> Choose an unmarked block $B_{s'}$ randomly uniformly, and a server from this block also randomly. We move this chosen server to the block B_s (such a move is called a *jump*), either to the requested point, or, if there is already a server occupying that point, to a randomly chosen unoccupied point of B_s. If the number of servers in $B_{s'}$ becomes $D_{s'}^*(\varrho_i^{(p)})$ via this move, we mark that block. In both B_s and $B_{s'}$ we restart algorithm $\mathcal{A}$ from the current configuration of the block.

If we cannot raise the number of servers in block B_s to $D_s^*(\varrho_i^{(p)})$ by repeating the above steps (all the blocks became marked), then phase $p+1$ is starting and the last request is belonging to this new phase.

Intuitively, Algorithm $\mathcal{SH}$ consists of the following parts: the server movements inside a block are handled by the *inner algorithm* $\mathcal{A}$, while the "jumps" from a block to another are determined by an online matching algorithm

(introduced by Csaba and Pluhár [14]), whose requests are induced by the demands. One may regard maximal demands as markings of the blocks.

For any phase p of Algorithm $\mathcal{SH}$ we can associate a *matching problem* MX. We recall from [14] that an online matching problem is defined similarly to the online k-server problem with the following two differences:

1. Each of the servers can move only once;
2. The number of the requests is at most k, the number of the servers.

The underlying metric space of MX is a finite uniform metric space that has the blocks B_s as points and a distance Δ between any two different points. Let $\hat{D}_s(p)$ denote the number of servers that are in the block B_s just at the end of phase p. Now in the associated matching problem we have $\hat{D}_s(p-1)$ servers originally occupying the point B_s. During phase p, if some value D_s^* increases, we make a number of requests in point B_s for the associated matching problem: we make the same number of requests that the value D_s^* has been increased with. Each of these requests have to be served by a server, moreover, one server can handle only one request (during the whole phase).

We also associate an *auxiliary matching algorithm*, $\mathcal{AMA}$ on this structure as follows. While there exists a server in the block B_s which have not served any request yet, let this server handle the request arriving to B_s. Otherwise, D_s^* increases at some time, causing jumps. These jumps are corresponding to requests of the associated matching problem; $\mathcal{AMA}$ satisfies these requests by the servers that are corresponding to those involved in these jumps.

For convenience we modify the request sequence ϱ in a way that does not increase the optimal cost and does not decrease the cost of any online algorithm, hence the bounds we get for this modified sequence will hold also in the general case. The modification is defined as follows: we extend the sequence by repeatedly requesting the points of the halting configuration of a (fixed) optimal solution. We do this till $\sum_{s=1}^{t} D_s^*(\varrho_{\leq i}^{(u)})$ becomes k. Observe that the optimal cost does not change via this transformation, and any online algorithm works the same way in the original part of the sequence (hence online), so the cost computed by any online algorithm is at least the original computed cost.

In the following two subsections we will give an upper bound for the cost of Algorithm $\mathcal{SH}$ and several lower bounds for the optimal cost in an arbitrary phase. Theorem 37 easily follows from these.

We remark that the number t of blocks do not appear in the statement of Theorem 37, which is not surprising, since in each phase, at most $2k$ blocks of $\mathcal{M}$ can be involved. This comes from the fact that each server jumps at most once during one phase (since if a server jumps into a block, that block has to be a marked one, thus the server is not allowed to jump out from that block during the same phase).

Upper bound

In the first step we prove an auxiliary result.

If p is not the last phase, let $\varrho^{(p)+}$ denote the request sequence we get by adding the first request of phase $p+1$ to $\varrho^{(p)}$. Now we have

$$D_s^*(\varrho^{(p)}) \leq \hat{D}_s(p) \leq D_s^*(\varrho^{(p)+}) \tag{4.2}$$

and in all blocks but at most one we have equalities there (this is the block that causes termination of the pth phase).

Denote

$$m_p := \sum_{s=1}^{t} \max\{0, \hat{D}_s(p) - \hat{D}_s(p-1)\}. \tag{4.3}$$

Since the auxiliary metric space is uniform, the optimal cost is $m_p \cdot \Delta$.

From Lemma 6 of [14] we immediately get the following:

Lemma 38 *The expected cost of $\mathcal{AMA}$ is at most $\log k \cdot m_p \Delta$.*

A lemma similar in nature to the above was also presented in [13].

Lemma 39 *The expected cost of Algorithm $\mathcal{SH}$ in the pth phase is at most*

$$f(k)\left(\sum_{s=1}^{t} \mathrm{opt}_s(D_s(\varrho^{(p)+}), \varrho^{(p)}) + \left(\sum_{s=1}^{t} D_s(\varrho^{(p)+}) - k\right)\Delta\right)$$

$$+\big(f(k)\log k + f(k) + \log k\big)m_p\Delta + \frac{f(k)}{\log k}\Delta.$$

Proof. Consider the pth phase of an execution of Algorithm $\mathcal{SH}$ on the request sequence ϱ. We fix a possible associated execution τ of $\mathcal{AMA}$ (which satisfies the request sequence induced by ϱ); let $\mathcal{E}_\tau$ denote the event that the

execution of $\mathcal{AMA}$ is this τ. We will give an upper bound to the overall expected cost of Algorithm $\mathcal{SH}$ during phase p assuming τ. After that, we get the expected cost appearing in Lemma 39 as a weighted sum.

Let B_s be a block in which some request arrives during this phase. For the sake of convenience we will omit the subscript s when it is clear from the context. While the block B_s is unmarked, only jump-outs can happen from this block (in phase p); let d^- be the number of these jump-outs. After B_s has been marked, only jump-ins happen into this block; let d^+ be the number of these jump-ins and let $d = d^- + d^+$ denote the total number of jumps involving B_s during phase p.

Also, for any $1 \le i \le d^-$ let r_i be the index of the request in $\varrho^{(p)}$ which causes the ith jump-out from B_s, and for any $1 \le i \le d^+$ let r_{d^-+i} be the index of the request which causes the ith jump-in to B_s.

Denote $\sigma_i = \varrho^{(p)}_{r_i} \dots \varrho^{(p)}_{r_{i+1}-1}$, where $\varrho^{(p)}_{r_0}$ is the first request of the phase and $\varrho^{(p)}_{r_{d+1}-1}$ is the last request of the phase. (In other words, σ_i is the ith maximal segment of $\varrho^{(p)}$ between two jumps. Figure 4.2 shows an illustration.)

It is clear that the number of servers inside B_s does not change between two jumps; for each $0 \le i \le d$, let k_i denote the number of servers inside B_s during σ_i. Finally, let $\ell_i = D_s(\varrho^{(p)}_{<r_{i+1}})$ (the demand of B_s for the sequence $\varrho^{(p)}_{<r_{i+1}}$). Observe that $\ell_i \le k$ for each i, moreover $D_s(\varrho^{(p)}_{\le r_i})$ is exactly k_i, when $i > d^-$, and is strictly less than k_i, when $i < d^-$.

jump		outs ↓		outs ↓		ins ↓		ins ↓	
	$\overbrace{\phantom{\dots \varrho^{(p)}_{r_1-1}}}^{\sigma_0}$		$\overbrace{\phantom{\varrho^{(p)}_{r_1} \dots \varrho^{(p)}_{r_2-1}}}^{\sigma_1}$		$\overbrace{\phantom{\varrho^{(p)}_{r_2} \dots \varrho^{(p)}_{r_3-1}}}^{\sigma_2}$		$\overbrace{\phantom{\varrho^{(p)}_{r_3} \dots \varrho^{(p)}_{r_4-1}}}^{\sigma_3}$		$\overbrace{\phantom{\varrho^{(p)}_{r_4} \dots}}^{\sigma_4}$
$\varrho^{(p)}_j$:	$\dots \varrho^{(p)}_{r_1-1}$		$\varrho^{(p)}_{r_1} \dots \varrho^{(p)}_{r_2-1}$		$\varrho^{(p)}_{r_2} \dots \varrho^{(p)}_{r_3-1}$		$\varrho^{(p)}_{r_3} \dots \varrho^{(p)}_{r_4-1}$		$\varrho^{(p)}_{r_4} \dots$

Figure 4.2: Partitioning of a phase. Here $d^- = d^+ = 2$.

