Contents

III. NUMERICAL METHODS

10. Competitive Analysis

In on-line computation an algorithm must make its decisions based only on past events without secure information on future. Such methods are called ***on-line algorithms***. On-line 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 have started to work on problems related to on-line algorithms. Many subfields have been developed and investigated. Nowadays new results of the area have been presented on the most important conferences about algorithms. This chapter does not give a detailed overview about the results, because it is not 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 best-known on-line problems—the on-line k-server problem—and some of the related results. Then we deal with a new area by presenting on-line problems belonging to computer networks. In the next section the on-line bin packing problem and its multidimensional generalisations are presented. Finally in the last section of this chapter we show some basic results concerning the area of on-line scheduling.

10.1. Notions, definitions

Since an on-line algorithm makes its decisions based on partial information without knowing the whole instance in advance, we cannot expect it to give the optimal solution which can be given by an algorithm having full information. An algorithm which knows the whole input in advance is called ***off-line algorithm.***

There are two main methods to measure the performance of on-line algorithms. One possibility is to use ***average case analysis*** where we hypothesise 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 off-line objective function value.

In case of on-line minimisation 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 off-line cost for each input. The ***competitive ratio of an algorithm*** is the smallest 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 off-line 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 $\mathrm{ALG}(I) \le C \cdot \mathrm{OPT}(I)$ is valid for each input I.

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

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

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

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

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

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

The main property of the asymptotic ratio is 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.

Similar definitions can be given for maximisation problems. In that case algorithm ALG is called C-competitive, if $\mathrm{ALG}(I) \ge C \cdot \mathrm{OPT}(I)$ is valid for each input I, and the algorithm is weakly C-competitive if there exists such a constant B that $\mathrm{ALG}(I) \ge C \cdot \mathrm{OPT}(I) + B$ is valid for each input I. The asymptotic ratio for maximisation problems can be given as follows:

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

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

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

Many papers are devoted ***randomised on-line algorithms,*** in which 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 randomised on-line algorithms.

10.2. The k-server problem

One of the best-known on-line problems is the on-line k-server problem. To give the definition of the general problem the notion of metric spaces is needed. A pair (M, d) (where M contains the points of the space and 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 requests appearing at the points of the metric space by sending a server to the point where the request appears.

The problem is on-line 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 minimise the total distance travelled by the servers. 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 stores for each server the total distance travelled by the server. The servers and the points in the space where the servers are located are denoted by $s_1, \ldots, s_k$. Let the total distance travelled by the server s_i be D_i. After the arrival of a request at point P algorithm BAL uses 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 server configuration $S = \{s_1, \ldots, s_k\}$ and the distances travelled by the servers which distances have starting values $D_1 = \cdots = D_k = 0$. The behaviour of the algorithm on input $I = P_1, \ldots, P_n$ can be given by the following pseudocode:

BAL(I)

```
for j ← 1 to n
    do i ← argmin{D_i + d(s_i, P_j)}
       serve the request with server i
       D_i ← D_i + d(s_i, P_j)
       s_i ← P_j
```

Example 10.1 Consider the two dimensional 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. Therefore 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 action 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 $(2, 4)$, so $D_1+d((0, 0), (2, 4)) = \sqrt{20} > D_2+d((1, 4), (2, 4)) = 3+1 = 4$, 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 $(1, 4)$, so $D_1 + d((0, 0), (1, 4)) = \sqrt{17} < D_2 + d((2, 4), (1, 4)) = 4 + 1 = 5$, 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 10.1 *Algorithm* BALANCE *is weakly k-competitive for the metric spaces containing $k + 1$ points.*

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

Theorem 10.2 *There is no 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 say ONL. Denote the points of the starting configuration of ONL by $P_1, P_2, \ldots, P_k$, and let P_{k+1} be 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.

First calculate the value ONL(I). The algorithm does not have any servers 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

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

where Q_{n+1} denotes the point from which the server was sent to serve Q_n. (This is the point where the $(n + 1)$-th request would appear.) Now consider the cost

$\mathrm{OPT}(I)$. Instead of calculating the optimal off-line cost we define k different off-line algorithms, and we use the mean of the costs resulting from these algorithms. Since the cost of each off-line algorithm is at least as much as the optimal off-line cost, the calculated mean is an upper bound for the optimal off-line cost.

We define the following k off-line algorithms, denoted by $\mathrm{OFF}_1, \ldots, \mathrm{OFF}_k$. Suppose that the servers are at 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 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 $P_1, P_2, \ldots, P_{k+1}$ contains a server, 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 because of the definition of the algorithms. Now consider the step where a request is served. Call the algorithms which do not move a server for serving the request stable, and the other algorithms unstable. The server configurations of the stable algorithms remain unchanged, so these configurations remain different from each other. Each unstable algorithm moves a server from point Q_{i-1}. This point is the place of the last request, thus the stable algorithms have server at it. Therefore, an unstable algorithm and a stable algorithm cannot have the same configuration after serving the request. Furthermore, each 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 configurations. On the other hand, each configuration has a server at point Q_{i-1}, therefore there is only one configuration where there is 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 of the input (this is the cost of moving the servers to the starting configuration of the defined algorithms).

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

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

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. ■

There are many interesting results in connection with this problem.have 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. Later Koutsoupias and Papadimitriou could analyse an algorithm based on the work function technique and they could prove that it 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. Determining the competitive ratio of the algorithm, or developing a k-competitive algorithm is still among the most important open problems in the area of on-line algorithms. We present the work function algorithm below.

