8. Online Scheduling

In online computation, an algorithm must make its decisions based only on past events without secure information on future. Such algorithms are called ***on-line algorithms***. Online algorithms have many applications in different areas such as computer science, economics and operations research.

The first results in this area appeared around 1970, and later since 1990 more and more researchers started research on problems related to on-line algorithms. Many subfields were developed and investigated. Nowadays new results of the area are presented on the most important conferences about algorithms. It is not the goal of this chapter to give a detailed overview about the results, this would not be possible in this framework. The goal of the chapter is to show some of the main methods of analysing and developing on-line algorithms by presenting some subareas in more details.

In the next section we define the basic notions used in the analysis of on-line algorithms. After giving the most important definitions we present one of the most known on-line problems – the on-line k-server problem – and some of the related results. Then we deal with a new area we present on-line problems belonging to computer networks. In the next section the on-line bin packing problem and its multidimensional generalizations are presented. Finally in the last chapter of the section we show some of the basic results concerning the area of on-line scheduling.

8.1. Notions, definitions

Since an on-line algorithm makes its decisions by partial information without knowing the whole instance in advance we cannot expect that it gives the optimal solution which can be given by an algorithm having full information. An algorithms which knows the whole instance in advance is called ***offline algorithm***.

There are two main methods to measure the performance of on-line algorithms. One possibility is to use ***average case analysis*** where we hypothesize some distribution on events and we study the expected total cost.

The disadvantage of this approach is that usually we do not have any information about the distribution of the possible inputs. In this chapter we do not use the average case analysis.

An another approach is a worst case analysis, which is called ***competitive analysis***. In

this case we compare the objective function value of the solution produced by the on-line algorithm to the optimal offline objective function value. Since we use this measure in this chapter we give the related definitions below.

In case of on-line minimization problems an on-line algorithm is called ***C*-competitive**, if the cost of the solution produced by the on-line algorithm is at most C times more than the optimal offline cost for each input. The ***competitive ratio of an algorithm*** is the smallest such C for which the algorithm is C-competitive.

For an arbitrary on-line algorithm ALG we denote the objective function value achieved on input I by $\mathrm{ALG}(I)$. The optimal offline objective function value on I is denoted by $\mathrm{OPT}(I)$. Using this notation we can define the competitiveness as follows.

Algorithm ALG is C-competitive, if $\mathrm{ALG}(I) \leq C \cdot \mathrm{OPT}(I)$ is valid for each input I.

Two further versions of the competitiveness are often used. For a minimization problem an algorithm ALG is called ***weakly C-competitive*,** if there exists such a constant B that $\mathrm{ALG}(I) \leq C \cdot \mathrm{OPT}(I) + B$ is valid for each input I.

The ***weak competitive ratio of an algorithm*** is the smallest such C for which the algorithm is weakly C-competitive.

A further version of the competitive ratio is the asymptotic competitive ratio. For minimization problems the ***asymptotic competitive ratio*** of algorithm ALG ($R^{\infty}_{\mathrm{ALG}}$) can be defined as follows:

$$R^{n}_{\mathrm{ALG}} = \sup\{\frac{\mathrm{ALG}(I)}{\mathrm{OPT}(I)} \mid \mathrm{OPT}(I) = n\},$$

$$R^{\infty}_{\mathrm{ALG}} = \limsup_{n\to\infty} R^{n}_{\mathrm{ALG}}.$$

An algorithm is called ***asymptotically C-competitive*** if its asymptotic competitive ratio is at most C.

The main property of the asymptotic ratio that it considers the performance of the algorithm under the assumption that the size of the input tends to ∞. This means that this ratio is not effected by the behaviour of the algorithm on the small size inputs.

We defined the basic notions of the competitive analysis for minimization problems. Similar definitions can be given for maximization problems. Then algorithm ALG is called C-competitive, if $\mathrm{ALG}(I) \geq C \cdot \mathrm{OPT}(I)$ is valid for each input I, and the algorithm is weakly C-competitive if there exists such a constant B that $\mathrm{ALG}(I) \geq C \cdot \mathrm{OPT}(I) + B$ is valid for each input I. The asymptotic ratio for maximization problems can be given as follows:

$$R^{n}_{\mathrm{ALG}} = \inf\{\frac{\mathrm{ALG}(I)}{\mathrm{OPT}(I)} \mid \mathrm{OPT}(I) = n\},$$

$$R^{\infty}_{\mathrm{ALG}} = \liminf_{n\to\infty} R^{n}_{\mathrm{ALG}}.$$

Then the algorithm is called ***asymptotically C-competitive*** if its asymptotic ratio is at least C.

Many scientific papers consider ***randomized on-line algorithms*,** in this case the objective function value achieved by the algorithm is a random variable and the expected value of this variable is used in the definition of the competitive ratio. Since we consider only deterministic on-line algorithms in this chapter we do not detail the notions related to randomized on-line algorithms.

8.2. The k-server problem

One of the most known online problems is the online k-server problem.To give the definition of the general problem we need the notion of the metric space. A pair (M, d) (where M contains the points of the space, d is the distance function defined on the set $M \times M$) is called metric space if the following properties are valid:

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

In the k-server problem a metric space is given and there are k servers which can move in the space. The decision maker has to satisfy a list of request appearing at the points of the metric space by sending a server to the point where the request appears.

The problem is online which means that the requests arrive one by one and we must satisfy each request without any information about the further requests. The goal is to minimize the total distance travelled by the servers. The model and its special versions have many applications. In the remaining parts of the section the multiset which contains the points where the servers are is called the ***configuration of the servers***. We use multisets since different servers can be at the same points of the space.

The first important results for the k-server problem were achieved by Manasse McGeoch and Sleator. They developed the following algorithm called Balance, which we denote by BAL. During the procedure the servers are in different points. The algorithm maintains for each server the total distance travelled by the server. We denote by $s_1, \ldots, s_k$ the servers and also the points in the space where the servers are located. Denote by $D_1, \ldots, D_k$ the total distance travelled by the servers. Then after the arrival at point P of a request algorithm BAL uses the server i for which the value $D_i + d(s_i, P)$ is minimal. This means that the algorithm tries to balance the distances travelled by the servers. Therefore the algorithm maintains the $S = \{s_1, \ldots, s_k\}$ server configuration and the distances travelled by the servers which distances have the starting values $D_1 = \cdots = D_k = 0$. Then the behaviour of the algorithm on input $I = P_1, \ldots, P_n$ can be given by the following pseudocode:

BAL(I)

1 **for** $j \leftarrow 1$ **to** n
2 $\quad i \leftarrow \text{argmin}\{D_i + d(s_i, P_j)\}$
3 $\quad$ serve the request with server i
4 $\quad D_i \leftarrow D_i + d(s_i, P_j)$
5 $\quad s_i \leftarrow P_j$

Example 8.1 Consider the two dimension Euclidean space as the metric space. The points are two dimensional real vectors (x, y), and the distance between (a, b) and (c, d) is $\sqrt{(a-c)^2 + (b-d)^2}$. Suppose that there are two servers which are located at points $(0, 0)$ and $(1, 1)$ at the beginning. Thus at the beginning $D_1 = D_2 = 0$, $s_1 = (0, 0)$, $s_2 = (1, 1)$. Suppose that the first request appears at point $(1, 4)$. Then $D_1 + d((0, 0), (1, 4)) = \sqrt{17} > D_2 + d((1, 1), (1, 4)) = 3$, thus the second server is used to satisfy the request and after the movement of the server $D_1 = 0, D_2 = 3$, $s_1 = (0, 0)$,

$s_2 = (1,4)$. Suppose that the second request appears at point $(2,4)$, then $D_1 + d((0,0),(2,4)) = \sqrt{20} > D_2 + d((1,4),(2,4)) = 3 + 1 = 4$, thus again the second server is used, and after serving the request $D_1 = 0, D_2 = 4$, $s_1 = (0,0)$, $s_2 = (2,4)$. Suppose that the third request appears at point $(1,4)$, then $D_1 + d((0,0),(1,4)) = \sqrt{17} < D_2 + d((2,4),(1,4)) = 4 + 1 = 5$, thus the first server is used and after serving the request $D_1 = \sqrt{17}, D_2 = 4$, $s_1 = (1,4)$, $s_2 = (2,4)$.

The algorithm is efficient in the cases of some particular metric spaces as it is shown by the following statement. The references where the proof of the following theorem can be found are in the chapter notes at the end of the chapter.

Theorem 8.1 *Algorithm* BALANCE *is weakly k-competitive for the metric spaces containing $k+1$ points.*

The following statement shows that there is no online algorithm which is better than k-competitive for the general k-server problem.

Theorem 8.2 *There is no such metric space containing at least $k+1$ points where an on-line algorithm exists with smaller competitive ratio than k.*

Proof. Consider an arbitrary metric space containing at least $k+1$ points and an arbitrary on-line algorithm, denote the algorithm by ONL. Denote the points of the starting configuration of ONL by $P_1, P_2, \ldots, P_k$, and let P_{k+1} be an another point of the metric space. Consider the following long list of requests $I = Q_1, \ldots, Q_n$. The next request appears at the point among $P_1, P_2, \ldots, P_{k+1}$ where ONL has no server.

Calculate first the value ONL(I). The algorithm does not have server at point Q_{j+1} after serving Q_j, thus the request appeared at Q_j is served by the server located at point Q_{j+1}. Therefore the cost of serving Q_j is $d(Q_j, Q_{j+1})$, which yields that

$$\mathrm{ONL}(I) = \sum_{j=1}^{n} d(Q_j, Q_{j+1}) \ ,$$

where Q_{n+1} denote the point from which the server sent to serve Q_n. (This is the point where the $(n+1)$-th request would appear.) Now consider the cost OPT(I). Instead of calculating the optimal offline cost we define k different offline algorithms, and we use the average of the costs resulted by these algorithms. Since the cost of each offline algorithm is at least as much as the optimal offline cost thus the calculated average is an upper bound on the optimal offline cost.

We define the following k offline algorithms, denoted by $\mathrm{OFF}_1, \ldots, \mathrm{OFF}_k$. Suppose that the servers are in the points $P_1, P_2, \ldots, P_{j-1}, P_{j+1}, \ldots P_{k+1}$ in the starting configuration of OFF_j. We can move the servers into this starting configuration using an extra constant cost C_j.

The algorithms satisfy the requests as follows. If an algorithm OFF_j has a server at point Q_i, then none of the servers moves, otherwise the request is served by the server which is located at point Q_{i-1}. The algorithms are well-defined, if Q_i does not contain a server then each of the other points among $P_1, P_2, \ldots, P_{k+1}$ contains a servers, thus there is a server located at Q_{i-1}. Moreover $Q_1 = P_{k+1}$, thus at the beginning each algorithm has a server at the requested point.

We show that the servers of algorithms $\mathrm{OFF}_1, \ldots, \mathrm{OFF}_k$ are always in different configurations. At the beginning this property is valid by the definition of the algorithms. Consider now the step where a request is served. Call the algorithms stable which do not move a server for serving the request, and the other algorithms unstable. The server configurations of the stable algorithms remain unchanged, so these configurations remain different from each other. Any unstable algorithm moves a server from the point Q_{i-1}. This point is the place of the last request thus the stable algorithms have server in it. Therefore, an unstable algorithm and a stable algorithm cannot have the same configuration after serving the request. Furthermore, any of the unstable algorithms moves a server from Q_{i-1} to Q_i, thus the server configurations of the unstable algorithms remain different from each other.

So at the arrival of the request at point Q_i the servers of the algorithms are in different configuration. On the other hand each configuration has a server at point Q_{i-1}, therefore there is only one configuration where no server located at point Q_i. Consequently the cost of serving Q_i is $d(Q_{i-1}, Q_i)$ for one of the algorithms and 0 for the other algorithms.

Therefore

$$\sum_{j=1}^{k} \mathrm{OFF}_j(I) = C + \sum_{i=2}^{n} d(Q_i, Q_{i-1}) \ ,$$

where $C = \sum_{j=1}^{k} C_j$ is an absolute constant which is independent from the input (this is the cost of moving the servers to the starting configuration of the defined algorithms).