A jump-in to the block satisfies the last request, hence there is no server movement inside the block during a jump. The expected cost of non-jump movements in this block (this is called the *inner cost*) is, applying (4.1), at most

$$\begin{aligned} \sum_{i=0}^{d} \mathrm{E}[\mathcal{A}_s(k_i, \sigma_i) | \, \mathcal{E}_\tau] &\le \sum_{i=0}^{d} \Big(f(k_i) \mathrm{opt}_s(k_i, \sigma_i) + \frac{k_i \cdot f(k_i)}{\log k_i} \delta \Big) \\ &\le f(k) \sum_{i=0}^{d} \mathrm{opt}_s(k_i, \sigma_i) + \delta \sum_{i=0}^{d} \frac{k_i \cdot f(k_i)}{\log k_i}. \quad (4.4) \end{aligned}$$

(Recall that $\mathcal{E}_\tau$ is the random event that τ is the associated run of $\mathcal{AMA}$.)

We bound the right hand side of (4.4) piecewise. In the first step we bound the inner costs till the $(d^- - 1)$th jump (which is still a jump-out):

$$\sum_{i=0}^{d^--1} \mathrm{opt}_s(k_i, \sigma_i) \leq \sum_{i=0}^{d^--1} \mathrm{opt}_s(k_{d^-}, \sigma_i) \leq \mathrm{opt}_s(k_{d^-}, \varrho^{(p)}_{\leq r_{d^-}}). \tag{4.5}$$

From the last jump-out till the last jump-in:

$$\begin{aligned}
\sum_{i=d^-}^{d-1} \mathrm{opt}_s(k_i, \sigma_i) &\leq \sum_{i=d^-}^{d-1} \mathrm{opt}_s(\ell_i, \sigma_i) \\
&= \sum_{i=d^-}^{d-1} \big(\mathrm{opt}_s(\ell_i, \sigma_i) + \mathrm{opt}_s(\ell_i, \varrho^{(p)}_{\leq r_i}) - \mathrm{opt}_s(\ell_i, \varrho^{(p)}_{\leq r_i})\big) \\
&\leq \sum_{i=d^-}^{d-1} \big(\mathrm{opt}_s(\ell_i, \varrho^{(p)}_{<r_{i+1}}) - \mathrm{opt}_s(\ell_i, \varrho^{(p)}_{\leq r_i})\big) \\
&\leq \sum_{i=d^-}^{d-1} \Big(\big(\mathrm{opt}_s(k_{i+1}, \varrho^{(p)}_{<r_{i+1}}) + (k_{i+1} - \ell_i)\Delta\big) \\
&\qquad - \big(\mathrm{opt}_s(k_i, \varrho^{(p)}_{\leq r_i}) + (k_i - \ell_i)\Delta\big)\Big) \qquad (4.6) \\
&\leq \sum_{i=d^-}^{d-1} \big(\mathrm{opt}_s(k_{i+1}, \varrho^{(p)}_{\leq r_{i+1}}) - \mathrm{opt}_s(k_i, \varrho^{(p)}_{\leq r_i}) + (k_{i+1} - k_i)\Delta\big) \\
&= \mathrm{opt}_s(k_d, \varrho^{(p)}_{\leq r_d}) - \mathrm{opt}_s(k_{d^-}, \varrho^{(p)}_{\leq r_{d^-}}) + (k_d - k_{d^-})\Delta. \qquad (4.7)
\end{aligned}$$

Inequality (4.6) follows from Definition 34, since the demand of B_s for $\varrho^{(p)}_{<r_{i+1}}$ is ℓ_i and the demand of B_s for $\varrho^{(p)}_{\leq r_i}$ is k_i.

Since $k_d \geq D_s(\varrho^{(p)})$, analogously we get

$$\begin{aligned}
\mathrm{opt}_s(k_d, \sigma_d) &\leq \mathrm{opt}_s(D_s(\varrho^{(p)}), \sigma_d) \\
&\leq \mathrm{opt}_s(D_s(\varrho^{(p)}), \varrho^{(p)}) - \mathrm{opt}_s(D_s(\varrho^{(p)}), \varrho^{(p)}_{\leq r_d}) \\
&\leq \mathrm{opt}_s(D_s(\varrho^{(p)+}), \varrho^{(p)}) + (D_s(\varrho^{(p)+}) - D_s(\varrho^{(p)}))\Delta \\
&\quad - \mathrm{opt}_s(k_d, \varrho^{(p)}_{\leq r_d}) - (k_d - D_s(\varrho^{(p)}))\Delta \qquad (4.8) \\
&= \mathrm{opt}_s(D_s(\varrho^{(p)+}), \varrho^{(p)}) - \mathrm{opt}_s(k_d, \varrho^{(p)}_{\leq r_d}) \\
&\quad + (D_s(\varrho^{(p)+}) - k_d)\Delta. \qquad (4.9)
\end{aligned}$$

Again, (4.8) follows from Definition 34, since the demand of B_s for $\varrho^{(p)}$ is $D_s(\varrho^{(p)})$ and the demand of B_s for $\varrho^{(p)+}_{\leq r_d}$ is k_d. Note that if the first request

of the $p+1$th phase arrives to block B_s, then $D_s(\varrho^{(p)}) < D_(\varrho^{(p)+})$, otherwise the two demands are equal.

Summing up the right hand sides of (4.5), (4.7) and (4.9) we get

$$\sum_{i=0}^{d} \mathrm{opt}_s(k_i, \sigma_i) \leq \mathrm{opt}_s(D_s(\varrho^{(p)+}), \varrho^{(p)}) + (D_s(\varrho^{(p)+}) - k_{d^-})\Delta, \tag{4.10}$$

and substituting this to the right hand side of (4.4) we get that the expected inner cost in B_s is at most

$$f(k)\left(\mathrm{opt}_s(D_s(\varrho^{(p)+}), \varrho^{(p)}) + (D_s(\varrho^{(p)+}) - k_{d^-})\Delta\right) + \sum_{i=0}^{d} \frac{k_i \cdot f(k_i)}{\log k_i}\delta. \tag{4.11}$$

On the other hand,

$$D_s(\varrho^{(p)+}) - k_{d^-} = (D_S(\varrho^{(p)+}) - \hat{D}_s(p)) + (\hat{D}_s(p) - k_{d^-}), \tag{4.12}$$

where we know that $(\hat{D}_s(p) - k_{d^-})$ is the number of jump-ins into this block.

From Definition 36,

$$\sum_{i=0}^{d} \frac{k_i \cdot f(k_i)}{\log k_i}\delta \leq \frac{f(k)}{\log k}\delta \sum_{i=0}^{d} k_i$$

(since $k_i \leq k$). Recall that by definition k_i denotes the number of servers in B_s, after the ith jump involving block s. Summing these values for every block and for every jump, we get $(|\tau|+1)k$ as an upper bound, where $|\tau|$ is the total number of jumps. Hence, the sum of the expressions of the form $\frac{k_i \cdot f(k_i)}{\log k_i}\delta$ can be bounded by

$$(|\tau|+1)\frac{f(k)}{\log k}k\delta. \tag{4.13}$$

We also remark that

$$|\tau| \leq k, \tag{4.14}$$

since any server can jump at most once: after a server jumps into a block, the block has to be marked, thus no server can jump out from that given block in this phase.

