Contents

9

Minmax criteria, no preemption

David B. Shmoys
Cornell University

Jan Karel Lenstra
Centrum Wiskunde & Informatica

Four authors wish to write a book, which will consist of fifteen chapters. Each chapter is to be written by a single author. The chapters differ in length, and the authors differ in expertise. The speed at which an author will be able to complete a chapter depends on his, or her, familiarity with its subject matter. The completion date of the manuscript is to be minimized. Who should do what?

Irrespective of the practical relevance of this model, it is one of the central problems in scheduling theory. In its general form, the authors are unrelated parallel machines, the chapters are jobs, and the problem is $R||C_{\max}$. The reader should recall the definitions of unrelated, uniform, and identical parallel machines (see Section 1.2). We have seen that, for these machine environments, there is a drastic difference in complexity between minimizing *maximum* and *total* completion time: $R||\sum C_j$ is solvable in polynomial time (see Section 8.1), but $P2||C_{\max}$ is already NP-hard (see Section 2.4). As long as the number of machines is a constant, dynamic programming can be applied to solve $Rm||C_{\max}$ in pseudopolynomial time (see Section 8.3); however, when m is an input, even $P||C_{\max}$ is NP-hard in the strong sense (see Section 2.4).

This chapter, then, is about hard problems. The design of efficient approximation algorithms and of enumerative optimization methods is called for. We will emphasize the performance analysis of approximation algorithms. The empirical analysis of approximation and optimization procedures will not be discussed in detail. We have chosen to summarize computational work of interest in the notes.

We start in Section 9.1 with some classical results for $P||C_{\max}$ due to R. L. Graham, whose work initiated the investigation of the worst-case performance of heuristic methods for optimization problems. More recent results on performance guarantees for identical, uniform, and unrelated machines are given in Sections 9.2, 9.3, and 9.4, respectively. Section 9.5 considers guarantees for unrelated machines which, in all likelihood, cannot be achieved.

In Section 9.6, we switch to probabilistic analysis. We will see that algorithms that can behave pretty badly in the worst case do quite well on average. From a practical point of view, the two approaches are complementary. A worst-case approach provides a performance bound for every instance, but the bound may be pessimistic, as it depends on anomalous cases that rarely occur. A probabilistic analysis has the potential to provide a more accurate picture of the real world. However, the resulting statements carry no guarantee for any individual instance, and the analysis is ultimately no more realistic than the probability distribution over the class of instances on which it is based.

9.1. Identical machines: classical performance guarantees

In Graham's ground-breaking paper, the problem studied is $P||C_{\max}$, and the method analyzed is the *list scheduling* (LS) rule: the jobs are listed in any fixed order, and whenever a machine becomes idle, the next job from the list is assigned to start processing on that machine. Surprisingly, this simple rule has a constant relative performance bound.

Theorem 9.1. *For any instance of* $P||C_{\max}$, $C_{\max}(LS)/C^*_{\max} \leq 2 - 1/m$.

Proof. Let J_ℓ be the last job to be completed in a list schedule (cf. Figure 9.1). Note that no machine can be idle before the starting time S_ℓ of J_ℓ. Intuitively, it is obvious that a list schedule is no longer than twice an optimal one. Consider two parts of the schedule, before and after S_ℓ. The optimum is at least as long as the first part, since no machine is idle between 0 and S_ℓ, and the optimum is also at least as long as the second part, since this part has the length of a single job.

More formally, the fact that no machine is idle prior to S_ℓ implies that $\sum_{j\neq\ell} p_j \geq mS_\ell$. Therefore,

$$C_{\max}(\mathrm{LS}) = S_\ell + p_\ell \leq \frac{1}{m}\sum_{j\neq\ell} p_j + p_\ell = \frac{1}{m}\sum_j p_j + \frac{m-1}{m}p_\ell. \tag{9.1}$$

By using the inequalities

$$C^*_{\max} \geq \frac{1}{m}\sum_j p_j, C^*_{\max} \geq p_\ell,$$

we obtain the claimed bound. □.

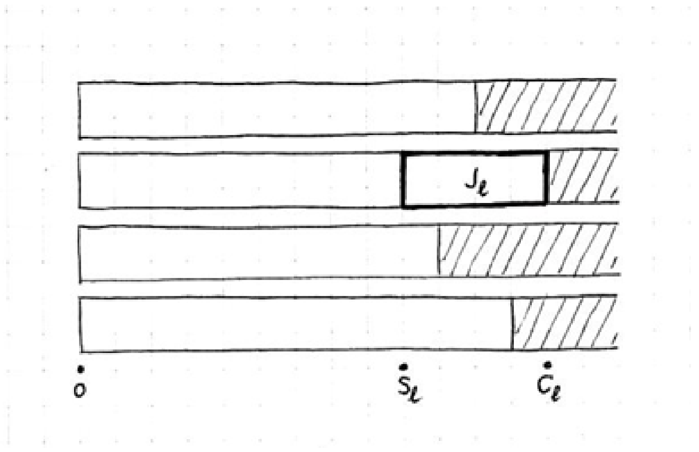

Figure 9.1. A list schedule

The bound of Theorem 9.1 is tight for any value of m, as is shown by the following class of instances. Let $n = m(m-1)+1$, $p_1 = ... = p_{n-1} = 1$, $p_n = m$, and consider the list $(J_1,...,J_n)$. The list scheduling rule first assigns $m-1$ unit-length jobs to each of the m machines and then is left scheduling J_n, so that $C_{\max}(\text{LS}) = 2m-1$. The optimal schedule has length $C^*_{\max} = m$: it assigns J_n to a machine by itself and balances the remaining $m(m-1)$ jobs among the $m-1$ other machines.