On the other hand the optimal offline cost cannot be larger than the cost of any of the above defined algorithms, thus $k \cdot \mathrm{OPT}(I) \leq \sum_{j=1}^{k} \mathrm{OFF}_j(I)$. This yields that

$$k \cdot \mathrm{OPT}(I) \leq C + \sum_{i=2}^{n} d(Q_i, Q_{i-1}) \leq C + \mathrm{ONL}(I),$$

which inequality shows that the weak competitive ratio of ONL cannot be smaller than k, since the value $\mathrm{OPT}(I)$ can be arbitrarily large as the length of the input is increasing. ■

Many researchers started to work on the problem and some interesting results appeared during the next few years. For the general case the first constant-competitive algorithm ($O(2^k)$-competitive) was developed by Fiat, Rabani and Ravid. Then the researchers could not significantly decrease the gap between the lower and upper bounds for a long time. Later Koutsoupias and Papadimitriou could analyze an algorithm based on the work function technique and they could prove that the algorithm is $(2k-1)$-competitive. They could not determine the competitive ratio of the algorithm, but it is a widely believed hypothesis that the algorithm is k-competitive. To determine the competitive ratio of the algorithm, or to develop a k-competitive algorithm are still among the most important open questions in the area of on-line algorithms. We present the work function algorithm below.

Denote by A_0 the starting configuration of the on-line servers. Then after the t-th request the ***work function*** value belonging to the multiset X is the minimal cost which can be used to serve the first t requests starting at the configuration A_0 and ending at the configuration X. This value is denoted by $w_t(X)$. The Work Function algorithm is based on the above defined work function. Suppose that A_{t-1} is the server configuration before the arrival of the t-th request, and denote the place of the t-th request by R_t. Then the Work Function algorithm uses the server s to serve the request for which the value $w_{t-1}(A_{t-1} \setminus \{P\} \cup \{R_t\}) + d(P, R_t)$ is minimal, where P denotes the point where the server is actually located.

Example 8.2 Consider the metric space which contains three points A, B and C and the distances are $d(A,B) = 1, d(B,C) = 2, d(A,C) = 3$. Suppose that we have two servers and the starting configuration is $\{A,B\}$. Then the starting work function values are $w_0(\{A,A\}) = 1$, $w_0(\{A,B\}) = 0$, $w_0(\{A,C\}) = 2$, $w_0(\{B,B\}) = 1$, $w_0(\{B,C\}) = 3$, $w_0(\{C,C\}) = 4$. Suppose that the first request appears at point C. Then $w_0(\{A,B\} \setminus \{A\} \cup \{C\}) + d(A,C) = 3 + 3 = 6$ and $w_0(\{A,B\} \setminus \{B\} \cup \{C\}) + d(B,C) = 2 + 2 = 4$, thus algorithm WORK FUNCTION uses the server from point B to serve the request.

The following statement is valid for the algorithm.

Theorem 8.3 *The* WORK FUNCTION *algorithm is weakly* $2k-1$*-competitive.*

Beside the general problem many particular cases were investigated. If the distance of any pair of points is 1, then we obtain the on-line paging problem as the special case. An another well investigated metric space is the line. The points of the line are considered as real numbers, and the distance of points a and b is $|a-b|$. In this special case a k-competitive algorithm was developed by Chrobak and Larmore, the algorithm is called DOUBLE-COVERAGE. A request at point P is served by the server s which is closest to P. Moreover, if there are also servers on the opposite side of P then the closest server among them moves distance $d(s,P)$ into the direction of P. In the following parts we denote the DOUBLE-COVERAGE algorithm by DC. The input of the algorithm is the list of requests which is a list of points (real numbers) denoted by $I = P_1, \ldots, P_n$ and the starting configuration of the servers denoted by $S = (s_1, \ldots, s_k)$ which contains also points (real numbers). The algorithm can be defined by the following pseudocode:

DC(I,S)

```
1  for j ← 1 to n
2      do i ← argmin_l d(P_j, s_l)
3  if s_i = max_l s_l or s_i = max_l s_l
4    then the request is served by the i-th server
5           s_i ← P_j
6  else if s_i ≤ P_j
7          then m ← argmin_{l:s_l>P_j} d(s_l, P_j)
8               the request is served by the i-th server
9               s_m ← s_m − d(s_i, P_j)
10              s_i ← P_j
11 else if s_i ≥ P_j
12         then n ← argmin_{l:s_l<P_j} d(s_l, P_j)
13              the request is served by the i-th server
14              s_n ← s_n + d(s_i, P_j)
15              s_i ← P_j
```

Example 8.3 Suppose that there are three servers s_1, s_2, s_3 located at the points $0, 1, 2$. If the next request appears at point 4 then DC uses the closest server s_3 to serve the request. The locations of the other servers remain unchanged, the cost of serving the request is 2 and the servers are at the points $0, 1, 4$. If the next request appears at point 2 then DC uses the closest server s_2 to serve the request, but there are servers on the opposite side of the request thus s_3 also moves distance 1 into the direction of 2. Therefore the cost of serving the request is 2 and the servers will be at the points $0, 2, 3$.

The following statement which can be proved by the potential function technique is valid for algorithm DC. This technique is often used in the analysis of on-line algorithms.

Theorem 8.4 *If the metric space is the line then algorithm* DC *is weakly k-competitive.*

Proof. Consider an arbitrary sequence of requests denote this input by I. During the analysis of the procedure we suppose that one offline optimal algorithm and DC are running parallel on the input. We also suppose that each request is served first by the offline algorithm and then by the on-line algorithm. The servers of the on-line algorithm and also the positions of the servers (which are real numbers) are denoted by $s_1, \ldots, s_k$, and the servers of the optimal offline algorithm and also the positions of the servers are denoted by $x_1, \ldots, x_k$. We suppose that for the positions $s_1 \leq s_2 \leq \cdots \leq s_k$ and $x_1 \leq x_2 \leq \cdots \leq x_k$ are always valid, this can be achieved by changing the notations of the servers.

We prove the theorem by the potential function technique. The potential function assigns a value to the actual positions of the servers, the on-line and offline costs are compared using the changes of the potential function. Define the following potential function

$$\Phi = k \sum_{i=1}^{k} |x_i - s_i| + \sum_{i<j} (s_j - s_i) \ .$$

We show that the following statements are valid for the potential function.

- While OPT serves a request the increase of the potential function is not more than k times the distance moved by the servers of OPT.
- While DC serves a request, the decrease of Φ is at least as much as the cost of serving the request.

If the above properties are valid then one can prove easily the theorem. In this case $\Phi_f - \Phi_0 \leq k \cdot \mathrm{OPT}(I) - \mathrm{DC}(I)$, where Φ_f and Φ_0 are the final and the starting values of the potential function. Furthermore Φ is nonnegative so we obtain that $\mathrm{DC}(I) \leq k\mathrm{OPT}(I) + \Phi_0$, which yields by the definition that the algorithms is weakly k-competitive.

Now we prove the properties of the potential function.

First consider the case when one of the offline servers moves distance d. Then the first part of the potential function increases at most kd. The second part does not change thus we proved the first property of the potential function.

Consider the servers of DC. Suppose that the request appears at point P. Since the request is first served by OPT, $x_j = P$ for some j. Distinguish the following two cases depending on the positions of the on-line servers.

Suppose first that the on-line servers are on the same side of P. We can assume that the positions of the servers are not smaller than P, the other case is completely similar. Then s_1 is the closest server and DC sends s_1 to P, the other on-line servers do not move. Therefore the cost of DC is $d(s_1, P)$. In the first sum of the potential function only $|x_1 - s_1|$ is changing that decreases $d(s_1, P)$, thus the first part decreases $kd(s_1, P)$. The second sum is increasing the increase is $(k-1)d(s_1, P)$, thus the value of Φ decreases $d(s_1, P)$.

Consider the second case. Then there are servers on both sides of P, suppose that the closest servers are s_i and s_{i+1}. We assume that s_i is closer to P, the other case is completely similar. Then the cost of DC is $2d(s_i, P)$. Consider the first sum of the potential function. The i-th and the $i+1$-th part are changing. Since $x_j = P$ for some j thus one of the i-th and

the $i+1$-th parts decreases $d(s_i, P)$, the increase of the other one is at most $d(s_i, P)$ thus the first sum does not increase. The change of the second sum of Φ is

$$d(s_i, P)(-(k-i)+(i-1)-(i)+(k-(i+1))) = -2d(s_i, P) .$$

Therefore we proved that the second property of the potential function is also valid in this case.

Since we investigated all of the possible cases we proved the properties of the potential function, and the statement of the theorem follows. ■

Exercises

8.2-1 Suppose that (M, d) is a metric space. Prove that (M, q) is also a metric space where $q(x, y) = \min\{1, d(x, y)\}$.

8.2-2 Consider the greedy algorithm which serves each request by the server which is closest to the place of the request. Prove that the algorithm is not constant competitive for the line.

8.2-3 Prove that for arbitrary k-element multisets X and Z and for arbitrary t the inequality $w_t(Z) \leq w_t(X) + d(X, Z)$ is valid, where $d(X, Z)$ is the cost of the minimal matching of X and Z, (the minimal cost which can be used to move the servers from configuration X to configuration Z).

8.2-4 Consider the line as a metric space. Suppose that the servers of the on-line algorithm are at the points $2, 4, 5, 7$, and the servers of the offline algorithm are at the points $1, 3, 6, 9$. Calculate the value of the potential function used in the proof of theorem 8.4. How this potential function is changed if the on-line server moves from point 7 to point 8?

8.3. Models related to computer networks

The theory of computer networks became one of the most significant areas of the computer science. In the planning of computer networks many optimization problems arise and most of these problems are actually on-line, since neither the traffic nor the changes in the topology of a computer network cannot be precisely predicted. Recently some researchers working at the area of on-line algorithms defined some on-line mathematical models for problems related to computer networks. In this section we consider this area we present three problems and show the basic results. First the data acknowledgement problem is considered, then we present the web caching problem, and the section is closed by the on-line routing problem.

8.3.1. The data acknowledgement problem

In the communication of a computer network the information is sent by packets. If the communication channel is not completely safe then the arrival of the packets are acknowledged. In the data acknowledgement problem we try to determine the time of sending acknowledgements. One acknowledgement can acknowledge many packets but waiting for long time can cause the resending of the packets and that results the congestion of the network. On the other hand sending immediately an acknowledgement about the arrival of each packet would cause again the congestion of the network. The first optimization model for determining

the sending time of the acknowledgements was developed by Dooly, Goldman and Scott in 1998. Below we present the developed model and some of the basic results.

In the mathematical model of the data acknowledgement problem the input is the list of the arrival times $a_1, \ldots, a_n$ of the packets. The decision maker has to determine when to send acknowledgements, these times are denoted by $t_1, \ldots, t_k$. In the optimization model the cost function is:

$$k + \sum_{j=1}^{k} \nu_j \ ,$$

where k is the number of the sent acknowledgements and $\nu_j = \sum_{t_{j-1} < a_i \leq t_j} (t_j - a_i)$ is the total latency collected by the j-th acknowledgement. We consider the on-line problem which means that at time t the decision maker only knows the arrival times of the packets already arrived and has no information about the further packets. We denote the set of the unacknowledged packets at the arrival time a_i by σ_i.

For the solution of the problem the class of the alarming algorithms was developed. An ***alarming algorithm*** works as follows. At the arrival time a_j an alarm is set for time $a_j + e_j$. If no packet arrives before time $a_j + e_j$, then an acknowledgement is sent at time $a_j + e_j$ which acknowledges all of the unacknowledged packets. Otherwise at the arrival of the next packet at time a_{j+1} the alarm is reset for time $a_{j+1} + e_{j+1}$. Below we analyze in details an algorithm from this class. This algorithm sets the alarm to collect total latency 1 by the acknowledgement. The algorithm is called Alarm. We obtain the above defined rule from the general definition using the solution of the following equation as value e_j

$$1 = |\sigma_j| e_j + \sum_{a_i \in \sigma_j} (a_j - a_i) \ .$$

Example 8.4 Consider the following example. The first packet arrives at time 0 ($a_1 = 0$), then Alarm sets an alarm with the value $e_1 = 1$ for time 1. Suppose that the next arrival time is $a_2 = 1/2$. This arrival is before the alarm time thus the first packet is not acknowledged yet and we reset the alarm with the value $e_2 = (1-1/2)/2 = 1/4$ for time $1/2+1/4$. Suppose that the next arrival time is $a_3 = 5/8$. This arrival is before the alarm time thus the first two packets are not acknowledged yet and we reset the alarm with value $e_3 = (1 - 5/8 - 1/8)/3 = 1/12$ for time $5/8 + 1/12$. Suppose that the next arrival time is $a_4 = 1$. Then no packet arrived before the alarm time $5/8 + 1/12$, thus at that time the first three packets were acknowledged and the alarm is set for the new packet with the value $e_4 = 1$ for time 2.

The following theorem is valid for the competitive ratio of the algorithm.