Denote the starting configuration of the on-line servers by A_0. Then after the t-th request the ***work function*** value belonging to multiset X is the minimal cost needed to serve the first t requests starting at configuration A_0 and ending at 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. The Work-Function algorithm uses 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 10.2 Consider the metric space containing three points A, B and C with the distances $d(A,B) = 1$, $d(B,C) = 2$, $d(A,C) = 3$. Suppose that we have two servers and the starting configuration is $\{A,B\}$. In this case the starting work function values are $w_0(\{A,A\}) = 1$, $w_0(\{A,B\}) = 0$, $w_0(\{A,C\}) = 2$, $w_0(\{B,B\}) = 1$, $w_0(\{B,C\}) = 3$, $w_0(\{C,C\}) = 5$. 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 10.3 *The* Work-Function *algorithm is weakly* $2k-1$*-competitive.*

Besides the general problem many particular cases have been investigated. If the distance of any pair of points is 1, then we obtain the on-line paging problem as a special case. 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, which algorithm is called Double-Coverage. A request at point P is served by server s which is the closest to P. Moreover, if there are servers also on the opposite side of P, then the closest server among them moves distance $d(s,P)$ into the direction of P. Hereafter 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 is denoted by $S = (s_1, \ldots, s_k)$ which contains points (real numbers) too. The algorithm can be defined by the following pseudocode:

DC(I, S)

for $j \leftarrow 1$ **to** n
    do $i \leftarrow \text{argmin}_l d(P_j, s_l)$
if $s_i = \min_l s_l$ or $s_i = \max_l s_l$
  then $\triangleright$ the request is served by the i-th server
      $s_i \leftarrow P_j$
  else if $s_i \leq P_j$
      then $m \leftarrow \text{argmin}_{l:s_l > P_j} d(s_l, P_j)$
        $\triangleright$ the request is served by the i-th server
          $s_m \leftarrow s_m - d(s_i, P_j)$
          $s_i \leftarrow P_j$
      else **if** $s_i \geq P_j$
          then $r \leftarrow \text{argmin}_{l:s_l < P_j} d(s_l, P_j)$
            $\triangleright$ the request is served by the i-th server
              $s_r \leftarrow s_r + d(s_i, P_j)$
              $s_i \leftarrow P_j$

Example 10.3 Suppose that there are three servers s_1, s_2, s_3 located at 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 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 travels distance 1 into the direction of 2. Therefore the cost of serving the request is 2 and the servers will be at 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 10.4 *Algorithm* DC *is weakly k-competitive on the line.*

Proof Consider an arbitrary sequence of requests and denote this input by I. During the analysis of the procedure we suppose that one off-line optimal algorithm and DC are running parallel on the input. We also suppose that each request is served first by the off-line 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 off-line 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 swapping 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, so the on-line and off-line costs are compared using the changes of the potential function. Let us define the following potential function:

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

The following statements are valid for the potential function.

- While OPT is serving a request the increase of the potential function is not more than k times the distance travelled by the servers of OPT.
- While DC is serving 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 the theorem easily. 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 that the algorithms is weakly k-competitive (Φ_0 does not depend on the input sequence only on the starting position of the servers).

Now we prove the properties of the potential function.

First consider the case when one of the off-line servers travels distance d. The first part of the potential function increases at most by 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. The following two cases are distinguished depending on the positions of the on-line servers.

First suppose 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, since the other case is completely similar. In this case s_1 is the closest server and DC sends s_1 to P and 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 $|x_1 - s_1|$ changes; it decreases by $d(s_1, P)$, thus the first part decreases by $kd(s_1, P)$. The second sum is increasing; the increase is $(k-1)d(s_1, P)$, thus the value of Φ decreases by $d(s_1, P)$.

Assume that 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. In this case 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 by $d(s_i, P)$ and 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

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

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

Exercises

10.2-1 Suppose that (M, d) is a metric space. Prove that (M, q) is also a metric space where $q(x, y) = \min\{1, d(x, y)\}$.

10.2-2 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.

10.2-3 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 needed to move the servers from configuration X to configuration Z).

10.2-4 Consider the line as a metric space. Suppose that the servers of the on-line algorithm are at points $2, 4, 5, 7$, and the servers of the off-line algorithm are at points $1, 3, 6, 9$. Calculate the value of the potential function used in the proof of Theorem 10.4. How does this potential function change, if the on-line server moves from point 7 to point 8?

10.3. Models related to computer networks

The theory of computer networks has become one of the most significant areas of computer science. In the planning of computer networks many optimisation 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 have 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.

10.3.1. 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. The data acknowledgement problem is to determine the time of sending acknowledgements. An acknowledgement can acknowledge many packets but waiting for long time can cause the resending of the packets and that results in the congestion of the network. On the other hand, sending an acknowledgement about the arrival of each packet immediately would cause again the congestion of the network. The first optimisation model for determining the sending times of the acknowledgements was developed by Dooly, Goldman and Scott in 1998. 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 optimisation 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 has been 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 analyse an algorithm from this class in details. 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 10.4 Consider the following example. The first packet arrives at time 0 ($a_1 = 0$), so ALARM sets an alarm with 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 has not been acknowledged yet and we reset the alarm with 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 have not been 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 $a_4 = 1$. 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 value $e_4 = 1$ for time 2.

Theorem 10.5 *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 off-line algorithm sends k^* acknowledgements. If $k^* \geq k$, then $\mathrm{OPT}(I) \geq k = \mathrm{ALARM}(I)/2$ is obviously valid, thus we obtain that the algorithm is 2-competitive. If $k^* < k$, then at least $k - k^*$ time intervals among the ones defined by the acknowledgements of algorithm ALARM do not contain any of the off-line acknowledgements. This yields that the off-line 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 10.6 *There is no on-line algorithm for the data acknowledgement problem which has smaller competitive ratio than* 2.

Proof Consider an arbitrary on-line algorithm and denote it by ONL. Analyse the following input. Consider a long sequence of packets where the packets always arrive immediately after the time when ONL sends an acknowledgement. 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 EVEN sends the acknowledgements after the even numbered packets.