As the previous example suggests, list scheduling may perform poorly when the last job to finish is very long, and inequality (9.1) implies that this is the only way in which poor schedules are generated. A natural way to try to prevent this is to use a list in which the jobs are sorted in order of nonincreasing processing requirement. The next theorem shows that this *longest processing time* (LPT) rule performs significantly better than arbitrary list scheduling.

Theorem 9.2. *For any instance of $P||C_{\max}$, $C_{\max}(LPT)/C^*_{\max} \le 4/3 - 1/3m$.*

Proof. Suppose that the theorem is false, and consider a counterexample with the minimum number of jobs. Let job J_ℓ be the job that completes last. If $\ell < n$, then the instance consisting of jobs $\{J_1,...,J_\ell\}$ is a smaller counterexample, since the completion time of the LPT schedule is unchanged, whereas the optimal schedule length can only decrease by considering only a subset of the jobs. Hence, job J_n is the last to finish. We will consider two cases separately: (*i*) $p_n \le C^*_{\max}/3$; and (*ii*) $p_n > C^*_{\max}/3$. In case (*i*) it is easy to see that (9.1) immediately implies the theorem, since

$$C_{\max}(\text{LPT}) \le \frac{1}{m}\sum_j p_j + \frac{m-1}{m}p_\ell \le C^*_{\max} + \frac{m-1}{m}\frac{C^*_{\max}}{3} = (\frac{4}{3} - \frac{1}{3m})C^*_{\max}.$$

We will show that if case (ii) holds, then the LPT rule delivers an optimal solution. Observe that in the optimal schedule it is impossible for three jobs to be assigned to one machine, since the total processing time of those jobs is at least $3p_n$, which exceeds $C^*_{\max}$. Hence $n \leq 2m$. Furthermore, we can assume that $n > m$, since otherwise, the LPT rule assigns each machine at most one job, which is clearly optimal.

Consider the following schedule π: we first show that it is optimal, and then show that LPT must output π.

For π, we first assign job J_i to machine M_i, $i = 1,...,m$. (Of course, we can assume without loss of generality that is also true for the LPT schedule.) Furthermore, we also assign the job J_{m+i} to machine M_{m+1-i}, $j = 1,\ldots,n-m$. In other words, jobs J_j and J_{2m-j+1} are paired together on machine M_j, $j = 2m-n+1,\ldots,m$.

Suppose, for a contradiction, that this schedule is not optimal; let π' denote an optimal schedule. If $C^{\pi}_{\max} = p_j$ for some job J_j (requiring of course that $n < 2m$), then we get a contradiction directly, since p_j must be a lower bound on the makespan of any schedule. Hence $C^{\pi}_{\max} = p_j + p_{2m-j+1}$, for some $j = 1,\ldots,m$. In π', for each $k = 1,\ldots,j-1$, job J_k cannot be paired on the same machine with some job J_ℓ, where $\ell = 1,\ldots,2m-j+1$, since otherwise,

$$C^{\pi'}_{\max} \geq p_k + p_\ell \geq p_j + p_{2m-j+1} \geq C^{\pi}_{\max},$$

and hence π' is not better than π. This means that in π', the jobs $J_1,\ldots,J_{j-1}$ must all be assigned to distinct machines; index the machines so that J_i is scheduled in π' on M_i, $i = 1,\ldots,j-1$. Furthermore, we know that jobs $J_j,\ldots,J_{2m-j+1}$ are not scheduled on machines $M_1,\ldots,M_{j-1}$. Hence, there are $(2m-j+1)-(j-1) = 2(m-j+1)$ jobs scheduled on $m-j+1$; since each machine gets at most 2 jobs, we can conclude that each of the machines $M_j,\ldots,M_m$ is assigned exactly 2 jobs in π'. Consider the machine with job J_j; this machine completes at time at least $p_j + p_{2m-j+1} = C^{\pi}_{\max}$, and hence π' is not better than π. We have shown that π is optimal.

Now we want prove that the LPT rule produces π. We have indexed the machines and jobs so that the LPT schedule coincides with π for scheduling the first $m+1$ jobs. (We will assume that when scheduling jobs $J_{m+1}, J_{m+2},\ldots,$ if two machines complete their jobs at the same time, we next assign a job to the higher indexed machine.) Consider the first time that LPT schedules in a way different from π, when scheduling some job J_k, where $k > m+1$. In π, we assign job J_k to machine M_{2m-k+1}. For LPT to select a different machine for J_k, it must be that some machine M_j, $j = 2m-k+1,\ldots,m$ completes its second job no later than M_{2m-k+1} completes its first (i.e., J_{2m-k+1}). Hence,

$$C_{2m-k+1} = p_{2m-k+1} \geq p_j + p_{2m-j+1} > 2C^*_{\max}/3.$$

But then, in π, job J_k is assigned to the same machine as J_{2m-k+1}, and hence it completes in π at time $p_{2m-k+1} + p_k$, which must be greater than $C^*_{\max}$, which contradicts the optimality of π. Hence, in case (ii), the schedule produced by the LPT rule is optimal. □

Again, the bound of Theorem 9.2 is tight for any value of m; see Exercise 9.1.