Theorem 8.5 *Algorithm* Alarm *is 2-competitive.*

Proof. Suppose that algorithm Alarm sends k acknowledgements. These acknowledgements divide the time into k time intervals. The cost of the algorithm is $2k$, since k is the cost of the acknowledgements, and the alarm is set to have total latency 1 for each acknowledgement.

Suppose that the optimal offline algorithm sends k^* acknowledgements. If $k^* \geq k$, then $\mathrm{OPT}(I) \geq k = \mathrm{Alarm}(I)/2$ is obviously valid, thus we have that the algorithm is 2-competitive. If $k^* < k$, then at least $k - k^*$ among the time intervals defined by the acknowledgements of algorithm Alarm do not contain any of the offline acknowledgements. This

yields that the offline total latency is at most $k - k^*$, thus we obtain that $\mathrm{OPT}(I) \geq k$ which inequality proves that ALARM is 2-competitive. ■

As the following theorem shows algorithm ALARM has the smallest possible competitive ratio.

Theorem 8.6 *There is not such on-line algorithm for the data acknowledgement problem which has smaller competitive ratio than* 2.

Proof. Consider an arbitrary on-line algorithm denote it by ONL. Analyze the following input. Consider a long sequence of packets where the packets always arrive immediately after the time when ONL sends an acknowledgement. Then the on-line cost of a sequence containing $2n$ packets is $\mathrm{ONL}(I_{2n}) = 2n + t_{2n}$, since the cost resulted from the acknowledgements is $2n$, and the latency of the i-th acknowledgement is $t_i - t_{i-1}$, where the value $t_0 = 0$ is used.

Consider the following two on-line algorithms. ODD sends the acknowledgements after the odd numbered packets and EV sends the acknowledgements after the even numbered packets.

Then the costs achieved by these algorithms are

$$\mathrm{EV}(I_{2n}) = n + \sum_{i=0}^{n-1} (t_{2i+1} - t_{2i}) \ ,$$

and

$$\mathrm{ODD} = n + 1 + \sum_{i=1}^{n} (t_{2i} - t_{2i-1}) \ .$$

Therefore $\mathrm{EV}(I_{2n}) + \mathrm{ODD}(I_{2n}) = \mathrm{ONL}(I_{2n}) + 1$. On the other hand none of the costs achieved by ODD and EV is greater than the optimal offline cost thus $\mathrm{OPT}(I_{2n}) \leq \min\{\mathrm{EV}(I_{2n}), \mathrm{ODD}(I_{2n})\}$, which yields that $\mathrm{ONL}(I_{2n})/\mathrm{OPT}(I_{2n}) \geq 2 - 1/\mathrm{OPT}(I_{2n})$. By this inequality it follows that the competitive ratio of ONL is not smaller than 2, because using a sufficiently long sequence of packets the value $\mathrm{OPT}(I_{2n})$ can be arbitrarily large. ■

8.3.2. The file caching problem

The file caching problem is a generalization of the caching problem presented in the memory management chapter. The world-wide-web browsers are using caches to store some files. This makes it possible to use the stored files if a user wants to see some web-page many times during a short time interval. If the cache becomes full then some files must be eliminated to make place for the new file. The file caching problem models this scenario, the goal is to find good strategies for determining which files should be eliminated. The difference to the standard paging problem is that the files have size and retrieval cost (the problem is reduced to the paging if each size and each retrieval cost are 1). So the following mathematical model describes the problem.

There is a given cache of size k, the input is a sequence of pages. Each page p has a ***size*** denoted by $s(p)$ and a ***retrieval cost*** denoted by $c(p)$. The pages arrive from a list one by one and after the arrival of a page the algorithm has to place it into the cache. If the page is not contained in the cache and there is not enough place to put it into the cache then the algorithm has to delete some pages from the cache to make enough place for the requested

page. If the required page is in the cache then the cost of serving the request is 0 otherwise the cost is $c(p)$. The objective is to minimize the total cost. The problem is on-line which means that for the decisions (which pages should be deleted from the cache) only the earlier pages and decisions can be used, the algorithm has no information about the further pages. We assume that the size of the cache and also the sizes of the pages are positive integers.

For the solution of the problem and for its particular cases many algorithms were developed. Here we present algorithm LANDLORD which algorithm was developed by Young.

The algorithm maintains a credit value $0 \leq cr(f) \leq c(f)$ for each page f which is contained in the actual cache. In the following part of the section the set of the pages in the actual cache of LANDLORD is denoted by LA. If LANDLORD has to retrieve a page g then the following steps are performed.

LANDLORD(LA, g)

1 **if** g is not contained in LA
2 **then while** there is not enough place for g
3 $\Delta \leftarrow \min_{f \in LA} cr(f)/s(f)$
4 for each $f \in LA$ let $cr(f) \leftarrow cr(f) - \Delta \cdot s(f)$
5 evict some pages with $cr(f) = 0$
6 place g into the cache LA and let $cr(g) \leftarrow c(g)$
7 **else** reset $cr(g)$ to any value between $cr(g)$ and $c(g)$

Example 8.5 Suppose that $k = 10$ and LA contains the following three pages: g_1 with $s(g_1) = 2, cr(g_1) = 1$, g_2 with $s(g_2) = 4, cr(g_2) = 3$ and g_3 with $s(g_3) = 3, cr(g_3) = 3$. Suppose that the next requested page is g_4, with the parameters $s(g_4) = 4, c(g_4) = 4$. Then there is not enough place for it in the cache, so some pages must be evicted. LANDLORD determines the value $\Delta = 1/2$ and changes the credits as follows: $cr(g_1) = 0, cr(g_2) = 1, cr(g_3) = 3/2$, thus g_1 is evicted from the cache LA. Still there is not enough place for g_4 in the cache. The new Δ value is $\Delta = 1/4$ and the new credits are: $cr(g_2) = 0, cr(g_3) = 3/4$, thus g_2 is evicted from the cache. Then there is enough place for g_4, thus it is placed into the cache LA with the credit value $cr(g_4) = 4$.

LANDLORD is weakly k-competitive, but a stronger statement is also true. For the web caching problem an on-line algorithm ALG is called (C, k, h)-competitive, if there exists such an absolute constant B, that $\mathrm{ALG}_k(I) \leq C \cdot \mathrm{OPT}_h(I) + B$ is valid for each input, where $\mathrm{ALG}_k(I)$ is the cost of ALG using a cache of size k and $\mathrm{OPT}_h(I)$ is the optimal offline cost using a cache of size h. The following statement is true for algorithm LANDLORD.

Theorem 8.7 *If $h \leq k$, then algorithm* LANDLORD *is $(k/(k-h+1), k, h)$-competitive.*

Proof. Consider an arbitrary input sequence of pages, denote the input by I. We use the potential function technique. During the analysis of the procedure we suppose that an offline optimal algorithm with cache size h and LANDLORD with cache size k are running parallel on the input. We also suppose that each page is placed first by the offline algorithm into the offline cache and then it is placed by the on-line algorithm into LA. We denote the set of the pages contained in the actual cache of the optimal offline algorithm by OPT. Consider the following potential function

$$\Phi = (h-1)\sum_{f\in LA} cr(f) + k\sum_{f\in OPT}(c(f)-cr(f)) \ .$$

Investigate the changes of the potential function during the retrievals of a page g.

- OPT places g into its cache.

  Then OPT has cost $c(g)$. In the potential function only the second part may change. On the other hand $cr(g) \geq 0$, thus the increase of the potential function is at most $k \cdot c(g)$.

- Landlord decreases the credit value for each $f \in LA$.

  In this case for each $f \in LA$ the decrease of $cr(f)$ is $\Delta \cdot s(f)$, thus the decrease of Φ is

  $$\Delta((h-1)s(LA) - ks(OPT \cap LA)) \ ,$$

  where $s(LA)$ and $s(OPT \cap LA)$ denote the total size of the pages contained in sets LA and $OPT \cap LA$ respectively. At the time when this step is performed OPT have already placed the page g into its cache OPT, but the page is not contained in the cache LA. Therefore $s(OPT \cap LA) \leq h - s(g)$. On the other hand this step is performed if there is not enough place for the page in LA thus $s(LA) > k - s(g)$, which yields $s(LA) \geq k - s(g) + 1$ because the sizes are positive integers. Therefore we obtain that the decrease of Φ is at least

  $$\Delta\big((h-1)(k-s(g)+1) - k(h-s(g))\big) \ .$$

  Since $s(g) \geq 1$ and $k \geq h$, this decrease is at least $\Delta((h-1)(k-1+1) - k(h-1)) = 0$.

- Landlord evicts a page f from cache LA.

  Since Landlord evicts only pages having credit 0, thus during this step Φ remains unchanged.

- Landlord places page g into the cache LA and sets the value $cr(g) = c(g)$.

  Then the cost of Landlord is $c(g)$. On the other hand g was not contained in the cache LA before the performance of this step, thus $cr(g) = 0$ was valid. Furthermore first OPT places the page into its cache thus $g \in OPT$ is also valid. Therefore the decrease of Φ is $-(h-1)c(g) + kc(g) = (k-h+1)c(g)$.

- Landlord resets for a page $g \in HA$ the credit to a value between $cr(g)$ and $c(g)$.

  In this case $g \in OPT$ is valid, since OPT places first the page g into its cache. The value $cr(g)$ is not decreased and $k > h-1$, thus Φ can not increase during this step.

We have investigated the possible steps of the algorithms and we proved the following properties of the potential function.

- If OPT places a page into its cache, then the increase of the potential function is at most k times more than the cost of OPT.
- If Landlord places a page into its cache, then the decrease of Φ is $(k-h+1)$ times more than the cost of Landlord.
- During the other steps Φ does not increase.

By the above properties we obtain that $\Phi_f - \Phi_0 \leq k \cdot \mathrm{OPT}_h(I) - (k-h+1) \cdot \textsc{Landlord}_k(I)$, where Φ_0 and Φ_f are the starting and final values of the potential function. The potential function is nonnegative, thus we obtain that $(k - h + 1) \cdot \textsc{Landlord}_k(I) \leq k \cdot \mathrm{OPT}_h(I) + \Phi_0$, which proves that LANDLORD is $(k/(k - h + 1), k, h)$-competitive. ■

8.3.3. On-line routing

In computer networks the congestion of the communication channels decreases the speed of the communication and may cause loss of information. Thus congestion control is one of the most important problems in the area of computer networks. A related important problem is the routing of the communication where we have to determine the path in the network for the messages. Since we have no information about the further traffic of the network thus routing is an on-line problem. Here we present two on-line optimization models for the routing problem.

The mathematical model

The network is given by a graph, each edge e has a maximal available bandwidth denoted by $u(e)$, the number of edges is denoted by m. The input is a sequence of requests, where the j-th request is given by a vector $(s_j, t_j, r_j, d_j, b_j)$, and to satisfy the request bandwidth r_j must be reserved on a path from s_j to t_j for time duration d_j, the benefit of serving a request is b_j. In the followings we assume the assumption $d_j = \infty$, and we omit the value of d_j from the requests. The problem is on-line which means that after the arrival of a request the algorithm has to make the decisions without any information about the further requests. We consider the following two models.

Load balancing model: In this model all requests must be satisfied. The objective is to minimize the maximum of the overload of the edges. The overload is the ratio of the sum of the bandwidths reserved on the edge and the available bandwidth. Since each request is served thus the benefit is not significant in this model.

Throughput model: In this model the decision maker is allowed to reject some requests. The sum of the bandwidths reserved on an edge can not be more than the available bandwidth. The goal is to maximize the sum of the benefits of the accepted requests. We investigate this model in details. It is important to note that this is a maximization problem thus we use the notion of competitiveness in the form defined for maximization problems.

Below we define the exponential algorithm. We need the following notations to define and analyze the algorithm. Denote P_i the path which is assigned to the accepted request i. Let A denote the set of requests accepted by the on-line algorithm. Then $l_e(j) = \sum_{i \in A, i<j, e \in P_i} r_i/u(e)$ is the ratio of the reserved bandwidth and the available bandwidth on e before the arrival of request j.

The basic idea of the exponential algorithm is the following. The algorithm assigns a cost which is exponential in $l_e(j)$ to each e and chooses the path which has the minimal cost. Below we define and analyze the exponential algorithm for the throughput model. Let μ be a constant which depends on the parameters of the problem, its value will be given later. Let $c_e(j) = \mu^{l_e(j)}$, for each request j and edge e. Then the exponential algorithm performs the following steps after the arrival of a request (s_j, t_j, r_j, b_j).