The costs achieved by these algorithms are

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

and

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

Therefore $\mathrm{EVEN}(I_{2n}) + \mathrm{ODD}(I_{2n}) = \mathrm{ONL}(I_{2n}) + 1$. On the other hand, none of the costs achieved by ODD and EVEN is greater than the optimal off-line cost, thus $\mathrm{OPT}(I_{2n}) \le \min\{\mathrm{EVEN}(I_{2n}), \mathrm{ODD}(I_{2n})\}$, which yields that $\mathrm{ONL}(I_{2n})/\mathrm{OPT}(I_{2n}) \ge 2 - 1/\mathrm{OPT}(I_{2n})$. From 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. ■

10.3.2. The file caching problem

The file caching problem is a generalisation of the caching problem presented in the chapter on memory management. World-wide-web browsers use 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 space 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. It differs from the standard paging problem in the fact 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 and 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 space to put it into the cache, then the algorithm has to delete some pages from the cache to make enough space 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 aim is to minimise 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 special cases many algorithms have been developed. Here we present algorithm LANDLORD which was developed by Young.

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

LANDLORD(LA, g)

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

Example 10.5 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 parameters $s(g_4) = 4$ and $c(g_4) = 4$. Therefore, there is not enough space for it in the cache, so some pages must be evicted. LANDLORD determines the value $\Delta = 1/2$ and changes the credits as follows: $cr(g_1) = 0, cr(g_2) = 1$ and $cr(g_3) = 3/2$, thus g_1 is evicted from cache LA. There is still not enough space 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 space for g_4, thus it is placed into 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 a constant B, that $\mathrm{ALG}_k(I) \le C \cdot \mathrm{OPT}_h(I) + B$ 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 off-line cost using a cache of size h. The following statement holds for algorithm LANDLORD.

Theorem 10.7 *If $h \le k$, then algorithm* LANDLORD *is $(k/(k-h+1), k, h)$-competitive.*

Proof Consider an arbitrary input sequence of pages and denote the input by I. We use the potential function technique. During the analysis of the procedure we suppose that an off-line 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 into the off-line cache by the off-line algorithm and then it is placed into LA by the on-line algorithm. We denote the set of the pages contained in the actual cache of the optimal off-line algorithm by OPT. Consider the following potential function:

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

The changes of the potential function during the different steps are as follows.

- OPT places g into its cache.

  In this case 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 page g into its cache OPT, but the page is not contained in cache LA. Therefore $s(OPT \cap LA) \leq h - s(g)$. On the other hand, this step is performed if there is not enough space 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 only evicts pages having credit 0, during this step Φ remains unchanged.

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

  The cost of Landlord is $c(g)$. On the other hand, g was not contained in 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 the credit of a page $g \in LA$ to a value between $cr(g)$ and $c(g)$.

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

Hence the potential function has the following properties..

- 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 $\Phi_f - \Phi_0 \le k \cdot \mathrm{OPT}_h(I) - (k-h+1) \cdot \textsc{Landlord}_k(I)$, 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 $(k-h+1) \cdot \textsc{Landlord}_k(I) \le k \cdot \mathrm{OPT}_h(I) + \Phi_0$, which proves that Landlord is $(k/(k-h+1), k, h)$-competitive. ∎

10.3.3. 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 of the messages in the network. 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 optimisation 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 $u(e)$ and 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 $(s_j, t_j, r_j, d_j, b_j)$ which means that to satisfy the request bandwidth r_j must be reserved on a path from s_j to t_j for time duration d_j and the benefit of serving a request is b_j. Hereafter, we assume that $d_j = \infty$, 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. Our aim is to minimise the maximum of the overload of the edges. The overload is the ratio of the total bandwidth assigned to 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 maximise the sum of the benefits of the accepted requests. We investigate this model in details. It is important to note that this is a maximisation problem thus the notion of competitiveness is used in the form defined for maximisation problems.

Below we define the exponential algorithm. We need the following notations to define and analyse the algorithm. Let P_i denote the path which is assigned to the accepted request i. Let A denote the set of requests accepted by the on-line algorithm. In this case $l_e(j) = \sum_{i \in A, i<j, e \in P_i} r_i/u(e)$ is the ratio of the total 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 to each e, which is exponential in $l_e(j)$ and chooses the path which has the minimal cost. Below we define and analyse 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. 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)

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

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

Example 10.6 Consider the network which contains vertices A, B, C, D and 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 path (A, B, D), 5/4 on path (A, C, D), 1/2 on path (B, D), 1/2 on path (A, C). The next request j is to reserve bandwidth 1/8 on some path between A and D. Therefore 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 and 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 path (A, C, D). Therefore, if $2mb_j = 8b_j \ge 1,035$, then the request is accepted and the bandwidth is reserved on path (A, C, D). Otherwise the request is rejected.

To analyse the algorithm consider an arbitrary input sequence I. Let A denote the set of the requests accepted by EXP, and A^* the set of the requests which are accepted by OPT and rejected by EXP. Furthermore let ${P_j}^*$ denote 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, which value gives the ratio of the reserved bandwidth and the available bandwidth for e at the end of the on-line algorithm. Furthermore, let $c_e(v) = \mu^{l_e(v)}$ for each e.

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

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

is valid. In this case the following statements hold.

Lemma 10.8 *The solution given by algorithm* EXP *is feasible, i.e. 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 accepted request which violates the available bandwidth on f.

