CSC 2421 - Approximation Algorithms

Lecture 7 - 25th February 2003

Lecturer: A. Borodin

Scribe notes by: A. Dimitromanolakis, K. Hoetmer

1 Scheduling Problems

Scheduling problems involve the allocation of scarce resources (like processors) to activities (like processing jobs). Due to the wide applicability of this framework, thousands of scheduling problems have been examined in the literature. In fact, various different courses can be easily taught entirely on different aspects scheduling theory.

Some elements are common to many of the scheduling problems considered. We suppose that we have:

- a set J of n jobs to be scheduled numbered $1, 2, \ldots, n$. Each job i has processing time p_i which means that it requires p_i units of time to finish after it has been scheduled to a specific processor.

- a set M of m identical, related or unrelated machines where jobs can be scheduled. Each machine can be running only 1 job at any specific time or else be idle. In non-preemptive scheduling, once we decide to schedule a job to a specific machine, we cannot decide later to switch the job to another machine (or to not schedule the job).

- possibly some constraints on the scheduling of the jobs. We can have precedence (partial order) constraints over two jobs, written as $p_i \prec p_j$ and meaning that job j may be scheduled only after job i has finished. We may also have deadline, release time and various other sorts of constraints required for specific problems.

- an optimality criterion for the solution that we are seeking. The solution is a schedule $\mathcal{S}$ that assigns each job to a specific processor at some specific time. We define $C_j^{\mathcal{S}}$ to be the time that job j completes under schedule $\mathcal{S}$.

The notation that we'll use was introduced by Graham et al. in their 1979 paper [1] and is now a standard in the area (although it is a bit obscure at first). A problem is denoted by $\alpha|\beta|\gamma$ where

- α denotes the machine environment used. It is 1 for the one machine environment or m when scheduling generally on m machines. The machine model can also be P , Q or R. P denotes the *identical parallel machine* model where all processors are identical. Q is the *uniformly related machines* environment, where each machine has a speed s_i and thus job j would take time p_j/s_i on this machine. And finally R is the *unrelated parallel machines* environment, where the performance of the job i on machine j is taken to be p_{ij}, which may have any value.

- β denotes the constraints of the scheduling. It can be empty, meaning that no additional constraints are imposed or can be "prec." meaning job precedence is specified, d_j meaning deadline times for each job are supplied, r_j meaning release times are supplied, etc. The deadline constraint is usually not stated but implied by the objective function.

- γ denotes the goal criterion of the schedule.

For example $1|\ |maxC_i$ means we are scheduling on one processor, trying to minimize the maximum completion time over all the jobs. All jobs must ne scheduled and there are no other constraints.

We will be concerned mainly with three major classes of problems: makespan problems, time constrained scheduling of jobs to maximize profit and minimization of average (or equivalently total) weighted completion problems. There are literally hundreds of variations of these problems, some of which can be solved efficiently and some not.

The three basic problem families are:

- **Makespan:** $Pm|\ |maxC_i$

  In makespan problems we have m identical processors and n jobs with processing times p_i to be scheduled. The schedule $\mathcal{S}$ is now a mapping $\Pi : \{1, \ldots, n\} \rightarrow \{1, \ldots, m\}$ since the order of the jobs executed on a specific processor does not matter. The makespan $C^{\mathcal{S}}_{max}$ of a particular schedule is the maximum completion time over all jobs for this schedule. The goal is to minimize the makespan:

$$\min_{\mathcal{S}} C^{\mathcal{S}}_{max} = \min_{\mathcal{S}}\{\max_{1 \leq j \leq n} C_j\}\}$$

- **Time constrained scheduling to maximize profit:** $P|\ |\sum w_j \overline{\mathcal{U}_j}$

  In this variation, each job may (or may not) have one of the following: release time r_j, deadline d_j, processing time p_j and profit w_j. We are not required to schedule all the jobs and of interest here is to maximize our profit: $\max_{\mathcal{S}} \sum w_j \overline{\mathcal{U}_j}$ where $\overline{\mathcal{U}_j}$ is 1 if job j was selected in this schedule.

  Variations of this problem can be solved in polynomial time. For example **ISP** (Interval Scheduling Problem) where $p_j = d_j - r_j$ belongs to **P**. The jobs here are intervals $\{d_j, r_j\}$. Also there can be weight (profit) for each job (**WISP** problem).