EXP(s_j, t_j, r_j, b_j)

1 let U_j be the set of the paths (s_j, t_j)
2 $P_j \leftarrow \text{argmin}_{P \in U_j}\{\sum_{e \in P} \frac{r_j}{u(e)} c_e(j)\}$
3 **if** $C(P_j) = \sum_{e \in P_j} \frac{r_j}{u(e)} c_e(j) \leq 2mb_j$
4 **then** reserve the bandwidth r_j on path P_j
5 **else** reject the request

Remark. If we modify this algorithm to accept each request then we obtain an exponential algorithm for the load balancing model.

Example 8.6 Consider the network which contains the vertices A, B, C, D and the edges (A, B), (B, D), (A, C), (C, D), where the available bandwidths of the edges are $u(A, B) = 1$, $u(B, D) = 3/2$, $u(A, C) = 2$, $u(C, D) = 3/2$. Suppose that $\mu = 10$ and that the reserved bandwidths are: $3/4$ on the path (A, B, D), $5/4$ on the path (A, C, D), $1/2$ on the path (B, D), $1/2$ on the path (A, C). The next request j is to reserve bandwidth $1/8$ on some path between A and D. Then the values $l_e(j)$ are: $l_{(A,B)}(j) = (3/4) : 1 = 3/4$, $l_{(B,D)}(j) = (3/4 + 1/2) : (3/2) = 5/6$, $l_{(A,C)}(j) = (5/4 + 1/2) : 2 = 7/8$, $l_{(C,D)}(j) = (5/4) : (3/2) = 5/6$. There are two paths between A and D, the costs are:

$$C(A, B, D) = 1/8 \cdot 10^{3/4} + 1/12 \cdot 10^{5/6} = 1.269 \ ,$$

$$C(A, C, D) = 1/16 \cdot 10^{7/8} + 1/12 \cdot 10^{5/6} = 1.035 \ .$$

The minimal cost belongs to the path (A, C, D). Therefore, if $2mb_j = 8b_j \geq 1,035$, then the request is accepted and the bandwidth is reserved on the path (A, C, D). Otherwise the request is rejected.

To analyze the algorithm consider an arbitrary input sequence I. Denote A the set of the requests accepted by EXP, and denote A^* the set of the requests which are accepted by OPT and rejected by EXP. Furthermore denote P_j^* the path reserved by OPT for each request j accepted by OPT. Define the value $l_e(v) = \sum_{i \in A, e \in P_i} r_i/u(e)$ for each e, this value gives the ratio of the reserved bandwidth and the available bandwidth for e at the end of the on-line algorithm.

Let $\mu = 4mPB$, where B is an upper bound on the benefits and for each request and each edge the inequality

$$\frac{1}{P} \leq \frac{r(j)}{u(e)} \leq \frac{1}{\lg \mu}$$

is valid. Then the following statements hold.

Lemma 8.8 *The solution given by algorithm* EXP *is feasible, the sum of the reserved bandwidths is not more than the available bandwidth for each edge.*

Proof. We prove the statement by contradiction. Suppose that there is an edge f where the available bandwidth is violated. Let j be the first such accepted request which violates the available bandwidth on f.

The inequality $r_j/u(f) \leq 1/\lg \mu$ is valid for j and f (it is valid for all edges and requests). Furthermore after the acceptance of request j the sum of the bandwidths is greater than the available bandwidth on edge f, thus we obtain that $l_f(j) > 1 - 1/\lg \mu$. On the other hand this yields that the inequality

$$C(P_j) = \sum_{e \in P_j} \frac{r_j}{u(e)} c_e(j) \geq \frac{r_j}{u(f)} c_f(j) > \frac{r_j}{u(f)} \mu^{1-1/\lg \mu}$$

holds for the value $C(P_j)$ used in algorithm EXP. Using the assumption on P we obtain that $\frac{r_j}{u(e)} \geq \frac{1}{P}$, and $\mu^{1-1/\lg m} = \mu/2$, thus by the above inequality we obtain that

$$C(P) > \frac{1}{P} \frac{\mu}{2} = 2mB \ .$$

On the other hand this inequality is a contradiction since EXP would reject the request. Therefore we obtained a contradiction thus we proved the statement of the lemma. ■

Lemma 8.9 *For the solution given by* OPT *the following inequality holds:*

$$\sum_{j \in A^*} b_j \leq \frac{1}{2m} \sum_{e \in E} c_e(v) \ .$$

Proof. Since EXP rejected j for each $j \in A^*$, thus $b_j < \frac{1}{2m} \sum_{e \in P_j^*} \frac{r_j}{u(e)} c_e(j)$ for each $j \in A^*$, because this inequality is valid for all paths between s_j and t_j. Therefore

$$\sum_{j \in A^*} b_j < \frac{1}{2m} \sum_{j \in A^*} \sum_{e \in P_j^*} \frac{r_j}{u(e)} c_e(j) \ .$$

On the other hand $c_e(j) \leq c_e(v)$ holds for each e, thus we obtain that

$$\sum_{j \in A^*} b_j < \frac{1}{2m} \sum_{e \in E} c_e(v) \Big(\sum_{j \in A^* : e \in P_j^*} \frac{r_j}{u(e)} \Big) \ .$$

The sum of the bandwidths reserved by OPT is at most the available bandwidth $u(e)$ for each e, thus $\sum_{j \in A^* : e \in P^*(j)} \frac{r_j}{u(e)} \leq 1$.

Consequently

$$\sum_{j \in A^*} b_j \leq \frac{1}{2m} \sum_{e \in E} c_e(v)$$

which inequality is the one which we wanted to prove. ■

Lemma 8.10 *For the solution given by algorithm* EXP *the following inequality holds*

$$\frac{1}{2m} \sum_{e \in E} c_e(v) \leq (1 + \lg \mu) \sum_{j \in A} b_j \ .$$

Proof. To prove the lemma it is enough to show that the inequality $\sum_{e \in P_j} (c_e(j+1) - c_e(j)) \leq 2mb_j \log_2 \mu$ is valid for each request $j \in A$. On the other hand

$$c_e(j+1) - c_e(j) = \mu^{l_e(j) + \frac{r_j}{u(e)}} - \mu^{l_e(j)} = \mu^{l_e(j)} (2^{\log_2 \mu \frac{r_j}{u(e)}} - 1) \ .$$

Since $2^x - 1 < x$, if $0 \leq x \leq 1$, and by the assumptions $0 \leq \log_2 \mu \frac{r_j}{u(e)} \leq 1$, thus we obtain that

$$c_e(j+1) - c_e(j) \leq \mu^{l_e(j)} \log_2 \mu \frac{r_j}{u(e)}.$$

Summarizing the bounds given above we obtain that

$$\sum_{e\in P_j}(c_e(j+1)-c_e(j)) \le \log_2\mu \sum_{e\in P_j}\mu^{l_e(j)}\frac{r_j}{u(e)} = \log_2\mu\cdot C(P_j)\ .$$

Since EXP accepts the requests with the property $C(P_j) \le 2mb_j$, thus the above inequality proves the required statement. ■

By the above lemmas we can prove the following theorem.

Theorem 8.11 *Algorithm* EXP *is* $1/O(\lg\mu)$*-competitive, if* $\mu = 4mPB$*, where* B *is an upper bound on the benefits, and for all edges and requests*

$$\frac{1}{P} \le \frac{r(j)}{u(e)} \le \frac{1}{\lg\mu}.$$

Proof. By lemma 8.8 it follows that the algorithm results in a feasible solution where the available bandwidths are not violated. Using the notations defined before the lemmas we obtain that the benefit of algorithm EXP on the input I is $\mathrm{EXP}(I) = \sum_{j\in A} b_j$, and the benefit of OPT is at most $\sum_{j\in A\cup A^*} b_j$. Therefore by lemma 8.9 and lemma 8.10 it follows that

$$\mathrm{OPT}(I) \le \sum_{j\in A\cup A^*} b_j \le (2+\log_2\mu)\sum_{j\in A} b_j \le (2+\log_2\mu)\mathrm{EXP}(I)\ ,$$

which inequality proves the theorem. ■

Exercises

8.3-1 Consider the modified version of the data acknowledgement problem with the objective function $k + \sum_{j=1}^{k}\mu_j$, where k is the number of acknowledgements and $\mu_j = \max_{t_{j-1}<a_i\le t_j}\{t_j - a_i\}$ is the maximal latency of the j-th acknowledgement. Prove that algorithm Alarm is also 2-competitive in this modified model.

8.3-2 Represent the special case of the web caching problem, where $s(g) = c(g) = 1$ for each page g as a special case of the k-server problem. Define the metric space which can be used.

8.3-3 In the web caching problem the cache LA of size 8 contains three pages a, b, c with the following sizes and credits: $s(a) = 3, s(b) = 2, s(c) = 3, cr(a) = 2, cr(b) = 1/2, cr(c) = 2$. We want to retrieve a page d of size 3, and retrieval cost 4. The optimal offline algorithm OPT with cache of size 6 already placed the page into its cache, its cache contains the pages d and c. Which pages are evicted by Landlord to place d? What kind of changes the potential function defined in the proof of theorem 8.7 has?

8.3-4 Prove that if in the throughput model no bounds are given for the ratios $r(j)/u(e)$ then there is not constant-competitive on-line algorithm.

8.4. On-line bin packing models

In this section we consider the on-line bin packing problem and its multidimensional generalizations. First we present the classical on-line bin packing problem and some fundamental results of the area. Then we define the multidimensional generalizations and present some details from the area of on-line strip packing.

8.4.1. On-line bin packing

In the bin packing problem the input is a list of items, where the i-th item is given by its size $a_i \in (0, 1]$. The goal is to pack the items into unit size bins and minimize the number of the used bins. In a more formal way we can say that we have to divide the items into groups where each group has the property that the total size of the items is at most 1 and the goal is to minimize the number of groups. This problem also appears in the area of memory management.

In this section we investigate the on-line problem which means that the decision maker has to make decisions about the packing of the i-th item based on the values $a_1, \ldots, a_i$ without any information about the further items.

NF algorithm, bounded space algorithms

First we consider the model where the number of the open bins is limited. The k-bounded space model means that if the number of open bins reaches the bound k then the algorithm can open a new bin only after closing some of the bins, and the closed bins cannot be used for packing further items. If only one bin can be open then the natural algorithm packs the item into the open bin if it fits otherwise it closes the bin, opens a new one and put the item into it. We call this algorithm NF (Next Fit) algorithm. We do not present the pseudocode of the algorithm it can be found in this book in the chapter about memory management. The asymptotic competitive ratio of algorithm NF is determined by the following theorem.

Theorem 8.12 *The asymptotic competitive ratio of* NF *is 2.*

Proof. Consider an arbitrary sequence of items, denote it by σ. Denote n the number of bins used by OPT and denote m the number of bins used by NF. Furthermore let S_i, $i = 1, \ldots, m$ the total size of the items packed into the i-th bin by NF.

Then $S_i + S_{i+1} > 1$, since in the opposite case the first item of the $(i + 1)$-th bin fits into the i-th bin which contradicts to the definition of the algorithm. Therefore the total size of the items is more than $\lfloor m/2 \rfloor$.

On the other hand the optimal offline algorithm cannot put items with total size more than 1 into the same bin, thus we obtain that $n > \lfloor m/2 \rfloor$. This yields that $m \leq 2n - 1$, thus

$$\frac{\mathrm{NF}(\sigma)}{\mathrm{OPT}(\sigma)} \leq \frac{2n-1}{n} = 2 - 1/n \; .$$

Consequently we proved that the algorithm is asymptotically 2-competitive.

Now we prove that the bound is tight. Consider for each n the following sequence denoted by σ_n. The sequence contains $4n - 2$ items, the size of the $2i - 1$-th item is $1/2$, the size of the $2i$-th item is $1/4n$, $i = 1, \ldots, 2n$. Then algorithm NF puts the $(2i - 1)$-th and the $2i$-th items into the i-th bin for each bin, thus $\mathrm{NF}(\sigma_n) = 2n - 1$. The optimal offline algorithm puts pairs of $1/2$ size items into the first $n - 1$ bins and it puts one $1/2$ size item and the small items into the n-th bin, thus $\mathrm{OPT}(\sigma_n) = n$. Since $\mathrm{NF}(\sigma_n)/\mathrm{OPT}(\sigma_n) = 2 - 1/n$ and this function tends to 2 as n tends to ∞, thus we proved that the asymptotic competitive ratio of the algorithm is at least 2. ■

If $k > 1$ then better algorithms than NF are known for the k-bounded space model. The best known bounded space on-line algorithms belong to the family of ***harmonic algorithms***, where the basic idea is that the interval $(0, 1]$ is divided into subintervals and each item has

a type which is the subinterval of its size. The items of the different types are packed into different bins. The algorithm uses parallel versions of NF to pack the items belonging to the same type.