Now we bound the cost of the jumps. Let T be the set of the potential associated runs of $\mathcal{AMA}$ and let η be the random variable $\mathcal{E}_\tau \mapsto |\tau|$, for

each $\tau \in T$. Applying Lemma 38 we get that the expected value of the total number of jumps is

$$\mathrm{E}[\eta] = \sum_{\tau \in T} \Pr(\mathcal{E}_\tau)|\tau| \leq \log k \cdot m_p \tag{4.15}$$

Summing up the results (4.11), (4.12), (4.13) and (4.15) for all the blocks and applying the law of total expectation we get the following bound for the expected cost of Algorithm $\mathcal{SH}$:

$$\begin{aligned}
& \sum_{s=1}^{t} \sum_{i=0}^{d_s^- + d_s^+} \mathrm{E}[\mathcal{A}_s(k_i, \sigma_i) + \eta\Delta] && (4.16)\\
= \; & \sum_{\tau \in T} \left(\Pr(\mathcal{E}_\tau) \sum_{s=1}^{t} \sum_{i=0}^{d_s^- + d_s^+} \mathrm{E}[\mathcal{A}_s(k_i, \sigma_i) | \, \mathcal{E}_\tau] \right) + \mathrm{E}[\eta]\Delta \\
\leq \; & \sum_{\tau \in T} \Pr(\mathcal{E}_\tau) \Bigg(f(k) \Bigg(\sum_{s=1}^{t} \mathrm{opt}_s(D_s(\varrho^{(p)+}), \varrho^{(p)+}) \\
& + \sum_{s=1}^{t} \big(D_s(\varrho^{(p)+}) - \hat{D}_s(p) \big)\Delta + |\tau|\Delta \Bigg) + (|\tau| + 1)\frac{f(k)}{\log k} k\delta \Bigg) \\
& + \log k \cdot m_p \cdot \Delta \\
\leq \; & \sum_{\tau \in T} \Pr(\mathcal{E}_\tau) f(k) \Bigg(\sum_{s=1}^{t} \mathrm{opt}_s(D_s(\varrho^{(p)+}), \varrho^{(p)+}) + \sum_{s=1}^{t} D_s(\varrho^{(p)+})\Delta - k\Delta \Bigg) \\
& + \sum_{\tau \in T} \Pr(\mathcal{E}_\tau)|\tau| \Bigg(f(k)\Delta + \frac{f(k)}{\log k} k\delta \Bigg) + \frac{f(k)}{\log k} k\delta + \log k \cdot m_p \cdot \Delta && (4.17)\\
\leq \; & f(k) \left(\sum_{s=1}^{t} \mathrm{opt}_s(D_s(\varrho^{(p)+}), \varrho^{(p)+}) + \sum_{s=1}^{t} D_s(\varrho^{(p)+})\Delta - k\Delta + \log k \cdot m_p \Delta \right) \\
& + \left(\frac{f(k)}{\log k} m_p \log k + \frac{f(k)}{\log k} + m_p \log k \right) \Delta,
\end{aligned}$$

if we apply (4.15), $\sum_{\tau \in T} \Pr(\mathcal{E}_\tau) = 1$ and $k\delta \leq \Delta$ in (4.17). □

Analyzing the optimal cost

Consider an optimal solution of the k-server problem. Let $C_s^*(\varrho)$ be the maximal number of servers in B_s of this optimal solution during ϱ and let $C_s(\varrho)$ be the number of servers in B_s of the optimal solution at the end ϱ. We modify ϱ as follows: we extend each phase (except the last one) with a copy of the first request of the next phase, and consider $\varrho^{(p)+}$ instead of $\varrho^{(p)}$. In this section we bound the optimal cost for this modified sequence. It is obvious that the optimal cost of these sequences is the same.

Observe that for any s and p, $C_s^*(\varrho^{(p)+}) \geq C_s(\varrho^{(p-1)+})$, since each (modified) phase p begins with the last configuration of phase $p-1$.
Then, $\sum_{s=1}^{t}(C_s^*(\varrho^{(p)+}) - C_s(\varrho^{(p-1)+}))$ is clearly a lower bound for the number of jumps of the optimal solution during (the modified) phase p. Thus, the cost of the optimal solution during $\varrho^{(p)+}$ (which has $C_s(\varrho^{(p-1)+})$, $s = 1, \ldots, t$ as the initial configuration) can be bounded by

$$\begin{aligned} \mathrm{opt}(k, \varrho^{(p)+}) \quad \geq \quad & \sum_{s=1}^{t}(C_s^*(\varrho^{(p)+}) - C_s(\varrho^{(p-1)+}))\Delta \\ & + \sum_{s=1}^{t} \mathrm{opt}_s(C_s^*(\varrho^{(p)+}), \varrho^{(p)+}), \end{aligned} \tag{4.18}$$

i.e., Δ times a lower bound for the number of jumps, plus a lower bound for the inner cost, where we treat each block as if we had the maximal number of servers during the whole phase.

Lemma 40

$$\mathrm{opt}(k, \varrho^{(p)+}) \geq \sum_{s=1}^{t} \mathrm{opt}_s(D_s(\varrho^{(p)+}), \varrho^{(p)+}) + \left(\sum_{s=1}^{t} D_s(\varrho^{(p)+}) - k\right)\Delta.$$

Proof. From Definition 34 we have

$$\begin{aligned} & \sum_{s=1}^{t} \left(\mathrm{opt}_s(C_s^*(\varrho^{(p)+}), \varrho^{(p)+}) + (C_s^*(\varrho^{(p)+}) - C_s(\varrho^{(p-1)+}))\Delta\right) \\ & \geq \sum_{s=1}^{t} \left(\mathrm{opt}_s(D_s(\varrho^{(p)+}), \varrho^{(p)+}) + (D_s(\varrho^{(p)+}) - C_s(\varrho^{(p-1)+}))\Delta\right). \end{aligned}$$

Since $\sum_{s=1}^{t} C_s(\varrho^{(p-1)+}) = k$, the statement follows by (4.19). □

Proposition 41 *For any s and p,*

$$\begin{aligned} & \mathrm{opt}_s(C_s^*(\varrho^{(p)+}), \varrho^{(p)+}) + (C_s^*(\varrho^{(p)+}) - C_s(\varrho^{(p-1)+}))\Delta \\ \geq\ & \max\{0, D_s^*(\varrho^{(p)+}) - C_s(\varrho^{(p-1)+})\}\Delta. \end{aligned}$$

Proof. Let $\varrho^{(p)*}$ be the subsequence of $\varrho^{(p)}$ which we get by omitting each request that arrives to a block B_s after the demand of that block reaches $D_s^*(\varrho^{(p)+})$ (note that $\varrho^{(p)*}$ is not necessarily a prefix of $\varrho^{(p)}$). Now we have two cases: first, if $D_s^*(\varrho^{(p)+}) > C_s(\varrho^{(p-1)+})$, then by Definition 34

$$\begin{aligned} & \mathrm{opt}_s(C_s^*(\varrho^{(p)+}), \varrho^{(p)+}) + (C_s^*(\varrho^{(p)+}) - C_s(\varrho^{(p-1)+}))\Delta \\ \geq\ & \mathrm{opt}_s(C_s^*(\varrho^{(p)+}), \varrho^{(p)*}) + (C_s^*(\varrho^{(p)+}) - C_s(\varrho^{(p-1)+}))\Delta \\ \geq\ & (\mathrm{opt}_s(D_s^*(\varrho^{(p)+}), \varrho^{(p)*}) + \left(D_s^*(\varrho^{(p)+}) - C_s(\varrho^{(p-1)+})\right))\Delta \\ \geq\ & \max\{0, D_s^*(\varrho^{(p)+}) - C_s(\varrho^{(p-1)+})\}\Delta. \end{aligned}$$

Otherwise it holds that

$$\max\{0, D_s^*(\varrho^{(p)+}) - C_s(\varrho^{(p-1)+})\} = 0,$$

and also obviously

$$\mathrm{opt}_s(C_s^*(\varrho^{(p)+}), \varrho^{(p)+}) + (C_s^*(\varrho^{(p)+}) - C_s(\varrho^{(p-1)+}))\Delta \geq 0.$$