The inequality $r_j/u(f) \le 1/\lg\mu$ 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

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

holds for value $C(P_j)$ used in algorithm EXP. Using the assumption on P we obtain that $\frac{r_j}{u(e)} \ge \frac{1}{P}$, and $\mu^{1-1/\lg\mu} = \mu/2$, thus from 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 10.9 *For the solution given by* OPT *the following inequality holds:*

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

Proof Since EXP rejected 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 $j \in A^*$, and 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, $c_e(j) \le c_e(v)$ holds for each e, thus we obtain that

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

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

Consequently,

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

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

Lemma 10.10 *For the solution given by algorithm* EXP *the following inequality holds:*

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

Proof It is enough to show that the inequality $\sum_{e\in P_j}(c_e(j+1)-c_e(j)) \le 2mb_j\log_2\mu$ 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 $0\le x\le 1$, and because of the assumptions $0\le \log_2\mu\frac{r_j}{u(e)}\le 1$, we obtain that

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

Summarising 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 $C(P_j)\le 2mb_j$, the above inequality proves the required statement. ■

With the help of the above lemmas we can prove the following theorem.

Theorem 10.11 *Algorithm* EXP *is $\Omega(1/\lg\mu)$-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 From Lemma 10.8 it follows that the algorithm results in a feasible solution where the available bandwidths are not violated. Using the notations defined above 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 10.9 and Lemma 10.10 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

10.3-1 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.

10.3-2 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.

10.3-3 In the web caching problem 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 off-line algorithm OPT with cache of size 6 already placed the page into its cache, so its cache contains the pages d and c. Which pages are evicted by Landlord to place d? In what way does the potential function defined in the proof of Theorem 10.7 change?

10.3-4 Prove that if in the throughput model no bounds are given for the ratios $r(j)/u(e)$, then there is no constant-competitive on-line algorithm.

10.4. On-line bin packing models

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

10.4.1. 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 minimise the number of the bins used. 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 its items is at most 1, and the goal is to minimise the number of groups. This problem appears also 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 values $a_1, \ldots, a_i$ without any information about the further items.

Algorithm Next-Fit, 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 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 into them. If only one bin can be open, then the evident algorithm packs the item into the open bin if it fits, otherwise it closes the bin, opens a new one and puts the item into it. This algorithm is called Next-Fit (NF) algorithm. We do not present the pseudocode of the algorithm, since 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 10.12 *The asymptotic competitive ratio of* NF *is 2.*

Proof Consider an arbitrary sequence of items, denote it by σ. Let n denote the number of bins used by OPT and m the number of bins used by NF. Furthermore, let S_i, $i = 1, \ldots, m$ denote 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 off-line 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 the following sequence for each n 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 $1/(4n-2)$, $i = 1, \ldots, 2n-1$. 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 off-line 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 ∞, we proved that the asymptotic competitive ratio of the algorithm is at least 2. ■

If $k > 1$, then there are better algorithms than NF 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 runs several NF algorithms simultaneously; each for the items of a certain type.

Algorithm First-Fit and the weight function technique

In this section we present the weight function technique which is often used in the analysis of the bin packing algorithms. We show this method by analysing algorithm First-Fit (FF).

Algorithm FF can be used when the number of open bins is not bounded. The algorithm puts the item into the first 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 10.13 FF *is asymptotically 1.7-competitive.*

Proof In the proof we use the weight function technique whose idea is that a weight is assigned to each item to measure in some way how difficult it can be to pack the certain item. The weight function and the total size of the items are used to bound the off-line and on-line objective function values. We use the following weight function:

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

Let $w(H) = \sum_{i \in H} w(a_i)$ for any set H of items. The properties of the weight function are summarised in the following two lemmas. 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 we omit them.

Lemma 10.14 *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 10.15 *For an arbitrary list L of items $w(L) \ge \mathrm{FF}(L) - 2$.*

Using these lemmas we can prove that the algorithm is asymptotically 1.7-competitive. Consider an arbitrary list L of items. The optimal off-line 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 from Lemma 10.14 it follows that $w(L) \le 1.7\mathrm{OPT}(L)$. On the other hand considering Lemma 10.15 we obtain that $\mathrm{FF}(L) - 2 \le w(L)$, which yields that $FF(L) \le 1.7\mathrm{OPT}(L) + 2$, and this inequality proves that the algorithm is asymptotically 1.7-competitive. ■

It is important to note that the bound is tight, i.e. it is also true that the asymptotic competitive ratio of FF is 1.7. Many algorithms have been developed with smaller asymptotic competitive ratio than 17/10, the best algorithm known at present time 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 10.16 *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 the number of bins containing two items by k. In this case the number of the used bins is $A(L_1) = k + n - 2k = n - k$. On the other hand, the optimal off-line algorithm can pack pairs of items into the bins, thus $\mathrm{OPT}(L_1) = \lceil n/2 \rceil$.

Now suppose that the input is the combined list L_1L_2. The algorithm is an on-line algorithm, therefore it does not know whether the input is L_1 or L_1L_2 at the beginning, 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 and the other ones need separate bin. Thus $\mathrm{A}(L_1L_2) \geq n-k+(n-(n-2k)) = n+k$. On the other hand, the optimal off-line algorithm can pack a smaller (size $1/3+\varepsilon$) item and a larger (size $1/2+\varepsilon$) item into each bin, thus $\mathrm{OPT}(L_1L_2) = n$.

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

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

Moreover for the above constructed lists $\mathrm{OPT}(L)$ is at least $\lceil n/2 \rceil$, which can be arbitrarily great. 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 (in this proof L_1L_2) is considered, and depending on the behaviour of the algorithm a prefix of the sequence is selected as input for which the ratio of the costs is maximal. It is an evident extension to consider more difficult sequences. Many lower bounds have been proven based on different sequences. On the other hand, the computations which are necessary to analyse the sequence have become more and more difficult. Below we show how the analysis of such sequences can be interpreted as a 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 an algorithm for which the technique can achieve the minimal lower bound, thus our aim 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 problem we need the following definitions.