Algorithm FF and the weight function technique

In this part we present a method which is often used in the analysis of the bin packing algorithms. We show the method by analyzing algorithm FF (First Fit).

Algorithm FF can be used if the number of open bins is not bounded. The algorithm puts the item into the earliest opened bin where it fits. If the item does not fit into any of the bins then a new bin is opened and the algorithm puts the item into it. The pseudocode of the algorithm is also presented in the chapter of memory management. The asymptotic competitive ratio of the algorithm is bounded above by the following theorem.

Theorem 8.13 FF *is asymptotically 1.7-competitive.*

Proof. In the proof we use the weight function technique where the idea is that a weight is assigned to each item which measures in some way that how difficult can be to pack the item. Then the weight function and the total size of the items are used to bound the offline and on-line objective function values. Define the following weight function:

$$w(x) = \begin{cases} 6x/5, & \text{ha } 0 \le x \le 1/6 \\ 9x/5 - 1/10, & \text{ha } 1/6 \le x \le 1/3 \\ 6x/5 + 1/10, & \text{ha } 1/3 \le x \le 1/2 \\ 6x/5 + 2/5, & \text{ha } 1/2 < x \ . \end{cases}$$

Let $w(H) = \sum_{i \in H} w(a_i)$ for any set H of items. Then the following statements are valid for the weight function. Both lemmas can be proven by case disjunction based on the sizes of the possible items. The proofs are long and contain many technical details, therefore here we omit them.

Lemma 8.14 *If* $\sum_{i \in H} a_i \le 1$ *is valid for a set* H *of items, then* $w(H) \le 17/10$ *also holds.*

Lemma 8.15 *For an arbitrary list* L *of items* $w(L) \ge \mathrm{FF}(L) - 2$.

Using these lemmas we can prove easily that the algorithm is asymptotically 1.7-competitive. Consider an arbitrary list L of items. The optimal offline algorithm can pack the items of the list into $\mathrm{OPT}(L)$ bins. The algorithm packs items with total size at most 1 into each bin, thus by Lemma 8.14 it follows that $w(L) \le 1.7\mathrm{OPT}(L)$. On the other hand by Lemma 8.15 we obtain that $\mathrm{FF}(L) - 2 \le w(L)$ which yields that $FF(L) \le 1.7\mathrm{OPT}(L) + 2$, and that inequality proves that the algorithm is asymptotically 1.7-competitive.

It is important to note that the above bound is tight, it is also true that the asymptotic competitive ratio of FF is 1.7. ■

Many algorithms were developed with smaller asymptotic competitive ratio than 17/10, the best known algorithm is asymptotically 1.5888-competitive.

Lower bounds

In this part we consider the techniques for proving lower bounds on the possible asymptotic competitive ratio. First we present a simple lower bound and then we show how the idea of the proof can be extended into a general method.

Theorem 8.16 *No on-line algorithm for the bin packing problem can have smaller asymptotic competitive ratio than 4/3.*

Proof. Let A be an arbitrary on-line algorithm. Consider the following sequence of items. Let $\varepsilon < 1/12$ and L_1 be a list of n items of size $1/3 + \varepsilon$, and L_2 be a list of n items of size $1/2 + \varepsilon$. The input is started by L_1. Then A packs two items or one item into the bins. Denote by k the number of bins containing two items. Then the cost of the algorithm is $A(L_1) = k + n - 2k = n - k$. On the other hand the optimal offline algorithm can pack pairs of items into the bins thus $OPT(L_1) = n/2$.

Now suppose that the input is the combined list L_1L_2. The algorithm is an on-line algorithm it does not know at the beginning that it is the input L_1 or L_1L_2, thus it also uses k bins for packing two items from the part L_1. Therefore among the items of size $1/2 + \varepsilon$ only $n - 2k$ can be paired with earlier items the other ones need their own bin. Thus $A(L_1L_2) \geq n - k + (n - (n - 2k)) = n + k$. On the other hand the optimal offline algorithm can pack one smaller (size $1/3 + \varepsilon$) item and one larger (size $1/2 + \varepsilon$) item into each bin, thus $OPT(L_1L_2) = n$.

So we obtained that there is a list for algorithm A where

$$A(L)/OPT(L) \geq \max\left\{\frac{n-k}{n/2}, \frac{n+k}{n}\right\} \geq 4/3 \ .$$

Moreover in the constructed lists $OPT(L)$ is at least $n/2$ which can be arbitrarily large. This yields that the above inequality proves that the asymptotic competitive ratio of A is at least $4/3$, and this is what we wanted to prove. ■

The fundamental idea of the above proof is that a long sequence is considered (in this proof L_1L_2) and depending on the behaviour of the algorithm such prefix of the sequence is selected as input where the ratio of the costs is maximal. It is a natural extension to consider more difficult sequences. Many lower bounds were proven based on different sequences. On the other hand the computations which are necessary to analyze the sequence became more and more difficult. Below we show how the analysis of such sequences can be interpreted as mixed integer programming problem, which makes it possible to use computers to develop lower bounds.

Consider the following sequence of items. Let $L = L_1L_2 \ldots L_k$, where L_i contains $n_i = \alpha_i n$ identical items of size a_i. If algorithm A is asymptotically C-competitive then the inequality

$$C \geq \limsup_{n\to\infty} \frac{\mathrm{A}(L_1 \ldots L_j)}{\mathrm{OPT}(L_1 \ldots L_j)}$$

is valid for each j. It is enough to consider such algorithm for which the technique can achieve the minimal lower bound, thus or goal is to determine the value

$$R = \min_{\mathrm{A}} \max_{j=1,\ldots,k} \limsup_{n\to\infty} \frac{\mathrm{A}(L_1 \ldots L_j)}{\mathrm{OPT}(L_1 \ldots L_j)} \ ,$$

which value gives a lower bound on the possible asymptotic competitive ratio. We can determine this value as an optimal solution of a mixed integer programming problem. To define this programming problem we need the following definitions.

The contain of a bin can be described by the packing pattern of the bin, which gives

how many elements are contained in the bin from the subsequences. Formally a ***packing pattern*** is a k-dimensional vector $(p_1, \ldots, p_k)$, where the coordinate p_j is the number of elements contained in the bin from subsequence L_j. For the packing patterns the constraint $\sum_{j=1}^{k} a_j p_j \leq 1$ must hold. (This constraint ensures that the items described by the packing pattern fit into the bin.)

Classify the set T of the possible packing patterns. For each j let T_j be the set of the patterns for which the first positive coordinate is the j-th. (The pattern p belongs to class T_j if $p_i = 0$ for $i < j$ and $p_j > 0$.)

Consider the packing produced by A. Each bin is packed by some packing pattern therefore the packing can be described by the packing patterns. For each $p \in T$ denote by $n(p)$ the number of bins which are packed by the pattern p. The packing produced by the algorithm is given by the variables $n(p)$.

Observe that the bins which are packed by a pattern from class T_j receive their first element from the subsequence L_j. Therefore we obtain that the number of bins opened by A to pack the elements of subsequence $L_1 \ldots L_j$ can be given by the variables $n(p)$ as follows:

$$A(L_1 \ldots L_j) = \sum_{i=1}^{j} \sum_{p \in T_i} n(p).$$

Consequently for a given n the required value R can be determined by the solution of the following mixed integer programming problem.

Min R

$$\sum_{p \in T} p_j n(p) = n_j, \qquad 1 \leq j \leq k$$

$$\sum_{i=1}^{j} \sum_{p \in T_i} n_p \leq R \cdot OPT(L_1 \ldots L_j), \qquad 1 \leq j \leq k$$

$$n(p) \in \{0, 1, \ldots\}, \qquad p \in T$$

The first k constraints describe that the algorithm has to pack all items. The second k constraints describe that R is at least as large as the ratio of the on-line and offline costs for the subsequences considered.

By the list $L_1 L_2 \ldots L_k$ the set T of the possible packing patterns and also the optimal solutions $OPT(L_1 \ldots L_j)$ can be determined.

In this programming problem the number and the value of the variables can be large, thus instead of the problem its linear programming relaxation is considered. Moreover we are interested in the solution under the assumption that n tends to ∞ and it can be proven that the integer programming and the linear programming relaxation give the same bound in this case.

The best currently known bound was proven by this method and it states that no on-line algorithm can have smaller asymptotic competitive ratio than 1.5401.

8.4.2. Multidimensional models

The bin packing problem has three different multidimensional generalizations the vector packing, the box packing and the strip packing models. We only consider in details the strip packing problem. For the other generalizations we just give the model. In the ***vector packing problem*** the input is a list of d-dimensional vectors, and the algorithm has to pack these

vectors into the minimal number of bins. A packing is legal for a bin if for each coordinate the sum of the values of the elements packed into the bin is at most 1. In the on-line version the vectors are coming one by one and the algorithm has to assign the vectors to the bins without any information about the further vectors. In the ***box packing problem*** the input is a list of d-dimensional boxes and the goal is to pack the items into the minimal number of d-dimensional unit cube without overlapping. In the on-line version the items are coming one by one and the algorithm has to pack them into the cubes without any information about the further items.

On-line strip packing

In the ***strip packing problem*** there is a set of two dimensional rectangles, defined by their widths and heights, and the task is to pack them without rotation into a vertical strip of width w by minimizing the total height of the strip. We assume that the width of the rectangles is at most w and the height of the rectangles is at most 1. This problem appears in many situations. Usually, scheduling of tasks with shared resource involves two dimensions, the resource and the time. We can consider the widths as the resource and the heights as the time. Our goal is to minimize the total amount of time used. Some applications can be found in computer scheduling problems. We consider the on-line version where the rectangles arrive from a list one by one and we have to pack the rectangle into the vertical strip without any information about the further items. Most of the algorithms developed for the strip packing problem belong to the class of shelf algorithms. We consider this family of algorithms below.

SHELF algorithms

One basic way of packing into the strip is to define shelves and pack the rectangles into the shelves. By ***shelf*** we mean a rectangular part of the strip. Shelf packing algorithms place each rectangle into one of the shelves. If the algorithm decides which shelf will contain the rectangle, then the rectangle is placed into the shelf as much to the left as it is possible without overlapping the other rectangles placed earlier into the shelf considered. Therefore, after the arrival of a rectangle, the algorithm has to make two decisions. The first decision is whether to create a new shelf or not. If the algorithm creates a new shelf it also has to decide the height of the new shelf. The created shelves always start from the top of the previous shelf. The first shelf is placed to the bottom of the strip. The algorithm also has to choose the shelf to which it puts the rectangle. In what follows, we will say that it is possible to pack a rectangle into a shelf, if there is enough room for the rectangle in the shelf. It is obvious that if a rectangle is higher than a shelf we cannot place it into the shelf.

We consider only one algorithm in details. This algorithm was developed and analyzed by Baker and Schwarz in 1983 and it is called NFS_r algorithm. The algorithm depends on a parameter $r < 1$. For each j there is at most one active shelf with height r^j. We give the behaviour of the algorithm below.

After the arrival of a rectangle $p_i = (w_i, h_i)$ choose a value for k which satisfies $r^{k+1} < h_i \leq r^k$. If there is an active shelf with height r^k and it is possible to pack the rectangle into it, then pack it there. If there is no active shelf with height r^k, or it is not possible to pack the rectangle into the active shelf with height r^k, then create a new shelf with height r^k, put the rectangle into it, and let this new shelf be the active shelf with height r^k (if we had earlier an active shelf with height r^k then we close it).

Example 8.7 Let $r = 1/2$. Suppose that the size of the first item is $(w/2, 3/4)$. Then it is assigned to a shelf of height 1. We define a shelf of height 1 at the bottom of the strip, this shelf will be the active shelf with height 1 and we place the item into the left corner of this shelf. Suppose that the size of the next item is $(3w/4, 1/4)$. Then it is placed into a shelf of height $1/4$. There is no active shelf with this height so we define a new shelf of height $1/4$ on the top of the previous shelf. This will be the active shelf of height $1/4$ and the item is placed into its left corner. Suppose that the size of the next item is $(3w/4, 5/8)$. Then this item is placed into a shelf of height 1. It is not possible to pack it into the active shelf thus we close the active shelf and we define a new shelf of height 1 on the top of the previous shelf. This will be the active shelf of height 1 and the item is placed into its left corner. Suppose that the size of the next item is $(w/8, 3/16)$. Then this item is placed into a shelf of height $1/4$. We can pack it into the active shelf of height $1/4$ thus we pack it into that shelf as left as it is possible.

For the competitive ratio of NFS_r the following statements are valid.