Inequality (9.1) holds whenever there is no idle time prior to the start of the job that completes last. We will exploit this observation in the design of the following algorithm A_k and its analysis. Let k be a fixed positive integer. Partition the job set into two parts: the *long* jobs and the *short* jobs, where a job J_s is said to be short if $p_s \leq (1/km)\sum_j p_j$. Note that there are less than km long jobs, and hence less than m^{km} possible schedules for the long jobs. Enumerate all of these schedules, and choose the shortest one. Extend this schedule by using list scheduling for the short jobs. As in the analysis of the LPT rule, we consider the last job J_ℓ to be completed, and distinguish between two cases. If J_ℓ is a short job, then inequality (9.1) implies that

$$C_{\max}(A_k) < C^*_{\max} + p_\ell \leq C^*_{\max} + \frac{1}{km}\sum_j p_j \leq (1+\frac{1}{k})C^*_{\max}.$$

If J_ℓ is a long job, then the schedule obtained is optimal, since $C_{\max}(A_k)$ is equal to the length of the optimal schedule for just the long jobs, which is clearly no more than $C^*_{\max}$. Algorithm A_k can be implemented to run in $O((m^{km}+n)\log m)$ time, which is polynomial for any fixed k and m. We have proved the following theorem.

Theorem 9.3. *The family of algorithms $\{A_k\}$ forms a polynomial approximation scheme for $Pm||C_{\max}$.*

This result has been improved into various directions. There is even a *fully* polynomial approximation scheme for $Pm||C_{\max}$. It is based on the same ideas as the more general scheme for $Rm||C_{\max}$, which is to be described in Section 9.4; see Theorem 9.16. Further, in Section 9.2 we will give a polynomial approximation scheme for $P||C_{\max}$, where m is an input; see Corollary 9.7.

For the special case that $m = 2$, there is an algorithm with the same performance guarantee as the LPT rule (see Exercise 9.3), but with much better empirical behavior. The main idea of this *differencing* (D) method is that two jobs are assigned at a time, one to each machine. The decision as to which job is assigned to which machine is made by scheduling a smaller instance, where both jobs are replaced by a single artificial job with processing time equal to the difference of their processing times. More precisely, if there is only one job, assign it to M_1; otherwise, if J_j and J_k are the longest and second longest jobs, respectively, replace these by a new job J_ℓ with $p_\ell = p_j - p_k$ and call this procedure recursively; to convert the resulting schedule to one for the original instance, if J_ℓ is assigned to M_i, then instead assign J_j to M_i and J_k to the other machine. We will return to the differencing method in Section 9.6.

Exercises

9.1. Prove that the bound given in Theorem 9.2 is tight for each $m \geq 1$.

9.2. Use the following approach to obtain a polynomial approximation scheme similar to the scheme $\{A_k\}$ of Theorem 9.3: schedule the ℓ longest jobs optimally and

complete the schedule using list scheduling.
9.3. Prove that for any instance of $P2||C_{\max}$, $C_{\max}(D)/C^*_{\max} \leq 7/6$, and show that this bound is tight.

9.2. Identical machines: using bisection search for guarantees

The algorithms discussed in the previous section are only a first step towards understanding how to obtain good solutions for $P||C_{\max}$ efficiently. In this section, it will be useful to consider the *decision version* of $P||C_{\max}$: given a deadline d, does there exist a schedule in which all jobs are completed by time d? An equivalent way to pose this question is the following: given n items of specified sizes and m bins of capacity d each, is it possible to pack the items in the bins so that no bin contains items of total size more than d?

The problem of finding the minimum number of bins in which a given set of items can fit is called the *bin-packing problem*. For this problem, there is an approximation algorithm, called *first fit decreasing* (FFD), which resembles the LPT rule: number the bins, list the items in order of nonincreasing size, and iteratively place the next item from the list into the lowest-index bin in which it will fit. It can be shown that, if b^* is the minimum number of bins, then FFD uses at most $(11/9)b^* + 4$ bins.

The FFD procedure can also be used within an approximation algorithm for $P||C_{\max}$, which applies bisection search over the values of d. Recalling our assumption that all data are integral, we specify the *multifit* (MF) algorithm as follows:

```
$S :=$ a list schedule ;
$UB := C_{\max}(\text{LS})$;
$LB := \max\{\max_j p_j, \frac{1}{m}\sum_j p_j\}$;
while $LB \neq UB$ do
    $d := \lfloor \frac{LB+UB}{2} \rfloor$;
    run the $FFD$ algorithm to pack the jobs in bins of size $d$ (*);
    if more than m bins are used by the packing (**) then
        $LB := d+1$;
    else
        $UB := d$;;
        $S :=$ new schedule;
    end
end
```

Multifit has a better performance guarantee than is known for any variant of list scheduling.

Theorem 9.4. *When multifit is run for ℓ iterations of bisection search, then for any instance of $P||C_{\max}$, $C_{\max}(MF)/C^*_{\max} \leq 13/11 + 2^{-\ell}$.*

This bound is tight. A surprising aspect of the proof of the theorem is that it does not make use of the known analysis of the performance of FFD as a bin-packing algorithm; the two guarantees seem to be unrelated.

In the multifit algorithm, the statements indicated by (*) and (**) can be viewed as approximately answering the decision version of $P||C_{\max}$. Another way to obtain such an approximation is by means of a ρ-*relaxed decision procedure*. This is an algorithm with the following properties: for each instance and any deadline d, (i) it either outputs 'no' or produces a schedule with $C_{\max} \leq \rho d$, and (ii) if the output is 'no', then there is no schedule with $C_{\max} \leq d$. Let B_ρ denote the bisection search procedure in which statements (*) and (**) are replaced by:

```
run a ρ-relaxed decision procedure to schedule the jobs within the deadline $d$; (†)
**if** the output is 'no' (††)
```

We have now obtained a general framework for constructing approximation algorithms for $P||C_{\max}$ and, as we shall see, for other problems as well. The following lemma gives the main reason for its importance.