The contain of a bin can be described by the packing pattern of the bin, which gives that how many elements are contained in the bin from each subsequence. Formally, a ***packing pattern*** is a k-dimensional vector $(p_1,\ldots,p_k)$, where 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 one. (Pattern p belongs to class T_j if $p_i = 0$ for each $i<j$ 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 the number of bins which are packed by the pattern p by $n(p)$. The packing produced by the algorithm is given by variables $n(p)$.

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

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

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

Min R

$$\begin{aligned}
&\textstyle\sum_{p \in T} p_j n(p) = n_j, && 1 \le j \le k, \\
&\textstyle\sum_{i=1}^{j} \sum_{p \in T_i} n(p) \le R \cdot \mathrm{OPT}(L_1 \dots L_j), && 1 \le j \le k, \\
&n(p) \in \{0, 1, \dots\}, && p \in T.
\end{aligned}$$

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 off-line costs for the subsequences considered.

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

In this 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.

10.4.2. Multidimensional models

The bin packing problem has three different multidimensional generalisations: the vector packing, the box packing and the strip packing models. We consider only the strip packing problem in details. For the other generalisations we give only 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 into a vertical strip of width w without rotation minimising the total height of the strip. We assume that the widths of the rectangles is at most w and the heights of the rectangles is at most 1. This problem appears in many situations. Usually, scheduling of tasks with shared resource involves two dimensions, namely the resource and the time. We can consider the widths as the resources and the heights as the times. Our goal is to minimise 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 each 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

A basic way of packing into the strip is to define shelves and pack the rectangles onto the shelves. By ***shelf*** we mean a rectangular part of the strip. Shelf packing algorithms place each rectangle onto one of the shelves. If the algorithm decides which shelf will contain the rectangle, then the rectangle is placed onto the shelf as much to the left as it is possible without overlapping the other rectangles placed onto the shelf earlier. 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 which shelf to put the rectangle onto. Hereafter we will say that it is possible to pack a rectangle onto a shelf, if there is enough room for the rectangle on the shelf. It is obvious that if a rectangle is higher than a shelf, we cannot place it onto the shelf.

We consider only one algorithm in details. This algorithm was developed and analysed by Baker and Schwarz in 1983 and it is called Next-Fit-Shelf (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 define how the algorithm works below.

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

Example 10.7 Let $r = 1/2$. Suppose that the size of the first item is $(w/2, 3/4)$. Therefore, it is assigned to a shelf of height 1. We define a shelf of height 1 at the bottom of the strip;

this 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 $(3w/4, 1/4)$. In this case it is placed onto 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 onto its left corner. Suppose that the size of the next item is $(3w/4, 5/8)$. This item is placed onto a shelf of height 1. It is not possible to pack it onto 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 $(w/8, 3/16)$. This item is placed onto a shelf of height 1/4. We can pack it onto the active shelf of height 1/4, thus we pack it onto that shelf as left as it is possible.

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

Theorem 10.17 *Algorithm* NFS_r *is* $\left(\frac{2}{r} + \frac{1}{r(1-r)}\right)$*-competitive. Algorithm* NFS_r *is asymptotically* $2/r$*-competitive.*

Proof First we prove that the algorithm is $\left(\frac{2}{r} + \frac{1}{r(1-r)}\right)$-competitive. Consider an arbitrary list of rectangles and 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 ($h = r^j$ for some j), 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,

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

Consider the shelves which are not active at the end. Consider the shelves of height hr^i for each i, and 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 onto 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 onto S and S' is at least hwr^{i+1}. If we pair the shelves of height hr^i 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 the shelves of height hr^i is at least $wn_i hr^{i+1}/2$. Thus the total area of the rectangles is at least $\sum_{i=0}^{\infty} wn_i hr^{i+1}/2$, and this yields that $\mathrm{OPT}(L) \ge \sum_{i=0}^{\infty} n_i hr^{i+1}/2$. On the other hand, the total height of the closed shelves is $H_Z = \sum_{i=0}^{\infty} n_i hr^i$, and we obtain that $H_Z \le 2\mathrm{OPT}(L)/r$.

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

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

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

Besides this algorithm some other shelf algorithms have been 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 an evident idea to use other bin packing algorithms. The best shelf algorithm known at present time 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

10.4-1 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.
10.4-2 Suppose that the size of the items is bounded above by 1/3. Prove Lemma 10.15 under this assumption.
10.4-3 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 class T_2?

10.4-4 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 the same height as the shelf which is chosen to pack them onto. Prove that this algorithm is $2 + \frac{1}{r(1-r)}$-competitive.

10.5. On-line scheduling

The area of scheduling theory has a huge literature. The first result in on-line scheduling belongs to Graham, who analysed the List scheduling algorithm in 1966. We can say that despite of 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 analysed 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. We consider the parallel machines case, where m machines are given, and we can use them to schedule the 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 have 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 vector $P_j = (p_j(1), \ldots, p_j(m))$, 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, but here we consider only such models where the goal is the minimisation 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.

10.5.1. On-line scheduling models

Probably the following models are the most fundamental examples of on-line 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 on-line algorithm one by one. When a job is revealed, the on-line algorithm has to assign the job to a machine with a starting time and a completion time irrevocably.

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 minimise 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, i.e. the algorithm has to assign the jobs to the machines minimising the maximum load, which is the makespan in this case.

Example 10.8 Consider the LIST model and two identical machines. Consider the following sequence of jobs where the jobs are given by their processing time: $I = \{j_1 = 4, j_2 = 3, j_3 = 2, j_4 = 5\}$. The on-line algorithm first receives job j_1 from the list, and the algorithm has to assign this job to one of the machines. Suppose that the job is assigned to machine M_1. After that the on-line algorithm receives job j_2 from the list, and the algorithm has to assign this job to one of the machines. Suppose that the job is assigned to machine M_2. After that the on-line algorithm receives job j_3 from the list, and 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, and 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 time intervals $(0, 4)$ and $(4, 9)$, and we can schedule the jobs on the second machine in time intervals $(0, 3)$ and $(3, 5)$.