Theorem 8.17 *Algorithm* NFS_r *is* $(\frac{2}{r} + \frac{1}{r(1-r)})$*-competitive. Algorithm* NFS_r *is asymptotically* $2/r$*-competitive.*

Proof. First we prove that the algorithm is $(\frac{2}{r} + \frac{1}{r(1-r)})$-competitive. Consider an arbitrary list of rectangles denote it by L. Let H_A denote the sum of the heights of the shelves which are active at the end of the packing, and let H_C be the sum of the heights of the other shelves. Let h be the height of the highest active shelf, and let H be the height of the highest rectangle. Since the algorithm created a shelf with height h, we have $H > rh$. As there is at most 1 active shelf for each height

$$H_A \leq h \sum_{i=0}^{\infty} r^i = \frac{h}{1-r} \leq \frac{H}{r(1-r)} \ .$$

Consider the shelves which are not active at the end. Consider the shelves of height hr^i for each i, denote the number of the closed shelves by n_i. Let S be one of these shelves with height hr^i. The next shelf S' with height hr^i contains one rectangle which would not fit into S. Therefore, the total width of the rectangles is at least w. Furthermore the height of these rectangles is at least hr^{i+1}, thus the total area of the rectangles packed into S and S' is at least hwr^{i+1}. If we pair the shelves of height hr^k for each i in this way, using the active shelf if the number of the shelves of the considered height is odd, we obtain that the total area of the rectangles assigned to shelves of height hr^i is at least $wn_ihr^{i+1}/2$. Thus the total area of the rectangles is at least $\sum_{i=0}^{\infty} wn_ihr^{i+1}/2$, and this yields that $\mathrm{OPT}(L) \geq \sum_{i=0}^{\infty} n_ihr^{i+1}/2$. On the other hand the total height of the closed shelves is $H_Z = \sum_{i=0}^{\infty} n_ihr^i$, and we obtain that $H_Z \leq 2\mathrm{OPT}(L)/r$.

Since $\mathrm{OPT}(L) \geq H$ is valid we proved the required inequality

$$\mathrm{NFS}_r(L) \leq \mathrm{OPT}(L)(2/r + 1/r(1-r) \ .$$

Since the heights of the rectangles are bounded by 1, thus H and H_A are bounded by a constant so we obtain immediately the result about the asymptotic competitive ratio.

■

Besides this algorithm some other shelf algorithms were investigated for the solution of the problem. We can interpret the basic idea of NFS_r as follows. We define classes of items belonging to types of shelves and the rectangles assigned to the classes are packed by the

classical bin packing algorithm NF. It is a straightforward idea to use other bin packing algorithms. The best known shelf algorithm was developed by Csirik and Woeginger in 1997, that algorithm uses the harmonic bin packing algorithm to pack the rectangles assigned to the classes.

Exercises

8.4-1 Suppose that the size of the items is bounded above by $1/3$. Prove that under this assumption the asymptotic competitive ratio of NF is $3/2$.

8.4-2 Suppose that the size of the items is bounded above by $1/3$. Prove Lemma 8.15 under this assumption.

8.4-3 Suppose that the sequence of items is given by a list $L_1L_2L_3$, where L_1 contains n items of size $1/2$, L_2 contains n items of size $1/3$, L_3 contains n items of size $1/4$. Which packing patterns can be used? Which patterns belong to the class T_2?

8.4-4 Consider the version of the strip packing problem where one can lengthen the rectangles keeping the area fixed. Consider the extension of NFS_r which lengthen the rectangles before the packing to have the same height as the shelf which is chosen to pack it. Prove that this algorithm is $2 + \frac{1}{r(1-r)}$-competitive.

8.5. On-line scheduling

The area of scheduling theory has a huge literature. The first result in on-line scheduling belongs to Graham, who analyzed in 1966 the List scheduling algorithm. We can say that despite the fact that Graham did not use the terminology which was developed in the area of the on-line algorithms, and he did not consider the algorithm as an on-line algorithm, he analyzed it as an approximation algorithm.

From the area of scheduling we only recall the definitions which are used in this chapter.

In a scheduling problem we have to find an optimal schedule of jobs. In the most fundamental model each job has a known processing time and to schedule the job we have to assign it to a machine and we have to give its starting time and a completion time, where the difference between the completion time and the starting time is the processing time. No machine may simultaneously run two jobs.

Concerning the machine environment three different models are considered. If the processing time of a job is the same for each machine then we call the machines identical machines. If each machine has a speed s_i, the jobs has a processing weight p_j and the processing time of job j on the i-th machine is p_j/s_i, then we call the machines related machines. If the processing time of job j is given by an arbitrary positive $P_j = (p_j(1), \ldots, p_j(m))$ vector, where the processing time of the job on the i-th machine is $p_j(i)$, then we call the machines unrelated machines.

Many objective functions are considered for scheduling problems, here we only consider such models where the goal is the minimization of the makespan (the maximal completion time).

In the next subsection we define the two most fundamental on-line scheduling models, and in the following two subsections we consider these models in details.

8.5.1. On-line scheduling models

Probably the following models are the most fundamental examples of online machine scheduling problems.

LIST model

In this model we have a fixed number of machines denoted by $M_1, M_2, \ldots, M_m$ and the jobs arrive from a list. This means that the jobs and their processing times are revealed to the online algorithm one by one. When a job is revealed the online algorithm must irrevocably assign the job to a machine with a starting time and a completion time.

By the ***load*** of a machine, we mean the sum of the processing times of all jobs assigned to the machine. Since the objective function is to minimize the maximal completion time, it is enough to consider the schedules where the jobs are scheduled on the machines without idle time. For these schedules the maximal completion time is the load for each machine. Therefore this scheduling problem is reduced to a load balancing problem, the algorithm has to assign the jobs to the machines minimizing the maximum load, which is the makespan in this case.

Example 8.8 Consider the LIST model and two identical machines. Consider the following sequence of jobs where the jobs are given by their processing time: $I = \{(j_1 = 4, j_2 = 3, j_3 = 2, j_4 = 5)\}$. Then the on-line algorithm first receives job j_1 from the list, the algorithm has to assign this job to one of the machines. Suppose that the job is assigned to machine M_1. Next the on-line algorithm receives job j_2 from the list, the algorithm has to assign this job to one of the machines. Suppose that the job is assigned to machine M_2. Next the on-line algorithm receives job j_3 from the list, the algorithm has to assign this job to one of the machines. Suppose that the job is assigned to machine M_2. Finally the on-line algorithm receives job j_4 from the list, the algorithm has to assign this job to one of the machines, suppose that the job is assigned to machine M_1. Then the loads on the machines are $4 + 5$ and $3 + 2$, and we can give a schedule where the maximal completion times on the machines are the loads: we can schedule the jobs on the first machine in the time intervals $(0, 4)$ and $(4, 9)$, and we can schedule the jobs on the second machine in the time intervals $(0, 3)$ and $(3, 5)$.

TIME model

In this model again there are a fixed number of machines. Each job has a processing time and a ***release time***. A job is revealed to the online algorithm at its release time. For each job the online algorithm must choose which machine it will run on and assign a start time. No machine may simultaneously run two jobs. Note that the algorithm is not required to immediately assign a job at its release time. However, if the online algorithm assigns a job at time t then it cannot use information about jobs released after time t and it cannot start the job before time t. The objective is to minimize the makespan.

Example 8.9 Consider the TIME model with two related machines. Let the first machine be M_1 with speed 1, and the second machine be M_2 with speed 2. Consider the following input $I = \{j_1 = (1, 0), j_2 = (1, 1), j_3 = (1, 1), j_4 = (1, 1)\}$, where the jobs are given by the (processing time, release time) pairs. Thus a job arrives at time 0 with processing time 1, the algorithm can start to process it on one of the machines or it can wait for jobs with larger processing time. Suppose that the algorithm waits till time 1/2 and then it starts to process the job on machine M_1. The completion time of the job is 3/2. At time 1 three further jobs arrive, at that time only M_2 can be used. Suppose that the

algorithm starts to process job j_2 on this machine. At time 3/2 both jobs are completed, suppose that the remaining jobs are started on machines M_1 and M_2 the completion times are 5/2 and 2, thus the makespan achieved by the algorithm is 5/2. Observe that an algorithm which starts the first job immediately at time 0 can make a better schedule with makespan 2. But it is important to note that in some cases it can be useful to wait for larger jobs before starting a job.

8.5.2. LIST model

The first algorithm in this model was developed by Graham. Algorithm LIST assigns each job to the machine where the actual load is minimal, if there are more machines with this property it uses the machine with the smallest index. This means that the algorithm tries to balance the loads on the machines. The competitive ratio of this algorithm is determined by the following theorem.

Theorem 8.18 *The competitive ratio of algorithm* LIST *is* $2-1/m$ *in the identical machines case.*

Proof. First we prove that the algorithm is $2-1/m$-competitive. Consider an arbitrary input sequence denote it by $\sigma = \{j_1, \ldots, j_n\}$, denote the processing times by $p_1, \ldots, p_n$. Consider the schedule produced by LIST. Let j_l be a job with maximal completion time. Investigate the starting time S_l of this job. Since LIST chooses the machine with minimal load thus the load was at least S_l on each of the machines when j_l was scheduled. Therefore we obtain that

$$S_l \leq \frac{1}{m}\sum_{\substack{j=1\\ j\neq l}}^{n} p_j = \frac{1}{m}(\sum_{j=1}^{n} p_j - p_l) = \frac{1}{m}(\sum_{j=1}^{n} p_j) - \frac{1}{m}p_l \ .$$

This yields that

$$LIST(\sigma) = S_l + p_l \leq \frac{1}{m}(\sum_{j=1}^{n} p_j) + \frac{m-1}{m}p_l \ .$$

On the other hand OPT also processes all of the jobs thus we obtain that $OPT(\sigma) \geq \frac{1}{m}(\sum_{j=1}^{n} p_j)$. Furthermore p_l is scheduled on one of the machines of OPT thus $OPT(\sigma) \geq p_l$. By these bounds we obtain that

$$LIST(\sigma) \leq (1 + \frac{m-1}{m})OPT(\sigma),$$

which inequality proves that LIST is $2-1/m$-competitive.

Now we prove that the bound is tight. Consider the following input. It contains $m(m-1)$ jobs with processing time $1/m$ and one job with processing time 1. Then LIST assigns $m-1$ small jobs to each machine and the last large job is assigned to M_1. Therefore its makespan is $1+(m-1)/m$. On the other hand the optimal algorithm schedules the large job on M_1 and m small jobs on the other machines, and its makespan is 1. Thus the ratio of the makespans is $2-1/m$ which shows that the competitive ratio of LIST is at least $2-1/m$. ■

It is hard to imagine any other algorithm for the on-line case, but many other algorithms were developed. The competitive ratios of the better algorithms tend to smaller number than

2 as the number of machines tends to ∞. Most of these algorithms are based on the following idea. The jobs are scheduled keeping the load uniformly on most of the machines but in contrast with LIST the loads are kept low on some of the machines, keeping the possibility to use these machines for scheduling large jobs which may arrive later.

Below we consider the more general cases where the machines are not identical. LIST may perform very badly, the processing time of a job can be very large on the machine where the actual load is minimal. But we can easily change the greedy idea of LIST as follows. The extended algorithm is called GREEDY and it assigns the job to the machine where the load with the processing time of the job is minimal. If there are more machines which has minimal value then the algorithm chooses among them the machine where the processing time of the job is minimal, if there are more machines with this property the algorithm chooses among them the one with the smallest index.

Example 8.10 Consider the case of related machines where there are 3 machines M_1, M_2, M_3 and the speeds are $s_1 = s_2 = 1$, $s_2 = 3$. Suppose that the input is $I = \{j_1 = 2, j_2 = 1, j_3 = 1, j_4 = 3, j_5 = 2)\}$, where the jobs are defined by their processing weight. Then the load after the first job is $2/3$ on machine M_3 and 2 on the other machines, thus j_1 is assigned to M_3. The load after job j_2 is 1 on all of the machines, its processing time is minimal on machine M_3, thus GREEDY assigns it to M_3. The load after job j_3 is 1 on M_1 and M_2, and $4/3$ on M_3, thus the job is assigned to M_1. The load after job j_4 is 4 on M_1, 3 on M_2, and 2 on M_3, thus the job is assigned to M_3. Finally the load after job j_5 is 3 on M_1, 2 on and M_2, and $8/3$ on M_3, thus the job is assigned to M_2.