Lemma 9.5. *For any instance of* $P||C_{\max}$, $C_{\max}(B_\rho)/C^*_{\max} \leq \rho$*; furthermore, if the* ρ*-relaxed decision procedure runs in polynomial time, then* B_ρ *runs in polynomial time.*

Proof. To prove the performance bound, we show that the algorithm maintains two invariants: at each iteration, (i) the schedule S has length $C_{\max} \leq \rho UB$, and (ii) $C^*_{\max} \geq LB$. This suffices, since $UB = LB$ at termination, so that the final schedule has length $C_{\max} \leq \rho UB = \rho LB \leq \rho C^*_{\max}$.

Initially, both (i) and (ii) hold. Suppose that in a particular iteration UB is updated to d. In this case, the decision procedure has produced a schedule with $C_{\max} \leq \rho d$, so that (i) still holds. Now suppose that LB is updated to $d+1$. In this case, the decision procedure has output 'no' and so no feasible schedule completes by d; hence, $C^*_{\max} \geq d+1$ and (ii) still holds.

Furthermore, since the difference $UB - LB$ after ℓ iterations is an integer, bounded from above by $2^{-\ell}C^*_{\max}$, the algorithm must terminate after a polynomial number of iterations. Therefore, if the ρ-relaxed decision procedure runs in polynomial time, the entire bisection search is completed in polynomial time. □

As a consequence of this lemma, we need only design polynomial-time ρ-relaxed decision procedures in order to obtain efficient approximation algorithms. We will use this idea to obtain a polynomial approximation scheme for $P||C_{\max}$ (where m is an input). By Theorem 2.26, this is the best we can hope for, since no fully polynomial approximation scheme exists unless $P = NP$.

For any fixed positive integer k, we will show how to construct a $(1+1/k)$-relaxed decision procedure. Consider an instance of $P||C_{\max}$, along with a deadline $d \geq \max_j p_j$. Once again, we partition the job set into *short* jobs and *long* jobs: in this case, J_s is called short if $p_s \leq d/k$.

For the time being, suppose that we are fortunate and that there are no short jobs.

In the polynomial approximation scheme for a fixed number of machines, we solved such an instance to optimality by complete enumeration. In order to reduce the running time to be polynomial in m, we will produce only a near-optimal schedule for the long jobs. To this end, we will use a dynamic programming algorithm to solve a special case of the bin-packing problem. We assert that, when the bin-packing problem is restricted to the class of instances where there are at most c_1 distinct item sizes and at most c_2 items may be packed in one bin, then it can be solved in $O((c_2 n)^{c_1})$ time (see Exercise 9.4). If c_1 is a constant, this is a polynomial bound.

The decision procedure works as follows. Round down each processing time to the nearest multiple of d/k^2. Use the claimed algorithm for the special case of bin-packing to find the minimum number of machines that suffices to complete all of the rounded jobs by time d, assigning at most $k-1$ jobs to each machine. If the optimal packing uses more than m machines, then output 'no'; otherwise, consider the packing as a schedule for the original instance.

We want to show that this procedure satisfies the two properties of a $(1+1/k)$-relaxed decision procedure. Suppose that it produces a schedule. For any job, the difference between the rounded and the original processing time is less than d/k^2. Since each machine processes fewer than k jobs, the cumulative effect of taking the original processing times is less than $k(d/k^2) = d/k$. Hence, the schedule completes by time $(1+1/k)d$. On the other hand, suppose that the original instance has a schedule of length at most d. Clearly, fewer than k jobs are assigned to each machine. Hence, there must be a bin-packing for the rounded instance that uses at most m machines, each processing at most $k-1$ jobs. Consequently, if the procedure outputs 'no', then there is no schedule for the original instance that completes by time d.

The rounding ensures that there are at most k^2+1 distinct processing times. Since k is fixed, the bin-packing algorithm runs in polynomial time, and it follows that our procedure is a polynomial-time $(1+1/k)$-relaxed decision procedure.

Unfortunately, our discussion assumed that there were no short jobs. If there are short jobs, then first check if $\sum_j p_j > md$. If so, output 'no'. Otherwise, temporarily delete the short jobs and try to find a schedule for the long jobs with the procedure given above. Note that, if the original instance has a schedule that completes by d, then we will obtain a schedule for this subset of jobs that completes by $(1+1/k)d$. Once again, extend this schedule using list scheduling.

If this procedure outputs 'no', then clearly no schedule of length at most d exists. Now we only need to show that, if a schedule is produced, then all short jobs are completed by time $(1+1/k)d$. Suppose that a short job J_s finishes after $(1+1/k)d$. Since $p_s \leq d/k$, it must have started after d, and thus all machines are processing jobs until after time d. But this contradicts the fact that $\sum_j p_j \leq md$.

Summarizing, we have constructed the following $(1+1/k)$-relaxed decision procedure D_k:

We have already seen that, for any fixed value of k, the procedure can be implemented to run in polynomial time. We have proved the following result.

Theorem 9.6. *For any fixed integer $k > 0$, the algorithm D_k is a polynomial-time*

```
if $\sum_j p_j > md$ then
    output ‘no’;
else
    temporarily focus on the subset of jobs $J_L := \{J_j | p_j > d/k\}$ ;
    round down the processing time of each $J_j \in J_L$ to the nearest multiple of
      $d/k^2$ ;
    using the rounded data, find the optimal packing of $J_L$ into bins of capacity
      $d$ where each bin is assigned fewer than $k$ jobs;
    if the number of bins used is more than $m$ then
        output ‘no’;
    else
        interpret the packing as a schedule for the original data;
        extend the schedule to include each $J_j \notin J_L$ by using list scheduling.
    end
end
```

$(1+1/k)$*-relaxed decision procedure for* $P||C_{\max}$.