- **Minimize average or total weighted completion time:**
  $P1|prec|\sum w_j C_j$

  Here each job has an associated weight and the goal now is to minimize the average total weighted completion time.

2 Makespan: $Pm|\,|maxC_i$

The makespan problem was the first to be studied in approximation algorithms by Graham's 1966 and 1969 papers [2, 3]. The problem is already NP-complete when m equals 2. Suppose we have a set $\mathcal{K} = \{k_i\}$ of integers and wish to find if they can be partitioned into two sets S_1, S_2 with $\sum_{s \in S_1} s = \sum_{s \in S_2} s$, which is known to be NP-complete. Create an instance of makespan with 2 processors and a job for each element of the set K with processing time $p_i = k_i$. The original problem has a solution iff the optimal schedule completes at time $\frac{1}{2}\sum k_i$.

Variations of the problem can include additional constraints like $P_j(i) \in \{p_j, \infty\}$ meaning that each job can be scheduled only to a specific subset of the processors. The model $R|p_j(i) \in \{p_j, \infty\}|\max C_i$ will be denoted as $S|\,|\max C_i$.

2.1 LS Greedy Algorithms for Makespan

For makespan, "any" greedy algorithm achieves a tight approximation ratio of $2 - \frac{1}{m}$ even with precedence constraints. Surprisingly enough, greedy essentially achieves the best known approximation ratio when precedence is part of the problem.

LS (List Scheduling) Greedy Algorithm for Makespan: *At each given point of time, take a job that can be scheduled and schedule it at some machine having the least current makespan.* More generally, a greedy algorithm can schedule the job on any machine that will minimize the makespan after the job has been scheduled. The basic LS algorithm ("the greedy algorithm") is an online algorithm that cosnduers the jobs in any order.

Claim: The greedy algorithm achieves an $2 - \frac{1}{m}$ approximation ratio even with precedence.

Proof: Suppose greedy has found a schedule and let

$J_{j(1)}$: to be a job that has maximum completion time.
$J_{j(2)}$: to be a predecessor of $J_{j(1)}$ completing last among all predecessors.
$\vdots$
$J_{j(i+1)}$: to be a predecessor of $J_{j(i)}$ completing last among all predecessors.

Define sets

$$\begin{aligned} A &= \{t \mid \text{some } J_{j(i)} \text{ is scheduled}\} \\ B &= \{t \mid t \le \max C_i \text{ and } t \notin A\} \end{aligned}$$

Notice that for each time $t \in B$ all machines are busy. If it was not the case then by the greedy condition we could have scheduled something earlier. We note that:

$$\sum_{j=1}^{n} p_j = m|B| + |A|$$

and

$$OPT \geq \frac{\sum p_j}{m} \geq |B| + \frac{|A|}{m}$$

Also note that $OPT \geq |A|$. The bound follows:

$$(2 - \frac{1}{m})OPT = (1 - \frac{1}{m})OPT + OPT \geq (1 - \frac{1}{m})|A| + |B| + \frac{|A|}{m} = |A| + |B|$$

which is the length of the schedule that the greedy algorithm produced.

□

Claim: There is an instance that forces the $2 - \frac{1}{m}$ ratio for the greedy algorithm.

Proof: Take the case where the greedy algorithm is presented with $m(m-1)$ small jobs with processing time 1 and one large job with $p_j = m$. Moreover the jobs are in the following order: $(1, 1, \ldots, 1, m)$. Then $\frac{GREEDY}{OPT} = \frac{2m-1}{m}$

□

2.2 LPT Greedy Algorithm for Makespan

The above negative example motivates the idea that we use the biggest job first heuristic. This leads to the Longest Processing Time (LPT) greedy algorithm:

LPT Greedy Algorithm: sort jobs in non-increasing order of processing time and list schedule in that order.

Claim: LPT has approximation ratio upper bound of $\frac{4}{3} - \frac{1}{3m}$ for the model $P| \,| \max C_i$.