□

Lemma 42 *The optimal cost is at least*

$$\frac{1}{6}\sum_{p>1} m_p \cdot \Delta.$$

Proof. Since $\sum_{s=1}^{t} \hat{D}_s(p) = \sum_{s=1}^{t} C_s(\varrho^{(p-1)+}) = k$, it holds that

$$\begin{aligned} & \sum_{s=1}^{t} \max\{0, \hat{D}_s(p) - C_s(\varrho^{(p-1)+})\}\Delta \\ =\ & \sum_{s=1}^{t} \frac{1}{2}|\hat{D}_s(p) - C_s(\varrho^{(p-1)+})|\Delta. \end{aligned} \tag{4.19}$$

Summing up the cost of the jumps performed by the optimal solution we get

$$\sum_{s=1}^{t} |C_s(\varrho^{(p)+}) - C_s(\varrho^{(p-1)+})|\Delta \leq 2 \cdot \mathrm{opt}(k, \varrho^{(p)+}). \tag{4.20}$$

Note that the factor of 2 comes from the fact that each jump appears twice on the left hand side. Applying to (4.19) the statement of Proposition 41 and equality (4.19), using $D_s^*(\varrho^{(p)+}) \geq \hat{D}_s(p)$ we get

$$\begin{aligned} & 2 \cdot \mathrm{opt}(k, \varrho^{(p)+}) \\ \geq & \sum_{s=1}^{t} \big(|\hat{D}_s(p) - C_s(\varrho^{(p-1)+})|\big)\Delta. \end{aligned} \tag{4.21}$$

Now summing (4.21) and (4.20) and applying the triangle inequality we get

$$\begin{aligned} & 4 \cdot \mathrm{opt}(k, \varrho^{(p)+}) \\ \geq & \sum_{s=1}^{t} \big(|\hat{D}_s(p) - C_s(\varrho^{(p-1)+})| + |C_s(\varrho^{(p)+}) - C_s(\varrho^{(p-1)+})|\big)\Delta \\ \geq & \sum_{s=1}^{t} |\hat{D}_s(p) - C_s(\varrho^{(p)+})|\Delta. \end{aligned} \tag{4.22}$$

Also, from summing (4.21) and (4.22), the latter relativized to phase $p-1$, and applying again the triangle inequality,

$$\begin{aligned} & 2 \cdot \mathrm{opt}(k, \varrho^{(p)+}) + 4 \cdot \mathrm{opt}(k, \varrho^{(p-1)+}) \\ \geq & \sum_{s=1}^{t} \big(|\hat{D}_s(p) - C_s(\varrho^{(p-1)+})| + |\hat{D}_s(p-1) - C_s(\varrho^{(p-1)+})|\big)\Delta \\ \geq & \sum_{s=1}^{t} |\hat{D}_s(p) - \hat{D}_s(p-1)|\Delta = m_p \cdot \Delta, \end{aligned} \tag{4.23}$$

and the statement follows. □

Proof of Theorem 37

Now we are able to prove the theorem about competitiveness of Algorithm $\mathcal{SH}$ (see page 45).

Proof. [Theorem 37] The first term in the right hand of the formula in Lemma 39 can be bounded by $f(k)\mathrm{opt}(k, \varrho^{(p)+})$ by Lemma 40. Furthermore if $p = 1$ we can write k instead of $m_1 \log k$ by (4.14), otherwise applying 42 we get that $\sum_p m_p \Delta$ can be bounded by $\frac{\Delta \cdot k}{\log k} + 6 \cdot \mathrm{opt}(k, \varrho)$.

Summing up,

$$\begin{aligned}
\mathrm{E}(\mathcal{SH}(\varrho)) &\leq f(k)\mathrm{opt}(k,\varrho) \\
&\quad + \left(\frac{\Delta \cdot k}{\log k} + 6 \cdot \mathrm{opt}(k,\varrho)\right)\big(f(k)\log k + f(k) + \log k\big) \\
&\quad + \frac{f(k)}{\log k}\Delta, \\
&= \mathrm{opt}(k,\varrho)\big(6f(k)\log k + 7f(k) + 6\log k\big) \\
&\quad + \frac{f(k)\log k + f(k) + \log k + f(k)/\log k}{\log k} \cdot k \cdot \Delta
\end{aligned}$$

hence Algorithm $\mathcal{SH}$ is (f', μ)-efficient on $\mathcal{M}$ with $f'(k) = O(f(k)\log k)$. □

4.2.2 Conclusions

Starting from algorithm Partition, Equitable or Marking and iterating Theorem 37 we get the following result:

Corollary 43 *There exists a $(c_1 \log k)^h$-competitive randomized online algorithm on any μ-HST of height h (here $\mu \geq k$), where c_1 is a constant. Consequently, when $h < \frac{\log k}{\log c_1 + \log\log k}$, this algorithm is $o(k)$-competitive.*

It is a bit more natural to require $\Delta \geq \delta n_0$ to hold, where n_0 is the size of the greatest block. If additionally $n_0 < k$ holds, we may get a better competitive ratio.

4.3 Results on the k-server problem with rejection

In the rest of the chapter we investigate a more general model in which the requests can be rejected. The results of this section are in [36]. In this problem the ith request is a pair (ϱ_i, w_i) for each i, where ϱ_i is a point and $w_i > 0$ is the *penalty* for the rejection. Each request can be served the same way as before, or optionally it also can be rejected at the penalty given along with the request. The cost of an algorithm is the sum of the distances covered by the k servers plus the sum of the penalties of the rejected requests.

4.3.1 Deterministic problem on uniform spaces

Theorem 44 *There is no weakly c-competitive online algorithm for the k-server problem with rejection on uniform spaces with $c < 2k$.*

Proof. Suppose that there exists a weakly $(2k - c')$-competitive algorithm $\mathcal{A}$ on an uniform space $\mathcal{M}$ with $c' > 0$, $a \geq 0$. We construct the following input sequence ϱ for $\mathcal{A}$. Let be C the initial configuration and $v \notin C$ a point of $\mathcal{M}$. Let $N > k^2(2 + a + (a + 3)/2)^2$ be a large positive integer and $\varepsilon = 1/\sqrt{N}$. Each element of ϱ consists of the point of $C \cup \{v\}$ which is not in the current configuration of algorithm $\mathcal{A}$ and a penalty ε. The request sequence ends when the number of server movements of $\mathcal{A}$ achieves N (we can suppose that it achieves N, otherwise it is easy to see that there is no such c for which $\mathcal{A}$ is c-competitive). We note that there are N requests (including the first) differing form the preceding one. Let W stand for the sum of the penalties payed by algorithm $\mathcal{A}$. So the cost of $\mathcal{A}$ on input ϱ is $\mathcal{A}(k, \varrho) = N + W$. We deal with the following two cases.

<u>*Case 1.*</u> $N \cdot (1 - 2\varepsilon) < W$.
Obviously the optimal cost is at most $\lceil N/k \rceil$, which is the cost of serving all requests with the server covering a point which will be requested at the very latest after the current request. In this solution at least $k - 1$ requests arriving to different points covered by servers follow each server movement. Applying this and the assumption on $\mathcal{A}$ we get that

$$\begin{aligned} \mathcal{A}(k, \varrho) &\leq (2k - c') \cdot \mathrm{opt}(k, \varrho) + a \\ &< (2k - c') \cdot (N/k + 1) + a \\ &= N \cdot (2 - c'/k) + (2k - c' + a), \end{aligned}$$

with $c' > 0$ and some a. Moreover by the assumption of this case

$$\mathcal{A}(k, \varrho) = N + W > N \cdot (2 - 2\varepsilon) = 2N - 2\sqrt{N}$$

but it is a contradiction by the choice of N.