Corollary 9.7. *If the algorithm* B_ρ *uses the* ρ*-relaxed decision procedure* $D_{\lceil 1/(\rho-1)\rceil}$, *then the family of algorithms* $\{B_\rho\}$ *forms a polynomial approximation scheme for* $P||C_{\max}$.

Exercises

9.4. Give an $O((c_2 n)^{c_1})$ algorithm to solve the bin-packing problem in the case that there are at most at c_1 distinct item sizes and at most c_2 items may be packed in one bin.

9.5. Prove that the following algorithm is a 5/4-relaxed decision procedure for $P||C_{\max}$. The algorithm consists of six stages. In stage 1, check if $\sum_j p_j > md$; if so, output ‘no’ and halt. For stages 2 through 5, all J_j with $p_j < d/4$ are temporarily set aside and the instance consisting of the remaining jobs is considered. In stage 2, while there exists a job J_j with $p_j \geq d/2$, find the longest job J_k such that $p_j + p_k \leq d$ and schedule J_j and J_k by themselves on one machine. Stages 3, 4 and 5 are executed m times; if none of these attempts succeeds in scheduling all J_j with $p_j \geq d/4$ on m machines, then output ‘no’ and halt. For $k = 1,...,m$, the k th attempt is as follows: in stage 3, find the $2k$ unscheduled jobs with the longest processing times and schedule them two per machine in an arbitrary way; in stage 4, while there exists a job J_j with $p_j > 5d/12$, schedule it on a machine with the two longest unscheduled jobs with processing time at most $3d/8$; in stage 5, schedule the remaining jobs three per machine in an arbitrary way. If a schedule has been constructed for all J_j with $p_j \geq d/4$ using no more than m machines, then stage 6 extends it for the remaining jobs by list scheduling.

9.6. Give a 6/5-relaxed decision procedure for $P||C_{\max}$ that runs in $O(mn)$ time. Can you improve this to $O(n \log n)$?

9.7. (a) Consider the following variant of $P||C_{\max}$, which we denote by $P|mem|C_{\max}$. Each job J_j has both a processing requirement p_j and a memory requirement ρ_j. Machine M_i has a memory of size σ_i and can only process job J_j if $\sigma_i \geq \rho_j$. The aim is to find a schedule that minimizes the maximum completion time subject to the memory constraints. Consider the *largest memory first* (LMF) rule: the jobs are listed in order of nondecreasing memory requirement, and when a machine becomes idle, it chooses the next job from the list for which its memory is large enough. Prove that, for any instance of $P|mem|C_{\max}$, $C_{\max}(L\text{MF})/C^*_{\max} \leq 2 - 1/m$.
(b) Suppose that there is a total of $\bar{\sigma}$ units of memory. The memory can be partitioned among the m machines in any way, but it must be done prior to scheduling the jobs. The aim is to find the optimal partition of the memory so that the schedule length for the resulting $P|mem|C_{\max}$ instance is minimized. Show how to compute a partition of the memory for which $C_{\max}(L\text{MF})/C^*_{\max} \leq 2 - 1/m$, where $C^*_{\max}$ is the optimal schedule length with respect to the optimal memory partition. (*Hint*: Suppose that the machines and jobs are indexed in order of nondecreasing memory size and requirement, respectively, and let LB be a lower bound on $C^*_{\max}$. For each $i = 1, \ldots, m$, let $k(i)$ be the smallest integer k such that $\sum_{j=1}^{k} p_j > (i-1)LB$. Then we need $\sigma_i \geq \rho_{k(i)}$ in order to achieve the lower bound. If there is sufficient memory to assign $\rho_{k(i)}$ units to each M_i, then the algorithm terminates; if there is not, this fact can be used to increase LB and a new iteration is begun.)
9.8. It is not hard to extend the performance guarantees for variants of list scheduling to $P|r_j|L_{\max}$.
(a) Prove that, if list scheduling is used with the jobs given in *EDD* order, then, for any instance of $P|r_j|L_{\max}$, $L_{\max}(EDD) - L^*_{\max} \leq (2 - 1/m)\max_j p_j$.
(b) Prove that, in the delivery time model (as in Chapter 3), for any instance of $P|r_j|L_{\max}$, $L_{\max}(\text{LS})/L^*_{\max} < 2$.

9.3. Uniform machines: performance guarantees

Unfortunately, machines are not created equal, and may work at different speeds. Order the machines so that the speeds are nonincreasing; that is, $s_1 \geq s_2 \geq \ldots \geq s_m$. If p_j denotes the processing requirement of job J_j, then processing J_j on M_i takes p_j/s_i time units.

In this more general model, the simplest algorithms do not work quite as well. For example, consider the list scheduling rule. We can analyze this procedure using the same techniques as in the case of identical machines. Let J_ℓ be the job that completes last in some list schedule, and let t denote the time at which J_ℓ begins processing; no machine is idle prior to time t. Since M_i has the capacity to process ts_i units by time t, it follows that

$$\sum_{j \neq l} p_j \geq t \sum_i s_i.$$

Since there must be sufficient capacity to process all of the jobs by $C^*_{\max}$,

$$C^*_{\max} \sum_i s_i \geq \sum_j p_j.$$

The processing of J_ℓ takes at least p_ℓ/s_1 time units, and so $C^*_{\max} \geq p_\ell/s_1$. On the other hand, J_ℓ is certainly processed in the list schedule within p_ℓ/s_m time units. Combining these pieces, we see that

$$\begin{aligned} C_{\max}(\mathrm{LS}) \leq t + p_\ell/s_m \leq \frac{\sum_{j\neq l} p_j}{\sum_i s_i} + \frac{p_\ell}{s_m} &= \frac{\sum_j p_j}{\sum_i s_i} + p_\ell\left(\frac{1}{s_m} - \frac{1}{\sum_i s_i}\right) \\ &\leq C^*_{\max} + C^*_{\max} s_1\left(\frac{1}{s_m} - \frac{1}{\sum_i s_i}\right). \end{aligned}$$

Thus, we have shown the following theorem.