TIME model

In this model there are a fixed number of machines again. Each job has a processing time and a ***release time.*** A job is revealed to the on-line algorithm at its release time. For each job the on-line algorithm has to 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 assign a job immediately at its release time.

However, if the on-line 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. Our aim is to minimise the makespan.

Example 10.9 Consider the TIME model with two related machines. Let M_1 be the first machine with speed 1, and M_2 be the second machine 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, and the algorithm can either start to process it on one of the machines or 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, and 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, and 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.

10.5.2. LIST model

The first algorithm in this model has been 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 10.18 *The competitive ratio of algorithm* LIST *is* $2 - 1/m$ *in the case of identical machines.*

Proof First we prove that the algorithm is $2 - 1/m$-competitive. Consider an arbitrary input sequence denoted by $\sigma = \{j_1, \ldots, j_n\}$, and 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

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

This yields that

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

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

OPT, thus $\mathrm{OPT}(\sigma) \geq p_l$. Due to these bounds we obtain that

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

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. 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 $2 - 1/m$. ■

Although it is hard to imagine any other algorithm for the on-line case, many other algorithms have been developed. The competitive ratios of the better algorithms tend to smaller numbers 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 to LIST, the loads are kept low on some of the machines, keeping the possibility of using 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, and the processing time of a job can be very large on the machine where the actual load is minimal. However, 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 several machines which have minimal value, then the algorithm chooses the machine where the processing time of the job is minimal from them, if there are several machines with this property, the algorithm chooses the one with the smallest index from them.

Example 10.10 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. 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, and 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 10.11 Consider the case of unrelated machines 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. 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. 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 10.19 *The competitive ratio of algorithm* GREEDY *is m in the case of unrelated machines.*

Proof First we prove that the competitive ratio of the algorithm is at least m. Consider the following input sequence. Let $\varepsilon > 0$ be an arbitrarily 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 ($p_i(i-1) = 1+\varepsilon$, $p_i(i) = i$, $p_i(j) = \infty$, if $j \neq i-1$ and $j \neq i$).

In this case job j_i is scheduled on M_i by GREEDY and the makespan is m. On the other hand, the optimal off-line 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 off-line 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 off-line optimal schedule, 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, the statement is obviously true for $k = 1$. Let $1 \leq k < n$ be an arbitrary number 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 that the load on this machines is at most $L(k) + p_{k+1}(l) \leq \sum_{i=1}^{k+1} \min_j p_i(j)$ from the induction hypothesis.

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 case of the related machines 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 10.20 *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 follows immediately, i.e. 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 10.21 *If the loads are at least l on all machines having a speed of at least v then the loads are at least $l - 4L^*$ on all machines having a speed of 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 a speed of 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 a speed of at least v. This yields that there exists a job among them which is assigned by OPT to a machine having a speed of smaller than v (otherwise the optimal off-line makespan would be larger than L^*). Let J be such a job.

Since OPT schedules J on a machine having a speed of 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 a speed of 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 a speed of at most $v/2$ (otherwise Greedy would assign J to one of these machines). ■

Now we can prove the following statement.

Theorem 10.22 *The competitive ratio of algorithm* Greedy *is $\Theta(\lg m)$ in the case of the related machines.*

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 10.20 the load on this machine is at least $L - L^*$. Then using Lemma 10.21 we obtain that the loads are at least $L - L^* - 4iL^*$ on the machines having a speed of at least $v_{\max}2^{-i}$. Therefore the loads are at least $L - (1 + 4\lceil \lg m \rceil)L^*$ on the machines having a speed of at least $v_{\max}/m$. Denote the set of the machines having a speed of at most $v_{\max}/m$ by I.

Denote the sum of the processing weights of the jobs by W. 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

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

which yields that $L \le 3 + 4\lceil \lg m \rceil)L^*$, 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 and 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 processing weight 2^{-i} which can be scheduled during 1 time unit 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 $i \ge 1$, 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 off-line 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 off-line 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 1 time unit, and it takes also 1 time unit to process these jobs on the machines of G_k. Thus GREEDY schedules these jobs on the machines of $G_0, \ldots, G_{k-1}$, and each load is 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 $k = \Omega(\lg m)$, we proved the required statement. ■

10.5.3. TIME model

In this model we investigate only one algorithm. The basic idea is to divide the jobs into groups by the release time and to use an optimal off-line algorithm to schedule the jobs of 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 $i = 0$. The algorithm is defined by the following pseudocode:

INTV(I)

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

Example 10.12 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 off-line algorithm schedules j_1 on machine M_1 and j_2, j_3 on machine M_2, so the jobs are completed at time 1. Since no new job have been released in the time interval $(0, 1]$, the algorithm waits for a new job until 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 has been 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 10.23 *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 off-line cost. In this case $T_2 \leq T_{\text{OPT}}$. This inequality is obvious if $t_{i+1} \neq q_i$. If $t_{i+1} = q_i$, then the inequality holds, because also the optimal off-line algorithm has to schedule the jobs which are scheduled in the i-th iteration by INTV, and INTV uses an optimal off-line schedule in each iteration. On the other hand, $T_1 + T_3 \leq T_{\text{OPT}}$. 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 also the optimal algorithm has to schedule these jobs after time T_1. On the the other hand, it takes at least T_3 time units to process these jobs, because INTV uses optimal off-line algorithm in the iterations. The makespan of the schedule produced by INTV is $T_1 + T_2 + T_3$, and we have shown that $T_1 + T_2 + T_3 \leq 2T_{\text{OPT}}$, thus we proved that the algorithm is 2-competitive. ■

Some other algorithms have also been 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 each 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 10.24 *The competitive ratio of any on-line algorithm is at least* 1.3473 *in the TIME model for minimising the makespan.*

Proof Let $\alpha \approx 0.3473$ 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 finished and $\mathrm{ALG}(I)/\mathrm{OPT}(I) > 1+\alpha$, which proves the statement. So we can suppose that $S_1 \le \alpha$.