Proof Sketch:

1. If $\frac{LPT}{OPT} > \frac{4}{3} - \frac{1}{3m}$ can show OPT schedules at most 2 jobs per machine.
2. $OPT \geq \frac{\sum p_j}{m}$. Renumber the jobs in non-increasing order of p_j. Say job j_n is scheduled at time t. Then $p_1 \geq p_2 \geq \ldots \geq p_n$.
3. If $n \leq m$, OPT puts one job at each machine and so does LPT.
4. If OPT puts at least 2 job per machine, then we can transform OPT to LPT without increasing span.

Theorem: **LPT** has approximation ratio $\frac{2m}{m+1}$ for $P|prec.|\max C_i$ when there are precedence constraints.

Open Problem: For the model $P|prec.|\max C_i$ find any poly time algorithm with approximation ratio $2-\epsilon$, $\epsilon > 0$.

For $R|\,|C_{max}$ model, where $C_{max} = \max C_i$, a $3/2$ hardness result has been proved while an LP-based algorithm exists that achieves a 2-approximation. This algorithm is also the best known for the $S|\,|C_{max}$ model (4/3 hardness result exists).

2.3 Dynamic Programming based algorithms

For the makespan problem, we can get an optimal algorithm for $P|\,|C_{max}$ if

$$p_j \in \{z_1, \ldots, z_s\}$$

That is, there are s different processing times.

DP based decision algorithm for MakeSpan: We show first how to transform the decision version of MakeSpan the the optimization one. The optimal schedule time D is known to be between

$$\max p_i \leq D \leq \sum p_i.$$

Since the optimal solution is bounded and we are assuming that a decision algorithm is known, using binary search we can find the optimal schedule time with $\log_2 \sum p_i$ queries to the decision algorithm.

Now we describe the decision algorithm: *Is there a schedule that finishes in time $\leq D$?*

Let the different processing times be $z_1,\ldots,z_s$. Since the ordering of jobs does not matter for a specific machine, each machine's schedule can be describe using a vector

$$V = < v_1, \ldots, v_s >$$

where v_i is the number of jobs of length z_i scheduled on this machine. There are at most n_s such vectors. Let $\mathcal{V}$ be the set of feasible vectors: the ones that do not exceed the goal time D. Define

$$M(x_1, \ldots, x_s) := \min \#\text{machines needed to schedule } x_i \text{ jobs of size } z_i$$

Using the dynamic programming paradigm, the following recurrence relation for M holds:

$$M(x_1, \ldots, x_s) = 1 + \min_{\vec{N} \in \mathcal{V}} M(x_1 - N_1, \ldots, x_s - N_s)$$

There are n^s possible values in this table and each one takes time n^s to be computed by the recurrence relation, accounting for a total time of n^{2s}.

□

Definition: A *ρ-relaxed decision procedure* for a minimization problem accepts as input a target value D and returns either NO, asserting that there exists no solution of value $\leq D$, or returns a solution of value at most ρD. Note that if there does not exist a solution of value $\leq D$ then the algorithm may either return NO or a solution with value at most ρD. Using binary search again, the existance of a ρ-relaxed decision procedure implies a ρ-approximation algorithm.

Our goal is to describe an $(1+\epsilon)$-relaxed decision procedure for MakeSpan. The main idea is to schedule the big jobs optimally first and then small jobs in a greedy fashion.

ρ-relaxed decision procedure for MakeSpan: From the given instance I form a new instance I' by removing the small jobs having $p_j < \epsilon D$ and rounding the remaining jobs down to an integer multiple p'_j of $\epsilon^2 D$. We assume that e^{-1},e^{-2}, $e^1 D$ and $e^2 D$ are all integers. The proof can be easily modified (by scaling) when these are rational numbers.

The following hold:

- $0 \leq p_j - p'_j \leq \epsilon^2 D$
- there at most $\frac{D}{\epsilon^2 D} = \frac{1}{\epsilon^2}$ different job sizes.
- in a feasible schedule, each machine has at most $\frac{D}{\epsilon D} = \frac{1}{\epsilon}$ jobs.

We can schedule the instance I' optimally using the DP algorithm in polynomial time. Say the optimal schedule has length D'. If $D' > D$ we can safely return NO, since the all the job sizes of I' are no larger than those of I (and a subset of them).

If $D' \leq D$ and the makespan is $\leq D$, then a schedule of length $(1+\epsilon)D$ can be returned. Replace jobs on I' with their original time in this schedule. Schedule small jobs removed earlier on machines with load $\leq D$. If it cannot be done then answer NO (in this case we are guaranteed that scheduling in time $\leq D$ cannot be done, since all machine have load $> D$ after putting in some subset of the small jobs).