Example 8.11 Consider the unrelated machines case with two machines and the following input $I = \{j_1 = (1,2), j_2 = (1,2), j_3 = (1,3), j_4 = (1,3)\}$, where the jobs are defined by the vectors of processing times. The load after job j_1 is 1 on M_1 and 2 on M_2, thus the job is assigned to M_1. The load after job j_2 is 2 on M_1 and also on M_2, thus the job is assigned to M_1 because it has smaller processing time there. The load after job j_3 is 3 on M_1 and M_2, thus the job is assigned to M_1 because it has smaller processing time there. Finally the load after job j_4 is 4 on M_1 and 3 on M_2, thus the job is assigned to M_2.

The competitive ratio of the algorithm is determined by the following theorems.

Theorem 8.19 *The competitive ratio of algorithm* GREEDY *is m in the unrelated machines case.*

Proof. First we prove that the competitive ratio of the algorithm is at least m. Consider the following input sequence. Let $\varepsilon > 0$ be a small number. The sequence contains m jobs. The processing time of job j_1 is 1 on machine M_1, $1 + \varepsilon$ on machine M_m, and ∞ on the other machines, $(p_1(1) = 1, p_1(i) = \infty, i = 2, \ldots, m-1, p_1(m) = 1 + \varepsilon)$. For job j_i, $i = 2, \ldots, m$ the processing time is i on machine M_i, $1 + \varepsilon$ on machine M_{i-1} and ∞ on the other machines $(p_j(j-1) = 1 + \varepsilon, p_j(j) = j, p_j(i) = \infty$, if $i \neq j - 1$ and $i \neq j)$.

Then job j_i is scheduled on M_i by GREEDY and the makespan is m. On the other hand the optimal offline algorithm schedules j_1 on M_m and j_i is scheduled on M_{i-1} for the other jobs thus the optimal makespan is $1 + \varepsilon$. The ratio of the makespans is $m/(1 + \varepsilon)$. This ratio tends to m, as ε tends to 0, and this proves that the competitive ratio of the algorithm is at least m.

Now we prove that the algorithm is m-competitive. Consider an arbitrary input sequence, denote the makespan in the optimal offline schedule by L^* and let $L(k)$ denote the

maximal load in the schedule produced by GREEDY after scheduling the first k jobs. Since the processing time of the i-th job is at least $\min_j p_i(j)$, and the load is at most L^* on the machines in the offline optimal schedule, thus we obtain that $mL^* \geq \sum_{i=1}^{n} \min_j p_i(j)$.

We prove by induction that the inequality $L(k) \leq \sum_{i=1}^{k} \min_j p_i(j)$ is valid. Since the first job is assigned to the machine where its processing time is minimal, thus the statement is obviously true for $k = 1$. Let $1 \leq k < n$ be arbitrary and suppose that the statement is true for k. Consider the $k+1$-th job. Let M_l be the machine where the processing time of this job is minimal. If we assign the job to M_l then we obtain by the induction hypothesis that the load on this machines is at most $L(k) + p_{k+1}(l) \leq \sum_{i=1}^{k+1} \min_j p_i(j)$.

On the other hand the maximal load in the schedule produced by GREEDY can not be more than the maximal load in the case when the job is assigned to M_l, thus $L(k+1) \leq \sum_{i=1}^{k+1} \min_j p_i(j)$, which means that we proved the inequality for $k+1$.

Therefore we obtained that $mL^* \geq \sum_{i=1}^{n} \min_j p_i(j) \geq L(n)$, which yields that the algorithm is m-competitive. ■

To investigate the related machines case consider an arbitrary input. Let L and L^* denote the makespans achieved by GREEDY and OPT respectively. The analysis of the algorithm is based on the following lemmas which give bounds on the loads of the machines.

Lemma 8.20 *The load on the fastest machine is at least $L - L^*$.*

Proof. Consider the schedule produced by GREEDY. Consider a job J which causes the makespan (its completion time is maximal). If this job is scheduled on the fastest machine then the lemma immediately follows, the load on the fastest machine is L. Suppose that J is not scheduled on the fastest machine. The optimal maximal load is L^*, thus the processing time of J on the fastest machine is at most L^*. On the other hand the completion time of J is L, thus at the time when the job was scheduled the load was at least $(L - L^*)$ on the fastest machine, otherwise GREEDY would assign J to the fastest machine. ■

Lemma 8.21 *If the loads are at least l on all machines having speed at least v then the loads are at least $l - 4L^*$ on all machines having speed at least $v/2$.*

Proof. If $l < 4L^*$, then the statement is obviously valid. Suppose that $l \geq 4L^*$. Consider the jobs which are scheduled by GREEDY on the machines having speed at least v in the time interval $[l - 2L^*, l]$. The total processing weight of these jobs is at least $2L^*$ times the total speed of the machines having speed at least v. This yields that there exists such job among them which is assigned by OPT to a machine having speed smaller than v (otherwise the optimal offline makespan would be larger than L^*). Let J be such a job.

Since OPT schedules J on a machine having speed smaller than v, thus the processing weight of J is at most vL^*. This yields that the processing time of J is at most $2L^*$ on the machines having speed at least $v/2$. On the other hand GREEDY produces a schedule where the completion time of J is at least $l - 2L^*$, thus at the time when the job was scheduled the loads were at least $l - 4L^*$ on the machines having speed at most $v/2$ (otherwise GREEDY would assign J to one of these machines).

■

Now we can prove the following statement.

Theorem 8.22 *The competitive ratio of algorithm* GREEDY *is $\Theta(\log m)$ in the related machines case.*

Proof. First we prove that GREEDY is $O(\lg m)$-competitive. Consider an arbitrary input. Let L and L^* denote the makespans achieved by GREEDY and OPT respectively.

Let $v_{\max}$ be the speed of the fastest machine. Then by Lemma 8.20 the load on this machine is at least $L - L^*$. Then using Lemma 8.21 we obtain that the loads are at least $L - L^* - 4iL^*$ on the machines having speed at least $v_{\max}2^{-i}$. Therefore the loads are at least $L - (1 + 4\lceil \lg m \rceil)L^*$ on the machines having speed at least $v_{\max}/m$. Denote I the set of the machines having speed at most $v_{\max}/m$.

Denote W the sum of the processing weights of the jobs. OPT can find a schedule of the jobs which has maximal load L^*, and there are at most m machines having smaller speed than $v_{\max}/m$ thus

$$W \leq L^* \sum_{i=1}^{m} v_i \leq mL^* v_{\max}/m + L^* \sum_{i \notin I} v_i \leq 2L^* \sum_{i \notin I} v_i \ .$$

On the other hand GREEDY schedules the same jobs thus the load on some machine not included in I is smaller than $2L^*$ in the schedule produced by GREEDY (otherwise we would obtain that the sum of the processing weights is greater than W).

Therefore we obtain that

$$L - (1 + 4\lceil \lg m \rceil)L^* \leq 2L^* \ ,$$

which yields that $L \leq 3 + 4\lceil \lg m \rceil)L^*$, which proves that GREEDY is $O(\lg m)$-competitive.

Now we prove that the competitive ratio of the algorithm is at least $\Omega(\lg m)$. Consider the following set of machines. G_0 contains one machine with speed 1, G_1 contains 2 machines with speed $1/2$. For each $i = 1, 2, \ldots, k$, G_i contains machines with speed 2^{-i}, and G_i contains $|G_i| = \sum_{j=0}^{i-1} |G_j| 2^{i-j}$ machines. Observe that the number of jobs of size 2^{-i} which can be scheduled during time 1 is the same on the machines of G_i and on the machines of $G_0 \cup G_1 \ldots, \cup G_{i-1}$. It is easy to calculate that $|G_i| = 2^{2i-1}$, if $i \geq 1$, thus the number of machines is $1 + \frac{2}{3}(4^k - 1)$.

Consider the following input sequence. In the first phase $|G_k|$ jobs arrive having processing weight 2^{-k}, in the second phase $|G_{k-1}|$ jobs arrive having processing weight $2^{-(k-1)}$, in the i-th phase $|G_i|$ jobs arrive with processing weight 2^{-i}, and the sequence ends with the $k+1$-th phase, which contains one job with processing weight 1. An offline algorithm can schedule the jobs of the i-th phase on the machines of set G_{k+1-i} achieving maximal load 1, thus the optimal offline cost is at most 1.

Investigate the behaviour of algorithm GREEDY on this input. The jobs of the first phase can be scheduled on the machines of $G_0, \ldots, G_{k-1}$ during time 1, and it takes also time 1 to schedule these jobs on the machines of G_k. Thus GREEDY schedules these jobs on the machines of $G_0, \ldots, G_{k-1}$, and the loads are 1 on these machines after the first phase. Then the jobs of the second phase are scheduled on the machines of $G_0, \ldots, G_{k-2}$, the jobs of the third phase are scheduled on the machines of $G_0, \ldots, G_{k-3}$ and so on. Finally the jobs of the k-th and $k+1$-th phase are scheduled on the machine of set G_0. Thus the cost of GREEDY is $k+1$, (this is the load on the machine of set G_0). Since $k = \Omega(\lg m)$, thus we proved the required statement. ■

8.5.3. TIME model

In this model we only investigate one algorithm. The basic idea is to divide the jobs into groups by the release time and to use an optimal offline algorithm to schedule the jobs from the groups. This algorithm is called ***interval scheduling algorithm*** and we denote it by INTV. Let t_0 be the release time of the first job, and $i = 0$. Then the algorithm is defined by the following pseudocode:

INTV(I)

1 **while not** end of sequence
2 let H_i be the set of the unscheduled jobs released till t_i
3 let OFF_i be an optimal offline schedule of the jobs of H_i
4 schedule the jobs as it is determined by OFF_i starting the schedule at t_i
5 let q_i be the maximal completion time
6 **if** new job is released in time interval $(t_i, q_i]$ or the sequence is ended
7 **then** $t_{i+1} \leftarrow q_i$
7 **else** let t_{i+1} be the release time of the next job
8 $i \leftarrow i + 1$

Example 8.12 Consider two identical machines. Suppose that the sequence of jobs is $I = \{j_1 = (1, 0), j_2 = (1/2, 0), j_3 = (1/2, 0), j_4 = (1, 3/2), j_5 = (1, 3/2), j_6 = (2, 2)\}$, where the jobs are defined by the (processing time, release time) pairs. In the first iteration j_1, j_2, j_3 are scheduled, an optimal offline algorithm schedules j_1 on machine M_1 and j_2, j_3 on machine M_2, the jobs are completed at time 1. Then no new job released in the time interval (0, 1] thus the algorithm waits for new job till time 3/2. Then the second iteration starts j_4 and j_5 are scheduled on M_1 and M_2 respectively in the time interval $[3/2, 5/2)$. During this time interval j_6 released thus at time 5/2 the next iteration starts and INTV schedules j_6 on M_1 in the time interval $[5/2, 9/2]$.

The following statement holds for the competitive ratio of algorithm INTV.

Theorem 8.23 *In the TIME model algorithm* INTV *is 2-competitive.*

Proof. Consider an arbitrary input and the schedule produced by INTV. Denote the number of iterations by i. Let $T_3 = t_{i+1} - t_i$, $T_2 = t_i - t_{i-1}$, $T_1 = t_{i-1}$ and let T_{OPT} denote the optimal offline cost. Then $T_2 \leq T_{OPT}$. This inequality is obvious if $t_{i+1} \neq q_i$. If $t_{i+1} = q_i$, then the inequality holds because the optimal offline algorithm also has to schedule the jobs which are scheduled in the i-th iteration by INTV and INTV uses an optimal offline schedule in each iteration. On the other hand $T_1 + T_3 \leq T_{OPT}$. To prove this inequality first observe that the release time is at least $T_1 = t_{i-1}$ for the jobs scheduled in the i-th iteration (otherwise the algorithm would schedule them in the $i - 1$-th iteration). Therefore the optimal algorithm also must schedule these jobs after time T_1. On the the other hand it takes at least time T_3 to process these jobs because INTV uses optimal offline algorithm in the iterations. The makespan of the schedule produced by INTV is $T_1 + T_2 + T_3$, and we have shown that $T_1 + T_2 + T_3 \leq 2T_{OPT}$ thus we proved that the algorithm is 2-competitive. ■

Some other algorithms are also developed in the TIME model. Vestjens proved that the ***on-line*** **LPT** algorithm is 3/2-competitive. This algorithm schedules the longest unscheduled, released job at any time when some machine is available. The following lower bound

for the possible competitive ratios of the on-line algorithms is also given by Vestjens.