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 $S_2 \le S_1 + 1 - \alpha/(1-\alpha)$, then we end the sequence with $m-1$ jobs having release time S_2, and processing time $1 + \alpha/(1-\alpha) - S_2$. In this case an optimal off-line 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 $\mathrm{ALG}(I) \ge 1 + 2\alpha/(1-\alpha)$ in this case, which yields that the competitive ratio of the algorithm is at least $1+\alpha$. Therefore we can suppose that $S_2 > S_1 + 1 - \alpha/(1-\alpha)$.

At time $S_1 + 1 - \alpha/(1-\alpha)$ further $m-2$ jobs arrive with processing times $\alpha/(1-\alpha)$ and one job with processing time $1 - \alpha/(1-\alpha)$. The optimal off-line algorithm schedules the second and the last jobs on the same machine, and the other jobs are scheduled one by one 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, 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 $S_1 \le \alpha$, 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

10.5-1 Prove that the competitive ratio is at least $3/2$ for any on-line algorithm in the case of two identical machines.

10.5-2 Prove that LIST is not constant competitive in the case of unrelated machines.

10.5-3 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) instead of the optimal off-line schedule in each step is $2c$-competitive.

Problems

10-1 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.)

Analyse 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 algorithm.) Prove that the algorithm is k-competitive.

10-2* 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.

10-3 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.

10-4 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.

10-5 On-line* LPT *algorithm

Consider the algorithm in the TIME model which starts the longest released job to schedule at each 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 [7, 19].

The first results about the k-server problem (Theorems 10.1 and 10.2) are published by Manasse, McGeoch and Sleator in [33]. The presented algorithm for the line (Theorem 10.3) was developed by Chrobak, Karloff, Payne and Viswanathan (see [11]). Later Chrobak and Larmore extended the algorithm for trees in [9]. The first constant-competitive algorithm for the general problem was developed by Fiat, Rabani and Ravid (see [18]). The best known algorithm is based on the work function technique. The first work function algorithm for the problem was developed by Chrobak and Larmore in [10]. Koutsoupias and Papadimitriou have proven that the work function algorithm is $2k-1$-competitive in [31].

The first mathematical model for the data acknowledgement problem and the first results (Theorems 10.5 and 10.6) are presented by Dooly, Goldman, and Scott in [15]. Online algorithms with lookahead property are presented in [24].. Albers and Bals considered a different objective function in [1]. Karlin Kenyon and Randall investigated randomised algorithms for the data acknowledgement problem in [30]. The Landlord algorithm was developed by Young in [39]. The detailed description of the results in the area of on-line routing can be found in the survey [32] 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 [13]). The algorithms NF and FF are analysed with competitive analysis by Johnson, Demers, Ullman, Garey and Graham in [28, 29], further results can be found in the PhD thesis of Johnson ([27]). Our Theorem 10.12 is a special case of Theorem 1 in [26] and Theorem 10.13 is a special case of Theorems 5.8 and 5.9 of the book [12] and Corollary 20.13 in the twentieth chapter of this book [6]. Van Vliet applied the packing patterns to prove lower bounds for the possible competitive ratios in [37, 40]. For the on-line strip packing problem algorithm NFS_r was developed and analysed by Baker and Schwarz in [5]. Later further shelf packing algorithms were developed, the best shelf packing algorithm for the strip packing problem was developed by Csirik and Woeginger in [14].

A detailed survey about the results in the area of on-line scheduling was written by Sgall ([34]). The first on-line result is the analysis of algorithm LIST, it was published by Grahamin [21]. Many further algorithms were developed and analysed for the case of identical machines, the algorithm with smallest competitive ratio (tends to 1.9201 as the number of machines tends to ∞) was developed by Fleischer and Wahl in [20]. The lower bound for the competitive ratio of Greedy in the related machines model was proved by Cho and Sahni in [8]. 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 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 10.23 are based on the results of Shmoys, Wein and Williamson (see [35]). A detailed description of further results (on-line LPT, lower bounds) in the area TIME model can be found in the PhD thesis of Vestjens [41]. We presented only the most fundamental on-line scheduling models in the chapter, although an interesting model has been developed recently, where the number of the machines is not fixed, and the algorithm is allowed to purchase machines. The model is investigated in papers [16, 17, 23, 25].

Problem 10-1 is based on [36], Problem 10-2 is based on [15], Problem 10-3 is based on [38], Problem 10-4 is based on [22] and Problem 10-5 is based on [41].