<u>*Case 2.*</u> $N \cdot (1 - 2\varepsilon) \geq W$.
Clearly, the number of requests is $N + W/\varepsilon$, so the average number of requests arriving to a point is $r = (N + W/\varepsilon)/k$. Therefore, there exists a point u to which at most r requests arrive. The cost of the solution which initially moves the server covering u to v if $u \neq v$ and rejects all requests which are not covered, therefore also the optimal cost, is at most $r\varepsilon + 1$. Applying this

and the assumption on $\mathcal{A}$ we get that

$$\begin{aligned}\mathcal{A}(k,\varrho) &\leq (2k-c')\cdot \mathrm{opt}(k,\varrho)+a\\ &< (2k-c')\cdot\left(\frac{N\varepsilon+W}{k}+1\right)+a\\ &= 2N\varepsilon+2W-c'\cdot\frac{N\varepsilon+W-k-ka/c'}{k}\\ &= 2\sqrt{N}+2W-c'\cdot\frac{\sqrt{N}+W-k-ka/c'}{k}.\end{aligned}$$

On the other hand by the assumption of this case

$$\mathcal{A}(k,\varrho)=N+W>2N\varepsilon+2W=2\sqrt{N}+2W$$

but it is again a contradiction by the choice of N. □

We present a weakly competitive algorithm for the k-server problem with rejection on uniform spaces called Threshold. Algorithm Threshold uses the marking procedure seen before and is picky. Let $t>0$ be some fixed value.

Algorithm $\mathcal{TH}_t$

To each point a counter is assigned. Initially each counter is set to 0 and all the vertices are unmarked. After each request, the marks are updated, followed by a server movement if necessary, as follows:

(i) Increase the value of the counter of the requested point by the penalty of the request.

(ii) If the value assigned to the requested point is at least t, mark that point. At the moment when $k+1$ points become marked, all the marks (including this new one) are erased and all counters are set to 0.

(iii) If the requested point is already covered by a server, then no servers shall move.

(iv) If the requested point is unmarked and not covered then the request is rejected.

(v) If the requested point is marked and not covered, then the least recently moved server is moved to cover the requested point.

Theorem 45 *Algorithm $\mathcal{TH}_1$ is weakly $(2k+1)$-competitive on uniform spaces.*

Proof. Let $\mathcal{M}$ be a uniform space and $\varepsilon > 0$ be arbitrary. First of all we make the following observations. If in the request sequence we exchange a request with several requests arriving to the same point such that the sum of the penalties of the new requests is equal to the penalty of the old request, the optimal cost does not increase and the cost of Algorithm $\mathcal{TH}_1$ does not decrease. Therefore we can suppose without loss of generality that the penalty of each request is at most ε. The second observation is that we can divide the input sequence into *phase*s by the marking procedure in the following way: a phase ends with the request which causes mark erasing and new phase begins immediately after this request. The point where this request arrives is called the *terminating point* of the phase.

We call the points of $\mathcal{M}$ having a counter value strictly between 0 and $1(=t)$ at the end of phase p *non-critical points* of phase p. Denote n_p the number of points of $\mathcal{M}$ to which at least one request arrives in phase p. If p is not the last phase, then $n_p > k$. Let ℓ denote the number of points having a counter value at least 1 at the end of the last phase. There are exactly $n_p - \ell$ non-critical points of phase p if p is the last phase and $n_p - k - 1$ such points if p is not the last one. There are at least $n_p - k$ points to which requests arrive increasing the optimal cost in phase p (either by penalty or by the cost of server moving). Let W_p stand for the sum of the values of those counters belonging to non-critical points of phase p. Therefore the optimal cost in phase p is at least $W_p + 1$ if p is not the last phase, otherwise it is at least $\max\{W_p - k + \ell, 0\}$, so the total optimal cost is

$$\mathrm{opt}(k, \varrho) \geq \sum_p (W_p + 1) - 1 - k + \ell \geq \sum_p (W_p + 1) - 1 - k.$$

The cost of Algorithm $\mathcal{TH}_1$ is at most

- $W_p + 2\ell$ in the last phase,
- $W_p + 2k + 1$ otherwise,

therefore

$$\begin{aligned} \mathcal{TH}_1(k, \varrho) &\leq \sum_p (W_p + 2k + 1 + \varepsilon) - 1 - 2k + 2\ell \\ &\leq (2k + 1 + \varepsilon)(\mathrm{opt}(k, \varrho) + 1 + k) - 1 \end{aligned}$$

holds with the chosen ε. But since ε was chosen arbitrarily, we get that

$$\mathcal{TH}_1(k, \varrho) \leq (2k+1)\mathrm{opt}(k, \varrho) + 2k^2 + 3k.$$

□

4.3.2 Randomized problem on uniform spaces

Finally we investigate the randomized version of Algorithm $\mathcal{TH}_t$. Let $t > 0$ be some fixed value.

Algorithm $\mathcal{RTH}_t$

To each point a counter is assigned. Initially each counter is set to 0 and all the vertices are unmarked. After each request, the marks are updated, followed by a server movement if necessary, as follows:

(i) Increase the value of the counter of the requested point by the penalty of the request.

(ii) If the value assigned to the requested point is at least t, mark that point. At the moment when $k+1$ points become marked, all the marks (including this new one) are erased and all counters are set to 0.

(iii) If the requested point is already covered by a server, then no servers shall move.

(iv) If the requested point is unmarked and not covered then the request is rejected.

(v) If the requested point is marked and not covered, then a server is chosen uniformly at random from among the ones occupying an unmarked vertex and is moved to cover the requested point.

Recall that $\mathrm{H}_k = \sum_{i=1}^{k} 1/k$.

Theorem 46 *Algorithm* $\mathcal{RTH}_2$ *is* $(6\mathrm{H_k}+2)$*-competitive on uniform spaces.*

Proof. Analogously the deterministic case we again can assume that the penalty of each request is at most ε. We can divide the input sequence into phases like in the deterministic case. The marking procedure, so also each

phase, does not depend on the random choices of Algorithm $\mathcal{RTH}_2$. We use the notations of the proof of Theorem 45.

At first we give lower bounds to the optimal cost. Now the non-critical points of phase p are exactly the points with counter value strictly between 0 and 2 at the end of phase p. There are exactly $n_p - \ell$ non-critical points of phase p if p is the last phase and $n_p - k - 1$ such points if p is not the last one. There are at least $n_p - k$ points to which requests arrive increasing the optimal cost in phase p (either by penalty or by the cost of server moving). Now the requests arriving to such a point increase the optimal cost by

- at least 1 if the sum of the penalties exceeds 1;
- or, by the sum of these penalties (i.e., the value of the counter of the given point at the end of the phase) otherwise.

In any case, each non-critical point of a phase p contributes by at least half of the values of their counters at the end of the phase (since by non-criticalness, each of these counters is bounded by 2).

Therefore,

$$\mathrm{opt}(k, \varrho) \geq \sum_p (\frac{W_p}{2} + 1). \tag{4.24}$$

Moreover, the optimal cost does not increase if we omit any request arriving to non-critical points from the request sequence. Consider an optimal solution. Without loss of generality we can suppose that it is lazy, i.e., it does not move any server in response to a request to a covered point and moves at most one server in response to a request to an uncovered point. For any given algorithm, there is always a lazy one that incurs no more cost in the original k-server problem (see in [31]) so there is a lazy one in the k-server problem with rejection too. Call a request causing server movement according to Algorithm $\mathcal{RTH}_2$ a *critical request* and call a point to which a critical request arrives in phase p a *critical point* of phase p. There are at most k critical points in each phase. Consider a critical point v in phase p. Suppose that the chosen optimal solution does not move any server to v in phase p. So there exists a server s which does not move in phase p. The value of the counter of v is more than 2 at the end of phase p. Therefore the cost of the considered solution decreases if we modify the solution such that s moves to v at the beginning and return to its original position at the end of phase p, but this is a contradiction by the optimality of the solution. Consider the critical requests of phase p. Denote ϱ' the subsequence of ϱ consisting of the requests arriving to critical points of the actual phase. Denote ϱ^c the sequence of the

critical requests with penalties increased to infinity. So we can suppose that each request in ϱ' is served by the considered optimal solution and we get that

$$\mathrm{opt}(k, \varrho) \geq \mathrm{opt}(k, \varrho') \geq \mathrm{opt}(k, \varrho^c). \tag{4.25}$$