If it is successful then we created a schedule for I with makespan at most $D' + \epsilon D' = (1+\epsilon)D'$ (as there are $\frac{1}{\epsilon}$ jobs per machine and each job is increased by $\epsilon^2 D$ at most).

$\square$

2.4 IP/LP Formulation for Unrelated Makespan

The previous algorithms assumed job scheduling on a set of identical machines. For unrelated machines without precedence $(R| \,|\max C_i)$ a 2-approximation can be found using an IP/LP formulation.

The IP/LP formulation is designed such that a feasible solution for a given makespan D means the makespan problem had an optimal solution $\leq D$. This value D can be narrowed using binary search on a valid search space.

Let $x_{i,j}$ be an indicator variable such that $x_{i,j} = 1$ iff job j is scheduled on machine i. Assume also that $n > m$ so that we have more jobs than available machines. Then, we obtain the following IP:

- $\Sigma_{i=1}^{m} x_{i,j} = 1, \ 1 \leq j \leq n$
- $\Sigma_{j=1}^{n} p_{i,j} x_{i,j} \leq D, \ 1 \leq i \leq m$
- $x_{i,j} = 0, \text{ if } p_{i,j} \geq D$
- $x_{i,j} \in \{0, 1\}$

The first inequality states that each job j may be scheduled on at most 1 machine. The second inequality states that the makespan of the problem must be $\leq D$. The third constraint ensures we only attempt to schedule jobs on machines for which they can finish in time. Finally, the last constraint forces an integral solution. The LP relaxation is obtained by relaxing the last constraint to $0 \leq x_{i,j} \leq 1$.

A solution to the IP can now be used to schedule jobs by assigning job j to machine i if $x_{i,j} = 1$. Of course this IP problem is not generally polytime computable. We can, however, develop a 2-approximation to the scheduling problem by a non-trivial massaging of the fractional solution.

Claim: LP with massaging obtains a 2-approximation for $R||C_{max}$.

Proof (from Karger et. al. [4] page 32):
Note that there are $m + n$ inequalities. Therefore, by the theory of linear programming, if a feasible solution to the LP exists, there is also a "basic feasible solution" such that at most $m+n$ variables (in this case $x_{i,j}$) are positive (non-zero).

Assume we have such a "basic feasible solution". Then, since all n jobs must be scheduled, there is at least 1 positive variable for each job j. Therefore, at most m of the jobs can have more than 1 positive indicator variable. Furthermore, if a job j has only 1 positive variable $x_{i,j}$, we know that $x_{i,j} = 1$.

Then, for every variable $x_{i,j}$ such that $x_{i,j} = 1$, assign job j to machine i. Call this schedule S_1. S_1 then has a makespan $\leq D$ since the $x_{i,j}$ in S_1 form a subset of the feasible solution to the LP.

To schedule the fractional $x_{i,j}$, we build a bipartite graph G such that G has n vertices representing jobs and m vertices representing machines. We then create an edge (i, j) in G for every positive fractional $x_{i,j}$. We then construct a matching M on G, and for every edge (i, j) in M, we schedule job j on machine i. Call this schedule S_2. The matching has forced us to schedule at most 1 job per machine and since j is assigned to i only if $x_{i,j} > 0$, we know that $p_{i,j} < D$ for each job / machine pair in S_2. Hence the makespan of S_2 is at most D.

But S_1 and S_2 together form a schedule for the original problem, hence the LP formulation with matching produces a schedule with

$$makespan = makespan(S_1) + makespan(S_2) \leq D + D = 2D.$$

$\square$

Note: We have assumed the existence of a matching M on the graph construction G. A proof sketch of this claim is found in Karger et. al. [4] on page 32.

3 Time constrained scheduling to maximize profit: $Pm|r_j|\Sigma w_j \bar{U}_j$ and $Qm|r_j|\Sigma w_j \bar{U}_j$

This class of problems contains an extensive number of poly-time solvable cases. However, the general unit weighted problem on 1 machine $P1|r_j|\Sigma \bar{U}_j$ is strongly NP-hard.