Theorem 9.8. *For any instance of $Q||C_{\max}$, $C_{\max}(LS)/C^*_{\max} \leq 1 + s_1/s_m - s_1/\sum_i s_i$.*

In fact, this bound is tight, and by an appropriate choice of machine speeds may exceed any constant (see Exercise 9.9).

If we were to follow the approach adopted for identical machines, we would next focus on ways to construct a special list for which the list scheduling rule can then be run. However, unlike $P||C_{\max}$, there need not be a list for which the list scheduling rule gives the optimal schedule for $Q||C_{\max}$. The reason for this is that list scheduling always delivers a schedule in which no machine is idle while there is still an unscheduled job. It is easy to construct instances for which every optimal schedule contains such *unforced idle time* (see Exercise 9.10).

The following simple variant of list scheduling may create unforced idle time: schedule the jobs in the order of the list, always assigning the next job to the machine on which it would complete earliest. While this strategy has a better performance guarantee, it still does not deliver solutions for $Q||C_{\max}$ that are within a constant factor of the optimum. However, the LPT variant of this rule, which we shall call LPT$'$, works quite well.

Theorem 9.9. *For any instance of $Q||C_{\max}$, $C_{\max}(LPT')/C^*_{\max} \leq 2 - 1/m$.*

Proof. Suppose that the theorem is false, and consider a counterexample in which the sum of the number of jobs and the number of machines is minimum. Let m and n, respectively, denote the number of machines and the number of jobs in this counterexample. We shall assume that the jobs are indexed so that $p_1 \geq p_2 \geq \ldots \geq p_n$.

Consider the schedule generated by LPT$'$. Suppose that no job is scheduled on machine M_i. Since J_n is not scheduled on M_i, $p_n/s_i \geq C_{\max}(\mathrm{LPT}') > C^*_{\max}$. Therefore, M_i must be completely idle in any optimal schedule as well. But then, if we delete M_i, we have not changed either the LPT$'$ schedule or the optimal schedule, and we obtain a smaller counterexample, which is a contradiction. Similarly, suppose that J_ℓ is the last job to finish, where $\ell < n$. In that case, we can obtain a smaller counterexample by deleting all jobs J_j, $j > \ell$, since that does not change the length of the LPT$'$ schedule and does not increase $C^*_{\max}$. Thus, we may assume that J_n is the last job to finish.

By the way in which J_n is assigned to a machine, we see that for each M_i,

$$C_{\max}(\text{LPT}')s_i \leq p_n + \sum_{J_j \text{ on } M_i, j \neq n} p_j.$$

Summing this inequality for all $i = 1,...,m$, we get

$$C_{\max}(\text{LPT}')\sum_i s_i \leq mp_n + \sum_{j\neq n} p_j = (m-1)p_n + \sum_j p_j.$$

Since no machine is idle, $m \leq n$, and thus $mp_n \leq \sum_j p_j$. From these inequalities, we see that

$$C_{\max}(\text{LP}T') \leq (2-\frac{1}{m})\frac{\sum_j p_j}{\sum_i s_i} \leq (2-\frac{1}{m})C^*_{\max},$$

which shows that our instance is not a counterexample. □

Unlike all of the analyses that we have seen thus far, the bound given in Theorem 9.9 is not tight. In fact, by using additional structural information, one can show that the performance ratio is no more than 19/12. This is also not known to be tight; for the worst example known, $C_{\max}(\text{LPT}')/C^*_{\max} = 1.52$.

From the previous section, recall the notion of a ρ-relaxed decision procedure and the bisection search procedure B_ρ in which it was used. The concept of a ρ-relaxed decision procedure can be applied equally well to $Q||C_{\max}$. Furthermore, we can adapt the initial lower and upper bounds to obtain the following analogue of B_ρ.

```
$S :=$ the LPT' schedule;
$UB := C_{\max}(\text{LP}T')$ ;
$LB := \max\{p_1/s_1, (\sum_j p_j)/(\sum_i s_i), C_{\max}(\text{LP}T')/2\}$ ;
while $LB \neq UB$ do
    $d := \lfloor \frac{LB+UB}{2} \rfloor$ ;
    run a ρ-relaxed decision procedure to schedule the jobs within the deadline
      $d$ ;
    if the output is 'no' then
        $LB := d+1$;
    else
        $UB := d$; $S :=$ new schedule;
    end
end
```

It is easy to see that the analogue of Lemma 9.5 is also valid for $Q||C_{\max}$. We will give a simple $3/2$-relaxed decision procedure for $Q||C_{\max}$, which therefore yields a 3/2-approximation algorithm for $Q||C_{\max}$.

Our 3/2-decision procedure is a recursive algorithm; that is, it calls itself as a subroutine, but for a smaller instance. Let the procedure be called *Recurse*(J, m, d) where J denotes the set of jobs, m indicates that there are machines $M_1, M_2, ..., M_m$ (where $s_1 \geq s_2 \geq ... \geq s_m$) and d is the deadline being considered. Recall that the output must either be a schedule that completes by $(3/2)d$ or 'no', where 'no' is output only if there is no schedule that completes within the deadline d.