The cost of $\mathcal{RTH}_2$ in phase p is the cost on critical points of phase p plus the sum of the penalties of requests arriving to non-critical points of phase p plus an additional cost of 2, which is the sum of the penalties of the requests arriving to the terminating point of phase p. Observe that each critical point causes at most 2+1 cost in Algorithm $\mathcal{RTH}_2$ and the server moving is according to Algorithm $\mathcal{MARK}$ on input ϱ^c. Moreover we know that Algorithm $\mathcal{MARK}$ is $2\mathrm{H_k}$-competitive. Applying these facts and also inequalities (4.24) and (4.25) we get

$$\begin{aligned}
\mathrm{E}[\mathcal{RTH}_2(k, \varrho)] &= \sum_p (W_p + 2) + \mathrm{E}(\mathcal{RTH}_2(k, \varrho')) \\
&\leq \sum_p (W_p + 2) + (2 + 1) \cdot \mathrm{E}(\mathcal{MARK}(\varrho^c)) \\
&\leq 2 \cdot \mathrm{opt}(k, \varrho) + 3 \cdot 2\mathrm{H}_k \cdot \mathrm{opt}(k, \varrho^c) \\
&\leq (6\mathrm{H}_k + 2)\mathrm{opt}(k, \varrho) + 2k.
\end{aligned}$$

□

Obviously, the original k-server problem is a special case of the k-server problem with rejection, chosen all penalties to be infinity. Therefore the lower bound $\mathrm{H_k}$ of the competitive ratio holds in this model also.

Summary

This thesis is concerned with online problems. There are several families of problems of which an online version is defined. Here were considered three areas: the scheduling, the graph coloring and its generalization for hypergraphs and the k-server problem.

At first the thesis summarizes the basic notations of competitive analysis in general.

The first family of problems is the family of *online scheduling*, in which jobs are given with processing time, which have to be assigned to machines handling them. Jobs arrive one by one, and they have to be scheduled immediately at their arrival. The cost is the processing time of the most loaded machine (called the makespan) plus extra costs if there are any. In the model with machine cost [25] one may purchase machines. Algorithm called $\mathcal{A}_\rho$ [25] is φ-competitive in this model. In the model with rejection the online algorithm may reject jobs by paying some penalty [6]. Algorithm called $\mathcal{RTP}(\alpha)$ [6] is $(\varphi+1)$-competitive and no online algorithm exists that is c-competitive for some $c < \varphi + 1$ in this model. The surprising fact turns out from investigation of the combination of the models above that the simple combinations of $\mathcal{A}_\rho$ and $\mathcal{RTP}(\alpha)$ are not efficient, i.e., they are not c-competitive for any constant c in the combined model. In this situation the investigation of the relaxed version of the problem helps. There exists a solution (Relopt) of the relaxed problem having a behavior in terms of accepting which is consistent on the sequence of prefixes of the input, i.e. if it accepts a job on a prefix then it doesn't rejects that job on any longer prefix, and this solution can be computed in polynomial time. Furthermore the sum of the penalties of job which at first were rejected but later were accepted by Relopt can be bounded above with the optimal cost. Algorithm Optcopy was constructed on the basis of these results. This algorithm rejects the most recent job if Relopt does at least once and schedules the accepted jobs according to $\mathcal{A}_\rho$. The main result of this part that algorithm Optcopy is $(\varphi + 1)$-competitive.

The second family, the *online graph coloring*, is a well studied problem. There is an ordering given on the vertices of the input graph and the vertices are revealed one by one in this order and have to be colored immediately seeing only the subgraph spanned by the already revealed vertices. The cost of an algorithm is obviously the number of colors it uses. At first the generalizations of Algorithm $\mathcal{B}_n$ [27] are considered which colors graphs inducing neither C_3 nor C_5 for graphs with high girth and high oddgirth. The first generalization is $\mathcal{B}_{n,d}$, which colors online graphs on n vertices with girth greater than $4d+1$ with at most $(d+1)n^{1/(d+1)}$ colors. The second generalization is $\mathcal{BO}_{n,d}$, which uses an algorithm defined by Lovász which colors bipartite graphs on n' vertices with at most $\log_2 n'$ colors (see [26]) as an auxiliary algorithm. $\mathcal{BO}_{n,d}$ colors online graphs on n vertices with oddgirth greater than $4d+1$ with at most $2(2n\log_2 2d/d)^{1/2}$ colors.

Graph coloring can be generalized to hypergraphs in several ways. In the generalization considered here an ordering on the vertices is given, and the vertices are revealed one by one in this order and have to be colored immediately in such a way that edges containing only vertices which are already revealed don't become monochromatic. The most natural and simplest algorithm is the First Fit ($\mathcal{FF}$). It turned out that on several classes of hypergraphs $\mathcal{FF}$ is the best online algorithm according to the competitive analysis. The results in this model are the following.

Let $k \geq 3$. For every online hypergraph coloring algorithm $\mathcal{A}$ there exists a 2-colorable k-uniform hypergraph H on n vertices for which $\chi_{\mathcal{A}}(H) \geq \lceil n/(k-1) \rceil$.
If H is a k-uniform hypergraph then $\chi_{\mathcal{FF}}(H) \leq \lceil n/(k-1) \rceil$.

For every online hypergraph coloring algorithm $\mathcal{A}$ and integer $d > 2$ there exists a 2-colorable d-uniform hypergraph H on at most $\frac{(d-1)^k-1}{d-2}$ vertices with maximal degree k such that $\chi_{\mathcal{A}}(H) \geq k+1$.
Furthermore, $\mathcal{FF}$ colors hypergraphs with maximal degree k with at most $k+1$ colors.

For any hypergraph H Algorithm $\mathcal{FF}$ gives a coloring of H with at most $2 \cdot \nu(H)+1$ colors where $\nu(H)$ is the matching number of H, i.e., the maximal number of pairwise disjoint edges of H.

No online algorithm exists which can color a projective plane with less than 3 colors.
Furthermore, $\mathcal{FF}$ colors the projective planes with at most 3 colors.

At the end of the thesis some results on the *k-server problem* are given. There are several metric spaces on which $o(k)$-competitive randomized algorithms (the competitive ratio of which does not depend on any parameter of the space) for the k-server problem have been developed. Algorithm Shell ($\mathcal{SH}$) is one of them. It works on so-called decomposable spaces, and uses an algorithm $\mathcal{A}$ which serves the requests inside the blocks as auxiliary algorithm. The basis of execution of $\mathcal{SH}$ is the notion of the demand of a block, which means the number of servers for which the cost of moving them into the (empty) block plus the optimal cost inside the block is minimal. $\mathcal{SH}$ works in phases and keeps the number of servers in the block never less than their demand. If it moved a server into a block in a phase it will not move server outside from that block during the phase. The requests inside the block are served by $\mathcal{A}$. If $\mathcal{A}$ is $f(k)$-competitive in each block then $\mathcal{SH}$ is $c \cdot f(k) \log k$-competitive when using $\mathcal{A}$. Furthermore, if the metric space is a so-called μ-HST for some $\mu > k$ and height h, then starting from a $c'\mathrm{H}_k$-competitive algorithm one can define a $(c_1 \log k)^h$-competitive algorithm applying $\mathcal{SH}$ in a recursive manner. If $h < c_2 \log k / \log\log k$, then this algorithm is $o(k)$-competitive.

The extension of the k-server problem with rejection in which the algorithm may reject the request by paying some penalty is also considered. Algorithm $\mathcal{TH}$ is constructed for the deterministic version of the model, and $\mathcal{RTH}$ is a randomized variant. Both of them use a marking procedure hence they work in phases as Algorithm $\mathcal{MARK}$ [21] does. The corresponding results are the following.

> *There is no weakly c-competitive online algorithm for the k-server problem with rejection on uniform space with $c < 2k$.*
> *Furthermore, Algorithm $\mathcal{TH}$ (with appropriate parameter) is weakly $(2k+1)$-competitive on uniform spaces.*

> *Obviously, the lower bound $\mathrm{H_k}$ of the competitive ratio of the original model holds in this model also.*
> *Furthermore, Algorithm $\mathcal{RTH}$ (with appropriate parameter) is $(6\mathrm{H_k} + 2)$-competitive on uniform spaces.*

The spectrum of the problems tackled in the thesis is wide, thus it is not fully comprehensive, but rather opens several (hopefully) promising directions for future research.