Theorem 8.24 *The competitive ratio of any on-line algorithm is at least* 1.3473 *in the TIME model for minimizing the makespan.*

Proof. Let $\alpha = 0.3473$ be the solution of the equation $\alpha^3 - 3\alpha + 1 = 0$ which belongs to the interval $[1/3, 1/2]$. We prove that no on-line algorithm can have smaller competitive ratio than $1 + \alpha$. Consider an arbitrary on-line algorithm, denote it by ALG. Investigate the following input sequence.

At time 0 one job arrives with processing time 1. Let S_1 be the time when the algorithm starts to process the job on one of the machines. If $S_1 > \alpha$, then the sequence is ended and $\mathrm{ALG}(I)/\mathrm{OPT}(I) > 1 + \alpha$, which proves the statement. So we can suppose that $S_1 \leq \alpha$.

The release time of the next job is S_1 and its processing time is $\alpha/(1 - \alpha)$. Denote its starting time by S_2. If $S_2 \leq S_1 + 1 - \alpha/(1 - \alpha)$, then we end the sequence with $m - 1$ jobs having release time S_2, and processing time $1 + \alpha/(1 - \alpha) - S_2$. Then an optimal offline algorithm schedules the first two jobs on the same machine and the last $m - 1$ jobs on the other machines starting them at time S_2, thus its cost is $1 + \alpha/(1 - \alpha)$. On the other hand the on-line algorithm must schedule one of the last $m - 1$ jobs after the completion of the first or the second job thus $\mathrm{ALG}(I) \geq 1 + 2\alpha/(1 - \alpha)$ in this case, which yields that the competitive ratio of the algorithm is at least $1 + \alpha$. Therefore we can suppose that $S_2 > S_1 + 1 - \alpha/(1 - \alpha)$.

Then at time $S_1 + 1 - \alpha/(1 - \alpha)$ further $m - 2$ jobs arrive having processing time $\alpha/(1 - \alpha)$ and one job having processing time $1 - \alpha/(1 - \alpha)$. The optimal offline algorithm schedules the second and the last jobs on the same machine the other jobs are scheduled alone on the other machines and the makespan of the schedule is $1 + S_1$. Since before time $S_1 + 1 - \alpha/(1 - \alpha)$ none of the last m jobs is started by ALG thus after this time ALG must schedule at least two jobs on one of the machines and the maximal completion time is at least $S_1 + 2 - \alpha/(1 - \alpha)$. Since $S_1 \leq \alpha$, thus the ratio $\mathrm{OPT}(I)/\mathrm{ALG}(I)$ is minimal if $S_1 = \alpha$ and in this case the ratio is $1 + \alpha$, which proves the theorem. ■

Exercises

8.5-1 Prove that the competitive ratio is at least 3/2 for any on-line algorithm in the case of two identical machines.

8.5-2 Prove that LIST is not constant competitive in the unrelated machines case.

8.5-3 Prove that the modification of INTV which uses a c-approximation schedule (a schedule with at most c times more cost than the optimal cost) in each step instead of the optimal offline schedule is $2c$-competitive.

Problems

8-1. Paging problem

Consider the special case of the k-server problem where the distance between each pair of points is 1. (This problem is equivalent with the on-line paging problem.) Analyze the algorithm which serves the requests not having server on their place by the server which was used least recently. (This algorithm is equivalent with the LRU paging strategy.) Prove that the algorithm is k-competitive.

8-2.* ALARM2 *algorithm
Consider the following alarming algorithm for the data acknowledgement problem. ALARM2 is obtained from the general definition with the values $e_j = 1/|\sigma_j|$. Prove that the algorithm is not constant-competitive.

8-3. Bin packing lower bound
Prove, that no on-line algorithm can have smaller competitive ratio than 3/2 using a sequence which contains items of size $1/7 + \varepsilon$, $1/3 + \varepsilon$, $1/2 + \varepsilon$, where ε is a small positive number.

8-4. Strip packing with modifiable rectangles
Consider the following version of the strip packing problem. In the new model the algorithms are allowed to lengthen the rectangles keeping the area fixed. Develop a 4-competitive algorithm for the solution of the problem.

8-5. On-line* LPT *algorithm
Consider the algorithm in the TIME model which starts the longest unscheduled released job at any time when a machine is available. This algorithm is called on-line LPT. Prove that the algorithm is 3/2-competitive.

Chapter notes

More details about the results on on-line algorithms can be found in the books [6, 16].

The first results about the k-server problem (Theorems 8.1 and 8.2) are published by Manasse, McGeoch and Sleator in [27]. The presented algorithm for the line (Theorem 8.3) was developed by Chrobak, Karloff, Payne and Viswanathan (see [10]). Later Chrobak and Larmore in [8] extended the algorithm for trees. The first constant-competitive algorithm for the general problem was developed by Fiat, Rabani and Ravid (see [15]). The best known algorithm is based on the work function technique. The first work function algorithm for the problem was developed in [9] by Chrobak and Larmore. Koutsoupias and Papadimitriou proved in [25] that the work function algorithm is $2k - 1$-competitive.

The first mathematical model for the data acknowledgement problem and the first results (Theorems 8.5 and 8.6) are presented in [13] by Dooly, Goldman, and Scott. Albers and Bals considered a different objective function in [1]. Karlin Kenyon and Randall investigated randomized algorithms for the data acknowledgement problem in [24]. The Landlord algorithm was developed in [33] by Young. The detailed description of the results in the area of on-line routing can be found in the survey [26] written by Leonardi. The exponential algorithm for the load balancing model is investigated by Aspnes, Azar, Fiat, Plotkin and Waarts in [2]. The exponential algorithm for the throughput objective function is applied by Awerbuch, Azar and Plotkin in [3].

A detailed survey about the theory of on-line bin packing is written by Csirik and Woeginger (see [11]). The algorithms NF and FF are analyzed with competitive analysis by Johnson, Demers, Ullman, Garey and Graham in [22, 23], further results can be found in the PhD thesis of Johnson ([21]). Van Vliet applied the packing patterns to prove lower bounds for the possible competitive ratios in [31, 34]. For the on-line strip packing problem algorithm NFS_r was developed and analyzed by Baker and Schwarz in [5]. Later further shelf packing algorithms were developed, the best shelf packing algorithm for the strip pac-

king problem was developed by Csirik and Woeginger in [12].

A detailed survey about the results in the area of on-line scheduling was written by Sgall [28]. The first on-line result is the analysis of algorithm LIST, it was published in [18] by Graham. Many further algorithms were developed and analyzed for the identical machines case, the algorithm with smallest competitive ratio (tends to 1.9201 as the number of machines tends to ∞) was developed by Fleischer and Wahl in [17]. The lower bound for the competitive ratio of GREEDY in the related machines model was proven by Cho and Sahni in [7]. The upper bound, the related machines case and a more sophisticated exponential function based algorithm were presented by Aspnes, Azar, Fiat, Plotkin and Waarts in [2]. A summary of the further results about the applications of the exponential function technique in the area of on-line scheduling can be found in the paper of Azar ([4]). The interval algorithm presented in the TIME model and Theorem 8.23 are based on the results of Shmoys, Wein and Williamson (see [29]). A detailed description of the further results (on-line LPT, lower bounds) in the area TIME model can be found in the PhD thesis of Vestjens [35]. We only presented the most fundamental on-line scheduling models in the chapter, recently an interesting model was developed where the number of the machines is not fixed the algorithm is allowed to purchase machines, the model is investigated in the papers [20] and [14].

Problem *8-1.* is based on [30], Problem *8-2.* is based on [13], Problem *8-3.* is based on [32], Problem *8-4.* is based on [19] and Problem *8-5.* is based on [35].

Bibliography

[1] S. Albers, H. Bals. Dynamic TCP acknowledgement, penalizing long delays. In *Proceedings of the 25th ACM-SIAM Symposium on Discrete Algorithms*, pp. 47–55, 2003. 401

[2] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, O. Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. *Journal of the ACM*, 44:486–504, 1997. 401, 402

[3] B. Awerbuch, Y. Azar S. Plotkin. Throughput-competitive online routing. In *Proceedings of the 34th Annual Symposium on Foundations of Computer Science*, pp. 32–40, 1993. 401

[4] Y. Azar. On-line load balancing. Lecture Notes in Computer Science, Vol. 1442. Springer-Verlag, pp. 178–195, 1998. 402

[5] B. S. Baker, J. S. Schwartz. Shelf algorithms for two dimensional packing problems. *SIAM Journal on Computing*, 12:508–525, 1983. 401

[6] A. Borodin R. El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998. 401

[7] Y. Cho, S. Sahni. Bounds for list schedules on uniform processors. *SIAM Journal on Computing*, 9(1):91–103, 1980. 402

[8] M. Chrobak, L. Larmore. An optimal algorithm for k-servers on trees. *SIAM Journal on Computing*, 20:144–148, 1991. 401

[9] M. Chrobak, L. Larmore. The server problem and on-line games. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 7, pp. 11-64. American Mathematical Society, 1992. 401

[10] M. Chrobak, H. J. Karloff, T. Payne, S. Vishwanathan. New results on the server problem. *SIAM Journal on Discrete Mathematics*, 4:172–181, 1991. 401

[11] J. Csirik, G. Woeginger. On-line packing and covering problems. Lecture Notes in Computer Science, Vol. 1442, pp. 147–177. Springer-Verlag, 1998. 401

[12] J. Csirik, G. J. Woeginger. Shelf algorithms for on-line strip packing. *Information Processing Letters*, 63:171–175, 1997. 402

[13] D. R. Dooly, S. A. Goldman, S. D. Scott. On-line analysis of the TCP acknowledgement delay problem. *Journal of the ACM*, 48:243–273, 2001. 401, 402

[14] Gy. Dósa, Y. He. Better online algorithms for scheduling with machine cost. *SIAM Journal on Computing*, 33(5):1035–1051, 2004. 402

[15] A. Fiat, Y. Rabani, Y. Ravid. Competitive k-server algorithms. *Journal of Computer and System Sciences*, 48:410–428, 1994. 401

[16] A. Fiat, G. Woeginger (szerkesztők). *Online Algorithms. The State of Art*. Springer-Verlag, 1998. 401

[17] R. Fleischer, M. Wahl. On-line scheduling revisited. *Journal of Scheduling*, 3(6):343–353, 2000. 402

[18] R. L. Graham. Bounds for certain multiprocessor anomalies. *The Bell System Technical Journal*, 45:1563–1581, 1966. 402

[19] Cs. Imreh. Online strip packing with modifiable boxes. *Operations Research Letters*, 66:79–86, 2001. 402

[20] Cs. Imreh, J. Noga. Scheduling with machine cost. In *Proceedings of APPROX'99*, Lecture Notes in Computer Science, Vol. 1540, pp. 168–176, 1999. 402

[21] D. S. Johnson. *Near-Optimal Bin Packing Algorithms*. PhD thesis. 401

[22] D. S. Johnson. Fast algorithms for bin packing. *Journal of Computer and System Sciences*, 8:272–314, 1974. 401

[23] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, R. L. Graham. Worst-case performance-bounds for simple one-dimensional bin packing algorithms. *SIAM Journal on Computing*, 3:299–325, 1974. 401

[24] A. R. Karlin, C. Kenyon, D. Randall. Dynamic TCP acknowledgement and other stories about $e/(e-1)$. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, 502–509. o., 2001. 401

[25] E. Koutsoupias, C. Papadimitriou. On the k-server conjecture. *Journal of the ACM*, 42:971–983, 1995. 401

[26] S. Leonardi. On-line network routing. Lecture Notes in Computer Science, Vol. 1442, pp. 242–267. Springer-Verlag, 1998. 401

[27] M. Manasse, L. McGeoch, D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11:208–230, 1990. 401

[28] J. Sgall. On-line scheduling. Lecture Notes in Computer Science, Vol. 1442, pp. 196–231. Springer-Verlag, 1998. 402

[29] D. B. Shmoys, J. Wein, D. P. Williamson. Scheduling parallel machines online. *SIAM Journal on Computing*, 24:1313–1331, 1995. 402

[30] D. Sleator R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985. 402

[31] A. van Vliet. An improved lower bound for on-line bin packing algorithms. *Information Processing Letters*, 43:277–284, 1992. 401

[32] A. C. C. Yao. New algorithms for bin packing. *Journal of the ACM*, 27:207–227, 1980. 402

[33] N. Young. On-line file caching. *Algorithmica*, 33:371–383, 2002. 401

[34] A. van Vliet. *Lower and upper bounds for on-line bin packing and scheduling heuristics*. PhD thesis, Erasmus University, Rotterdam, 1995. 401

[35] A. Vestjens. *On-line machine scheduling*. PhD thesis, Eindhoven University of Technology, 1997. 402

Subject Index

Name index

Contents