The procedure first checks if the capacity of the machines is sufficient for the given jobs: if $\sum_{J_j \in J} p_j > d \sum_{i=1}^{m} s_i$, then output 'no' and halt. Otherwise, if $m = 1$, all jobs in J are assigned to machine M_1, and the procedure ends. If $m > 1$, let S be the set of all jobs $J_j \in J$ such that $p_j \leq ds_m/2$. If there are no jobs J_ℓ in $J - S$ with $p_\ell \leq ds_m$, call *Recurse*$(J - S, m - 1, d)$. Otherwise, among all such jobs in $J - S$, let J_r denote one of these with maximum processing requirement. Assign J_r to be scheduled on M_m and call *Recurse*$(J - S - \{J_r\}, m - 1, d)$. If the procedure has not output 'no' and halted, extend the schedule for $J - S$ on $M_1, ..., M_m$ to include S by using list scheduling; that is, order the jobs of S arbitrarily, and assign the next job to be scheduled on the machine that is currently finishing earliest.

Theorem 9.10. *Recurse*$(\{J_1, ..., J_n\}, m, d)$ *is a* $3/2$*-relaxed decision procedure for* $Q||C_{\max}$.

Proof. We first show that if there is a schedule that completes within time d, then the procedure outputs a schedule. To prove this, we will prove the following important claim: if the original instance has a schedule that completes by time d, then so must the smaller instance for which the recursive call is made. Given this, it is clear that no recursive call will ever output 'no' for an instance with a feasible deadline, and so a schedule will be output. If d is a feasible deadline to schedule J on the fastest m machines, then $J - S$ can also be scheduled on these m machines by time d. But for this subinstance, each job J_j has $p_j > ds_m/2$, and so in any schedule that completes by time d, at most one of these jobs is scheduled on M_m. If all jobs have processing requirement more than ds_m, then clearly, M_m must be completely idle. Otherwise, an interchange argument shows that there always exists a feasible schedule in which J_r is scheduled on M_m. In either case, we have shown that the subinstance for which the recursive call is made has a schedule that completes within time d.

For the second half of the proof, we will show by induction on m that if a schedule is produced, then it completes all jobs before $(3/2)d$. It is a simple exercise to verify the claim for $m = 1$. Suppose that the algorithm outputs a schedule. Then the recursive call must also output a schedule, and by the inductive hypothesis, all jobs in $J - S$ are completed before $(3/2)d$. The job J_r, if it exists, clearly completes by time d. Since the machines have sufficient capacity to schedule all jobs within d time units, in any partial schedule, the machine that is currently finishing earliest must complete its jobs before time d. Each job $J_j \in S$ has $p_j \leq ds_m/2$ and therefore takes at most $d/2$ time units on *any* machine. But then each job in the list is assigned to a machine on which it completes before $(3/2)d$. $\square$

There is an analogue of the multifit (MF) procedure for $Q||C_{\max}$ as well. This algorithm does still better.

Theorem 9.11. *When multifit is run for only ℓ iterations of bisection search, then for any instance of $Q||C_{\max}$, $C_{\max}(MF)/C^*_{\max} \leq 1.382 + 2^{-\ell}$.*

Finally, the approach that employs a $(1+1/k)$-relaxed decision procedure to construct a polynomial approximation scheme for $P||C_{\max}$ can also be extended to the case of $Q||C_{\max}$. The polynomial approximation scheme for $Q||C_{\max}$ is substantially more complicated, and beyond the scope of this book.

Exercises

9.9. Give a family of instances of $Q||C_{\max}$ that shows there does not exist a constant c such that $C_{\max}(\mathrm{LS})/C^*_{\max} \leq c$.

9.10. Give an instance of $Q||C_{\max}$ for which all optimal solutions have unforced idle time.

9.11. Show that the following algorithm is a 2-relaxed decision procedure for $Q||C_{\max}$. For each machine, construct a list of jobs so that these lists form a partition of the set of jobs. Assign each job J_j to the list of the slowest machine M_i such that $p_j \leq s_i d$. When a machine M_i becomes idle, schedule the next job in its list on M_i; if its list is empty, find the the next slower machine with a non-empty list, and schedule the next job from *that* list on M_i. (When a job is scheduled, delete it from its list.) If the schedule constructed has $C_{\max} > 2d$, instead output 'no'.

9.4. Unrelated machines: performance guarantees

When the machines are unrelated, it is possible that J_1 takes much longer on M_2 than on M_1, whereas J_2 takes much longer on M_1 than on M_2. This generalization makes it substantially more difficult to obtain good approximate solutions. We shall see that this statement can be made more precise. Throughout the next two sections, the time that it takes to process J_j on M_i will be denoted by $p_{ij} = p_j/s_{ij}$, which we assume to be integral.

The worst-case performance of list scheduling or any other known simple scheduling rule is pretty dismal. For example, the *greedy* (G) algorithm naively assigns each job to the machine on which it takes the least time. Clearly, this schedule completes within $T = \sum_j \min_i p_{ij}$. In addition, one can view T as the minimum total requirement of the jobs, and since the best one could hope for is to balance this load evenly over all machines, $C^*_{\max} \geq T/m$.

Theorem 9.12. *For any instance of $R||C_{\max}$, $C_{\max}(G)/C^*_{\max} \leq m$.*

This bound is tight (see Exercise 9.12).

A clever variant of list scheduling is significantly better. Maintain a separate list for each machine, and sort all n jobs for M_i 's list in order of nondecreasing *relative speed*, $p_{ij}/\min_h p_{hj}$, $j = 1, ..., n$. Whenever a machine is idle, assign the next unscheduled job in its list, unless the smallest ratio of an unscheduled job is more than *sqrtm*; in this case, no further jobs are scheduled on that machine. This *relative speed* (RS) rule can be shown to guarantee the following performance.

Theorem 9.13. *For any instance of $R||C_{\max}$, $C_{\max}(RS)/C^*_{\max} \leq 2.5\sqrt{m}+1+1/(2\sqrt{m})$.*

This bound is tight up to a constant factor.