Összefoglalás

A dolgozat online problémákkal foglalkozik. Több problémacsaládnak definiálták online változatát. Ezek közül három terület került elő: az ütemezés, a gráfszínezés, illetve annak egy általánosítása hipergráfokra, valamint a k-szerver probléma.

A dolgozat eleje a kompetitív elemzéssel kapcsolatos legfontosabb alapfogalmakat foglalja össze általánosan.

Az első problémakör az *online ütemezés*, amelyben ütemezési idővel rendelkező munkák érkeznek sorban, amelyeket gépekhez kell rendelni. A költség a legtovább futó gép működési ideje (makespan) és az esetleges egyéb költségek összege. A gépköltséges modellben [25] a gépeket 1 költségért lehet vásárolni, az elutasításos modellben [6] pedig a munkákat bizonyos büntetés fejében el lehet utasítani. Mindkét modellben ismert hatékony algoritmus: az előbbiben a φ-kompetitív $\mathcal{A}_\rho$ algoritmus [25], míg az utóbbiban a $(\varphi+1)$-kompetitív $\mathcal{RTP}(\alpha)$ [6], amelyről igazolták azt is, hogy nem létezik nála hatékonyabb online algoritmus a kompetitív elemzés szempontjából. A két modell kombinációjának vizsgálatából kiderül az a meglepő tény, hogy a fenti két hatékony algoritmus egyszerű kombinációi nem hatékonyak, azaz nem létezik olyan c konstans, amelyre c-kompetitív lenne bármelyik is. Segítséget a probléma relaxált változatának vizsgálata jelentett. A relaxált változatnak létezik olyan optimális megoldása (Relopt), amely a munkák elfogadását tekintve konzisztensen viselkedik a input kezdőszeleteinek sorozatán (rövidebb kezdőszeleten elfogadott munkát hosszabbon sem utasít el). Az is teljesül, hogy ez a megoldás polinom időben kiszámítható, illetve a Relopt által korábban elutasított, de később elfogadott munkák összbüntetése felülről korlátozható az optimális költséggel. Ezek alapján lett megkonstruálva az Optcopy algoritmus, amely a munkák elfogadását tekintve a Relopt algoritmus döntését alkalmazza a legutolsó munkára, az elfogadott munkákat pedig $\mathcal{A}_\rho$ segítségével ütemezi. Ennek a résznek a fő eredménye, hogy az Optcopy algoritmus $(\varphi+1)$-kompetitív.

A második problémacsalád az *online színezési* problémákat foglalja magába. Az eredeti gráfszínezési problémában adott az input gráf csúcsain egy sorrend, ebben a sorrendben érkeznek a csúcsok. A soron következő csúcs színét azonnal meg kell adni, az eddig érkezett csúcsok által feszített részgráfot látva. A költség nyilvánvalóan a használt színek száma. Először a C_3- és C_5-mentes gráfokat színező $\mathcal{B}_n$ [27] algoritmus általánosítása került sorra nagy derékbőségű illetve nagy páratlan derékbőségű gráfokra. Az egyik általánosított algoritmus $\mathcal{B}_{n,d}$, amely n csúcsú, $4d+1$-nél nagyobb derékbőségű gráfokat színez online módon legfeljebb $(d+1)n^{1/(d+1)}$ színnel. A másik $\mathcal{BO}_{n,d}$, amely segédalgoritmusként Lovász páros gráfokat (legfeljebb a csúcsszámban logaritmikus számú színnel) színező online algoritmusát (ld. [26]) használja. A $\mathcal{BO}_{n,d}$ algoritmus olyan n-csúcsú gráfokat színez online módon legfeljebb $2(2n\log_2 2d/d)^{1/2}$ színnel, amelyek nem tartalmaznak $4d+2$-nél rövidebb páratlan kört.

Az online gráfszínezés hipergráfokra többféle módon általánosítható. Az itt tekintett általánosításban az online hipergráf csúcsain szintén adott egy sorrend, és a sorrendben érkező csúcs színét azonnal meg kell adni úgy, hogy azon élek közül, amelyek csak addig érkezett csúcsokat tartalmaznak, egyik se legyen monokromatikus. Az egyik legtermészetesebb és legegyszerűbb algoritmus a First Fit ($\mathcal{FF}$), amelyről több hipergráfosztály esetében kiderült, hogy nincs is nála hatékonyabb algorimus. A dolgozatban szereplő eredmények ebben a modellben a következők:

Legyen $k \geq 3$. Bármely $\mathcal{A}$ online algoritmusra van olyan n csúcsú 2-színezhető k-uniform online hipergráf, amely színezéséhez $\mathcal{A}$ legalább $\lceil n/(k-1)\rceil$ színt használ.
Továbbá $\mathcal{FF}$ bármely n csúcsú 2-színezhető k-uniform online hipergráf színezéséhez legfeljebb $\lceil n/(k-1)\rceil$ színt használ.

Bármely $\mathcal{A}$ online algoritmusra és $d > 2$ és k pozitív egészekre létezik olyan 2-színezhető d-uniform legfeljebb $\frac{(d-1)^k-1}{d-2}$ csúcsú és k maximális fokú hipergráf, amely színezéséhez $\mathcal{A}$ legalább $k+1$ színt használ
Továbbá a k maximális fokú hipergráfok színezéséhez $\mathcal{FF}$ legfeljebb $k+1$ színt használ.

Minden H online hipergráfra az $\mathcal{FF}$ által használt színek száma legfeljebb $2\cdot\nu(H)+1$, ahol $\nu(H)$ a maximális független élrendszer elemszáma.

$\mathcal{FF}$ a véges projektív síkok színezéséhez legfeljebb 3 színt használ.
Továbbá nincs olyan online algoritmus, amely egy véges projektív síkot 3-nál kevesebb színnel tudna színezni.

A dolgozat végén a *k-szerver problémá*val kapcsolatos eredmények szerepelnek. A véletlen k-szerver problémára kevés metrikus téren ismert $o(k)$-kompetitív algoritmus (ahol az algoritmus kompetitív hányadosa nem függ a tér más paraméterétől). Az egyik eredmény ezen algoritmusok tárát bővíti. Úgynevezett μ-felbontható tereken működik a Shell algoritmus ($\mathcal{SH}$), amely segédalgoritmusként egy, a tér blokkjain hatékony $\mathcal{A}$ algoritmust használ. Az algoritmus működésének lényege, hogy az igény (demand) fogalmát definiálja a blokkokon, amely tulajdonképpen az a szerverszám, amelyre a szerverek bemozgatásának költsége (az üres blokkba) plusz a blokkon belül az optimális költség minimális. Az algoritmus fázisokban működik, minden blokkban legalább annyi szervert tart, amennyi az igény, és amelyik blokkba vitt szervert, onnan már nem visz el abban a fázisban. A blokkokon belül az $\mathcal{A}$ algoritmust használja a kérések kiszolgálására. Ha $\mathcal{A}$ algoritmus $f(k)$-kompetitív a blokkokon, akkor $\mathcal{SH}$ algoritmus $\mathcal{A}$-val $c \cdot f(k) \log k$-kompetitív. Továbbá egy h magas μ-HST-n, ahol $\mu > k$, valamelyik, az uniform téren $c' \cdot \mathrm{H}_k$-kompetitív algoritmusból indulva $\mathcal{SH}$ algoritmust rekurzív módon alkalmazva kapunk egy $(c_1 \log k)^h$-kompetitív algoritmust. Ha $h < c_2 \log k / \log\log k$, akkor a kapott algoritmus $o(k)$-kompetitív.