Bibliography

[1] S. Albers, H. Bals. Dynamic TCP acknowledgement, penalizing long delays. In *Proceedings of the 25th ACM-SIAM Symposium on Discrete Algorithms*, pages 47–55, 2003. 464

[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. 464, 465

[3] B. Awerbuch, Y. Azar S. Plotkin. Throughput-competitive online routing. In *Proceedings of the 34th Annual Symposium on Foundations of Computer Science*, pages 32–40, 1993. 464

[4] Y. Azar. On-line load balancing. Lecture Notes in Computer Science, Vol. 1442. Springer-Verlag, pages 178–195, 1998. 465

[5] B. S. Baker, J. S. Schwartz. Shelf algorithms for two dimensional packing problems. *SIAM Journal on Computing*, 12:508–525, 1983. 465

[6] Á. Balogh, A. Iványi. Memory management. In A. Iványi (Ed.) *Algorithms of Informatics*. Volume 2, pages 799–850. Mondat Kiadó, 2007. 465

[7] A. Borodin R. El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998. 464

[8] Y. Cho, S. Sahni. Bounds for list schedules on uniform processors. *SIAM Journal on Computing*, 9(1):91–103, 1980. 465

[9] M. Chrobak, L. Larmore. An optimal algorithm for k-servers on trees. *SIAM Journal on Computing*, 20:144–148, 1991. 464

[10] M. Chrobak, L. Larmore. The server problem and on-line games. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 7, pages 11-64. American Mathematical Society, 1992. 464

[11] M. Chrobak, H. J. Karloff, T. Payne, S. Vishwanathan. New results on the server problem. *SIAM Journal on Discrete Mathematics*, 4:172–181, 1991. 464

[12] E. Coffman. *Computer and Job Shop Scheduling*. John Wiley & Sons, 1976. 465

[13] J. Csirik, G. Woeginger. On-line packing and covering problems. Lecture Notes in Computer Science, Vol. 1442, pages 147–177. Springer-Verlag, 1998. 464

[14] J. Csirik, G. J. Woeginger. Shelf algorithms for on-line strip packing. *Information Processing Letters*, 63:171–175, 1997. 465

[15] 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. 464, 465

[16] Gy. Dósa, Y. He. Better online algorithms for scheduling with machine cost. *SIAM Journal on Computing*, 33(5):1035–1051, 2004. 465

[17] Gy. Dósa, Z. Tan. New upper and lower bounds for online scheduling with machine cost. *Discrete Optimization*, 7(3):125–135, 2010. 465

[18] A. Fiat, Y. Rabani, Y. Ravid. Competitive k-server algorithms. *Journal of Computer and System Sciences*, 48:410–428, 1994. 464

[19] A. Fiat, G. Woeginger (Eds.). *Online Algorithms. The State of Art.* Springer-Verlag, 1998. 464

[20] R. Fleischer, M. Wahl. On-line scheduling revisited. *Journal of Scheduling*, 3(6):343–353, 2000. 465

[21] R. L. Graham. Bounds for certain multiprocessor anomalies. *The Bell System Technical Journal*, 45:1563–1581, 1966. 465

[22] Cs. Imreh. Online strip packing with modifiable boxes. *Operations Research Letters*, 66:79–86, 2001. 465

[23] Cs. Imreh. Online scheduling with general machine cost functions. *Discrete Applied Mathematics*, 157(9):2070–2077, 2009. 465

[24] Cs. Imreh, T. Németh. On time lookahead algorithms for the online data acknowledgement problem. In *Proceedings of MFCS 2007* 32nd International Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science, Vol. 4708. Springer-Verlag, 2007, pages 288–297. 464

[25] Cs. Imreh, J. Noga. Scheduling with machine cost. In *Proceedings of APPROX'99*, Lecture Notes in Computer Science, Vol. 1540, pages 168–176, 1999. 465

[26] A. Iványi. Performance bounds for simple bin packing algorithms. *Annales Universitatis Scientiarum Budapestinensis de Rolando Eötvös Nominatae, Sectio Computarorica*, 5:77–82, 1984. 465

[27] D. S. Johnson. *Near-Optimal Bin Packing Algorithms.* PhD thesis, MIT Department of Mathematics, 1973. 465

[28] D. S. Johnson. Fast algorithms for bin packing. *Journal of Computer and System Sciences*, 8:272–314, 1974. 464

[29] 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. 464

[30] 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 pages, 2001. 464

[31] E. Koutsoupias, C. Papadimitriou. On the k-server conjecture. *Journal of the ACM*, 42:971–983, 1995. 464

[32] S. Leonardi. On-line network routing. Lecture Notes in Computer Science, Vol. 1442, pages 242–267. Springer-Verlag, 1998. 464

[33] M. Manasse, L. McGeoch, D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11:208–230, 1990. 464

[34] J. Sgall. On-line scheduling. Lecture Notes in Computer Science, Vol. 1442, pages 196–231. Springer-Verlag, 1998. 465

[35] D. B. Shmoys, J. Wein, D. P. Williamson. Scheduling parallel machines online. *SIAM Journal on Computing*, 24:1313–1331, 1995. 465

[36] D. Sleator R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985. 465

[37] A. van Vliet. An improved lower bound for on-line bin packing algorithms. *Information Processing Letters*, 43:277–284, 1992. 465

[38] A. C. C. Yao. New algorithms for bin packing. *Journal of the ACM*, 27:207–227, 1980. 465

[39] N. Young. On-line file caching. *Algorithmica*, 33:371–383, 2002. 464

[40] A. van Vliet. *Lower and upper bounds for on-line bin packing and scheduling heuristics.* PhD thesis, Erasmus University, Rotterdam, 1995. 465

[41] A. Vestjens. *On-line machine scheduling.* PhD thesis, Eindhoven University of Technology, 1997. 465

This bibliography is made by HBibTEX. First key of the sorting is the name of the authors (first author, second author etc.), second key is the year of publication, third key is the title of the document.

Underlying shows that the electronic version of the bibliography on the homepage of the book contains a link to the corresponding address.

Index

This index uses the following conventions. Numbers are alphabetised as if spelled out; for example, “2-3-4-tree" is indexed as if were “two-three-four-tree". When an entry refers to a place other than the main text, the page number is followed by a tag: *ex* for exercise, *exa* for example, *fig* for figure, *pr* for problem and *fn* for footnote.
The numbers of pages containing a definition are printed in *italic* font, e.g.

time complexity, *583*.

Name Index

This index uses the following conventions. If we know the full name of a cited person, then we print it. If the cited person is not living, and we know the correct data, then we print also the year of her/his birth and death.