Time constrained scheduling problems have been studied extensively by Bar Noy et. al. [5, 6] and Erlebach and Spieksma [7]. Their work enumerates a number of poly-time solvable special cases, as well as approximation bounds for many NP-hard cases. The bounds are achieved via either an earliest completion time first (ECT) greedy approach, or via an LP formulation. The following table enumerates Bar Noy et. al.'s findings (the second column represents number of machines m in the form $1, 2, 3, \ldots, m, \ldots, m \to \infty$):

Weight Function	Identical Machines	Unrelated Machines
Identical job weights (Greedy)	$(2, 1.8, 1.73, \ldots, \frac{(1+\frac{1}{m})^m}{(1+\frac{1}{m})^m - 1}, \ldots, 1.58)$	$(2, 2, 2, \ldots, 2)$
Arbitrary job weights / poly-size input (LP)	$(2, 1.8, 1.73, \ldots, \frac{(1+\frac{1}{m})^m}{(1+\frac{1}{m})^m - 1}, \ldots, 1.58)$	$(2, 3, 3, \ldots, 3)$
Arbitrary job weights / arbitrary input (LP)	$(3, 2.78, 2.7, \ldots, \frac{(1+\frac{1}{2m})^m}{(1+\frac{1}{2m})^m - 1}, \ldots, 2.54)$	$(3, 4, \ldots, 4)$

Bar Noy et. al. have also developed a one pass combinatorial algorithm (the Admission Algorithm) that achieves a $3 + 2\sqrt{2}$-approximation for arbitrary job weights and unrelated machines.

As noted before, if $p_j = d_j - r_j$ we have an instance of the Interval Scheduling Problem (ISP). It turns out that an earliest due date first (EDD) greedy approach produces a poly-time optimal solution for the unweighted version (ISP) in which each job has only 1 possible interval (proof omitted).

3.1 ECT Greedy Algorithm

ECT iteratively schedules the available job which will complete earliest.

Claim: ECT Greedy achieves a 2-approximation for $P1|r_j|\Sigma \bar{U}_j$

Proof (Bar-Noy et. al. [6]):
Let G be the set of intervals selected by the ECT greedy algorithm, and let O be the set of intervals in an optimal solution OPT. Let H be the set of jobs scheduled by OPT and not by ECT. Finally, let $OG = O \cap G$ be the set of jobs scheduled by both OPT and ECT. Then OPT $= OG \cup H$.

Now, $|H| \leq |G|$ (note since this is the unweighted case, we are simply counting the *number* of jobs scheduled) since for each interval $h \in H$, there is an interval in G that overlaps with it and finishes earlier (otherwise ECT would have chosen h), and because H is a subset of a feasible solution, each $h \in H$ can overlap with the endpoint of at most one interval in G.

But we also have $|OG| \leq |G|$ since $OG \subseteq G$. Therefore, $|\text{OPT}| = |OG \cup H| = |OG| + |H| \leq 2 \cdot |G|$.

$\square$

Claim: For m identical machines, first fit ECT (schedule machine 1, then machine 2, ...) achieves a bound of $\frac{(1+\frac{1}{m})^m}{(1+\frac{1}{m})^m - 1}$.

Proof: See Bar-Noy et. al. [6] pages 336-338.

$\square$

3.2 Bar-Noy Admission Algorithm

The Admission algorithm is a one pass combinatorial approximation that works as follows:

All intervals of all jobs are sorted from earliest to latest by their end points. Let I be this set. The algorithm maintains a set of accepted intervals A, initially set to $\emptyset$. Then, for each interval i in I, let i be the job of i. i is immediately rejected if an interval for j_i has already been accepted, and i is immediately accepted if i does not overlap any intervals in A. Otherwise, i overlaps at least one interval in A, and i is accepted if its weight is more than $\beta > 1$ times the sum of the weights of the overlapping intervals. Then, i is added to A and the overlapping intervals are removed from A. The process continues until each interval in I has been considered.

Claim: Admission achieves a $3 + 2\sqrt{2}$-approximation for $P1|r_j|\Sigma w_j \bar{U}_j$.