A k-szerver problémának definiálható elutasításos modellje is, amelyben a kéréseket – bizonyos büntetés fizetése mellett – el is lehet utasítani. A modell determinisztikus változatában konstruált algoritmus $\mathcal{TH}$, a véletlen változatában pedig $\mathcal{RTH}$. Mindkettő – a $\mathcal{MARK}$ [21] algoritmushoz hasonlóan – jelölő algoritmus. A velük kapcsolatos eredmények a következők:

> *A k-szerver probléma elutasításos változatára uniform téren nincs olyan (determinisztikus) online algoritmus, amely gyengén c-kompetitív lenne valamely $c < 2k$-ra.*
> *Továbbá $\mathcal{TH}$ algoritmus (megfelelő paraméterrel) $(2k+1)$-kompetitív uniform tereken.*
>
> *$\mathcal{RTH}$ algoritmus (megfelelő paraméterrel) $(6\mathrm{H}_k+2)$-kompetitív uniform tereken.*
> *Továbbá az eredeti probléma H_k alsó korlátja itt is alsó korlát a véletlen algoritmusok kompetitív hányadosára.*

Az áttekintett problémák köre viszonylag széles. Habár ennélfogva nem teljesen átfogó, sok további lehetőséget nyit meg.

Acknowledgement

I thank Péter Hajnal for the guidance and for the lots of help.
I am also thankful to my coauthor, Csanád Imreh for his time and for the joint work.
I thank to Béla Csaba for his help and useful advices.

Index

Bibliography

[1] D. Achlioptas, M. Chrobak, J. Noga, Competitive Analysis of Randomized Paging Algorithms, *Theor. Comput. Sci.* **234** (2000), 203–218.

[2] N. Alon, U. Arad, Y. Azar, Independent Sets in Hypergraphs with Applications to Routing Via Fixed Paths, *Proc. of APPROX 99, LNCS 1671* (1999), 16–27.

[3] N. Alon, R. M. Karp, D. Peleg, D. West, A graph-theoretic game and its application to the k-server problem, *SIAM Journal on Computing* **24** (1995), 78–100.

[4] Y. Bartal, On approximating arbitrary metrics by tree metrics, *Proceedings of 30th Annual ACM Symposium on Theory of Computing* (1998), 161–168.

[5] Y. Bartal, B. Bollobás, M. Mendel, A Ramsey-type theorem for metric spaces and its application for metrical task system and related problems, *Journal of Computer and System Sciences* **72** (2006), 890–921.

[6] Y. Bartal, S. Leonardi, A. Marchetti-Spaccamela, J. Sgall, L. Stougie, Multiprocessor scheduling with rejection, *SIAM Journal on Discrete Mathematics* **13** (2000) 64–78.

[7] Y. Bartal, M. Mendel, Randomized k-server algorithms for growth-rate bounded graphs, *Journal of Algorithms* **55** (2005), 192–202.

[8] Y. Bartal, N. Linial, M. Mendel, A. Naor, On metric Ramsey-type phenomena, *Annals of Mathematics*, **162** (2005), 643–709.

[9] S. Ben-David, A. Borodin, R. Karp, G. Tardos, A. Wigderson, On the power of randomization in on-line algorithms, *Algorithmica* **11** (1994), 2–14.

[10] C. Berge, *Hypergraphs, Combinatorics of finite sets*, North-Holland, Amsterdam, New York, 1989

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

[12] F. Buekenhout (ed.) *Handbook of incidence geometry. Buildings and foundations*, North-Holland, Amsterdam, 1995

[13] B. Csaba, S. Lodha, A randomized on-line algorithm for the k-server problem on a line, *Random Structures and Algorithms* **29** (2006), 82–104.

[14] B. Csaba, A. Pluhár, A randomized algorithm for on-line weighted bipartite matching problem, *Journal of Scheduling* **11** (2008), 449–455.

[15] J. Csirik, Cs. Imreh, J. Noga, S. S. Seiden, G. Woeginger, Buying a constant competitive ratio for paging, *Proceedings of ESA 2001, LNCS 2161*, 2001, 98–108.

[16] Gy. Dósa, Y. He, Better online algorithms for scheduling with machine cost, *SIAM Journal of Computing* **33** (2004), 1035–1051.

[17] Gy. Dósa, Y. He, Bin packing problems with rejection penalties and their dual problems, *Information and Computation* **204** (2006), 795–815

[18] Gy. Dósa, Y. He, Scheduling with machine cost and rejection, *Journal of Combinatorial Optimization* **12** (2006), 337–350

[19] L. Epstein, A. Levin, G. J. Woeginger, Graph coloring with rejection, *Proc. of ESA 2006, LNCS 4168* (2006), 364–375.

[20] P. Erdős, Graph theory and probability. *Canad. J. Math.* **11**, 34–38, 1959.

[21] A. Fiat, R. M. Karp, M. Luby, L. McGeoch, D. Sleator, N. E. Young, Competitive paging algorithms, *Journal of Algorithms* **12** (1991), 685–699.

[22] J. Fakcharoenphol, S. Rao, K. Talwar, A tight bound on approximating arbitrary metrics by tree metrics, *Journal of Computer System Science* **69** (2004), 485–497.

[23] R.L. Graham, Bounds for certain multiprocessor anomalies, *Bell System Technical Journal* **45** (1966), 1563–1581.

[24] A. Gyárfás, J. Lehel, On-line and first-fit colorings of graphs, *Journal of Graph Theory* **12** (1988), 217–227.

[25] Cs. Imreh, J. Noga, Scheduling with Machine Cost, *Proc. APPROX'99, Lecture Notes in Computer Science*, Vol. 1761, Springer, Berlin, 1999, 168–176.

[26] H. A. Kierstead, Coloring Graphs On-line, in: A. Fiat, and G. J. Woeginger (eds.) *Online algorithms: The State of the Art, LNCS 1442*, 1998, 281–305.

[27] H. A. Kierstead, On-line coloring k-colorable graphs, *Israel Journal of Mathematics* **105** (1998), 93–104.

[28] E. Koutsoupias, C. Papadimitriou, On the k-server conjecture, *Journal of the ACM* **42** (1995), 971–983.

[29] L. Lovász, M. Saks, W. T. Trotter, An on-line graph coloring algorithm with sublinear performance ratio, *Discrete Mathematics* **75** (1989), 319–325.

[30] L. McGeoch, D. Sleator, A strongly competitive randomized paging algorithm, *Algorithmica* **6** (1991), 816–825.

[31] M. Manasse, L. McGeoch, D. Sleator, Competitive algorithms for server problems, *Journal of Algorithms* **11** (1990), no. 2, 208–230.

[32] J. Nagy-György, Randomized algorithm for the k-server problem on decomposable spaces, *Journal of Discrete Algorithms* (2009), doi: 10.1016/j.jda.2009.02.005

[33] J. Nagy-György, Online coloring graphs with high girth and high odd-girth, submitted

[34] J. Nagy-György, Cs. Imreh, Online scheduling with machine cost and rejection, *Discrete Applied Mathematics* **155** (2007), 2546–2554.

[35] J. Nagy-György, Cs. Imreh, Online hypergraph coloring, *Information Processing Letters* **109** (2008), 23–26.

[36] J. Nagy-György, Cs. Imreh, k-server problem with rejection, manuscript

[37] P. Raghavan, A statistical adversary for on-line algorithms, in: *On-line algorithms, DIMACS, Series in Discrete Mathematics and Theoretical Computer Sc., AMS, Providence, RI/Assoc. for Comp. Mach.* vol. 7, New York, 1991, 79–83

[38] S. S. Seiden, Preemptive Multiprocessor Scheduling with Rejection, *Theoretical Computer Science* **262** (2001), 437–458.

[39] S. S. Seiden, A general decomposition theorem for the k-server problem, *Information and Computation* **174** (2002), 193–202.

[40] J. Sgall, On-line scheduling, in: A. Fiat, G.J. Woeginger (eds.) *Online algorithms: The State of the Art, Lecture Notes of Computer Science*, Vol. 1442, Springer-Verlag Berlin, 1998, 196–231.

[41] S. Vishwanathan, Randomized on-line graph coloring, *Journal of Algorithms*, **13** (1992), 657–669.