Proof (Bar-Noy et. al. [6]):
Let O be the set of intervals in an optimal solution OPT and let A be the set of intervals chosen by Admission. For every interval $o \in O$ that overlaps $i_1, i_2, ..., i_h \in A$, we split o into h intervals such that the weight w_j of o_j is $\frac{w_j \cdot w}{\sum_{k=1}^{h} w_k}$, the portion of o_j's weight covered by i_j and set $O = O \setminus o \cup \{o_1, o_2, ..., o_h\}$. Note that each $o \in O$ not in A must overlap some $i \in A$ or else o would have been selected by the algorithm.

Then, for each interval $i \in A$, define $R(i)$ to be the set of intervals in O that are "accounted for" by i. $R(i)$ contains i if $i \in O$ otherwise $R(i)$ contains every interval in O directly rejected or preempted by i. Then each $o \in O$ belongs to exactly one $R(i)$. Also, each i can directly reject at most one $o \in O$. This holds since each interval can only reject those which appear at a later time. Since we are performing ECT, an interval o rejected by interval i must overlap the end point of i. But since $o \in O$ is part of a feasible solution, there can only be 1 such o.

Now, choose $i \in A$. Let w be the weight of i and W be the sum weight of $R(i)$. Let $\rho = \frac{W}{w}$. Then, let w_r be the weight of $o_r \in O \cap R(i)$ that was directly rejected by i, if it exists. Then $w_r \leq \beta w$. Let $o_i \in O$ be the interval belonging to the same job as i, if it exists. Then the weight of o_i is w, the same as i. Then the sum of the rest of the intervals in $R(i) = W' \geq W - w_r - w$. Defining $\alpha = \frac{W'}{w}$, we have $\rho \leq \alpha + \beta + 1$.

Finally, assuming via induction that the ρ holds for each interval in A with earlier endpoint than the endpoint of i, we have that the weight of jobs directly preempted by i is at most w/β so that $\frac{w \cdot \rho}{\beta} \geq \alpha \cdot w$. Therefore $\frac{\alpha+\beta+1}{\beta} \geq \alpha$ so that $\alpha \leq 1 + \frac{2}{\beta - 1}$. Therefore, $\rho \leq 2 + \beta + \frac{2}{\beta-1}$ which is minimized for $\beta = 1 + \sqrt{2}$ so that $\rho \leq 3 + 2\sqrt{2} \approx 5.828$. Therefore,

$$OPT = \sum \text{ weights of all } R(i) \leq \rho \cdot \sum \text{ weights of all } i \in A$$

so that OPT is at most ρ times the weight of A.

□

Claim: For $m > 1$ unrelated machines, first-fit Admission achieves a bound of $3 + 2\sqrt{2}$.

Proof: See Bar-Noy et. al. [6] page 349.

□

Erlebach and Spieksma [7] have generalized the Admission algorithm so as to achiveve constant approximation ratios for the more general unweighted JISP and weighted WJISP problems where every job is a set of intervals and the goal is to schedule at most one interval for each job. The JISP problem is more general in the sense that the intervals that constitute a job can have different lengths (corresponding to the processing time) and different weights in the WJISP problem. On the other hand the time constrained scheduling problem cannot be simply transformed into the JISP/WJISP formulation when the size of the "windows" $d_j - r_j$ are not polynomial in the length of the input represntation. However, algorithms for the JISP can often be extended to the time constrained scheduling problem by keeping some sort of implicit representation of intervals.

References

[1] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of the Discrete Mathematics*, pages 287–326, 1979.

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

[3] R.L. Graham. Bounds on multiprocessing anomalies. *SIAM Journal of Applied Mathematics*, 17:263–269, 1969.

[4] D. Karger, C. Stein, and Joel Wein. Scheduling algorithms. In M. J. Atallah, editor, *Handbook of Algorithms and Theory of Computation*. CRC Press, 1997.

[5] Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph Naor, and Baruch Schieber. A unified approach to approximating resource allocation and scheduling. In *Proceedings of the 32nd annual ACM Symposium on Theory of Computing*, pages 735–744, 2000.

[6] A. Bar-Noy, S. Guha, J. Naor, and B. Schieber. Approximating throughput in real-time scheduling. *Siam Journal of Computing*, 31(2):331–352, 2001.

[7] T. Erlebach and F.C.R. Spieksma. Interval selection: Applications, algorithms, and lower bounds. TIK-Report Nr. 152, Swiss Federal Institute of Technology Zurich, October 2002.