Linear programming can be used to construct a much more effective procedure. One natural way to formulate $R||C_{\max}$ as an integer linear programming problem is as follows:

$$\min C_{\max}$$

subject to

$$\sum_j p_{ij}x_{ij} \leq C_{\max}, \qquad \text{for } i = 1,...,m, \tag{9.2}$$

$$\sum_i x_{ij} = 1, \qquad \text{for } j = 1,...,n, \tag{9.3}$$

$$x_{ij} \in \{0,1\}, \qquad \text{for } i = 1,...,m, j = 1,...,n, \tag{9.4}$$

where x_{ij} indicates whether job J_j is assigned to M_i. If each integer constraint (9.4) is relaxed to the linear constraint $x_{ij} \geq 0$, then we can solve the resulting linear program LP_1 to obtain a lower bound on $C^*_{\max}$.

In fact, LP_1 can also be used to obtain good integer solutions as well. The main idea is to obtain an optimal solution x^* to LP_1; then, if $x^*_{ij} = 1$, assign J_j to machine M_i, and deal with the unassigned jobs in some other manner. We may assume that x^* is an extreme point of LP_1, and we will use this in a critical way.

Consider any extreme point $\hat{x}$ of the feasible region of the linear program LP_1. There are $mn+1$ variables in LP_1, and so there must be $mn+1$ linearly independent constraints of LP_1 for which $\hat{x}$ is the unique feasible solution that satisfies these constraints with equality. However, other than those of the form $x_{ij} \geq 0$, there are only $m+n$ constraints. Therefore, all but $m+n-1$ of the components of $\hat{x}$ must equal 0. In order to satisfy the constraints (9.3), at least one component $\hat{x}_{ij}$ must be positive for each $j = 1,...,n$. There are at most $m-1$ other positive components, and so for all but at most $m-1$ of the constraints (9.3), exactly one variable is positive, and hence equal to 1. Therefore, the approach suggested above will be able to immediately assign all but at most $m-1$ jobs.

If the number of machines is small, then the schedule can be completed by enumerating all of the possible extensions and choosing the best one. This algorithm based on linear programming (LP) can be analyzed in the following way. Consider the schedule that is based in part on the integer assignments indicated by the optimal solution to LP_1, but is completed by assigning the remaining jobs to the machine on which they run in a particular optimal schedule. Of course, the schedule found by heuristic is at least as good as this new one. The new schedule can be split into two partial schedules in the obvious way. The partial schedule given by the linear programming solution has $C_{\max}$ no more than $C^*_{\max}$, as does the partial schedule for the remaining jobs. Since $C_{\max}(\mathrm{LP})$ is no more than the sum of these two parts, $C_{\max}(\mathrm{LP}) \leq 2C^*_{\max}$.

Unfortunately, there can be m^{m-1} ways to assign each of the $m-1$ jobs to one of m machines, and so there is no apparent way to find the optimal extension in polynomial time. However, this takes only constant time if m is fixed, and so we get the following result.

Theorem 9.14. *The algorithm LP is a polynomial-time algorithm for $Rm||C_{\max}$, and for any instance, $C_{\max}(LP)/C^*_{\max} \leq 2$.*

Once again, there exist examples that prove that this analysis is tight. Later in this section, we will derive a fully polynomial approximation scheme for $Rm||C_{\max}$.

This linear programming approach can be extended to yield a polynomial-time algorithm when m is an input. Once again, we will switch to the perspective of finding a ρ-relaxed decision procedure. In this case, the bisection search procedure must be modified to initialize S, UB and LB as follows:

$S :=$ the greedy schedule;
$UB := C_{\max}(G)$;
$LB := \sum_j \min_i p_{ij}$;

Since UB and LB can be within a factor of m initially, we might need to perform an additional $\log_2 m$ iterations until the bisection search terminates, but an analogue of Lemma 9.5 remains true.

We will show that the linear programming approach leads to a polynomial-time 2-relaxed decision procedure, which then yields a polynomial-time 2-approximation algorithm. Suppose that we wish to test if $C^*_{\max} \leq d$. We can view this as testing the feasibility of the following system of constraints:

$$\sum_j p_{ij}x_{ij} \leq d, \text{ for } i = 1,...,m, \tag{9.5}$$

$$\sum_i x_{ij} = 1, \text{ for } j = 1,...,n, \tag{9.6}$$

$$x_{ij} \in \{0,1\}, \text{ for } i = 1,...,m, j = 1,...,n. \tag{9.7}$$

We will, as before, consider the linear relaxation of this, but in order to obtain a tighter relaxation, we add the constraints

$$x_{ij} = 0, \text{ if } p_{ij} > d. \tag{9.8}$$

Let LP$_2$ denote the linear relaxation of the system of constraints (9.5)–(9.8). We will show how to round any extreme point of LP$_2$ into an *integer* solution that corresponds to a schedule with $C_{\max} < 2d$. This rounding procedure can be done in polynomial time, and so we get the following polynomial-time 2-relaxed decision procedure: test if LP$_2$ is feasible; if not, output 'no'; otherwise, find an extreme point of LP$_2$ and round it to obtain the desired integer solution.

Suppose that LP$_2$ is feasible and we want to round the extreme point $\hat{x}$. One way to model the structure of this solution is to form a bipartite graph: $G(\hat{x}) = (M,J,E)$, where $M = \{M_1,...,M_m\}$ and $J = \{J_1,...,J_n\}$ are the sets of machines and jobs, respectively, and $E = \{(M_i,J_j)|\hat{x}_{ij} > 0\}$. As in the proof of Theorem 9.14, by observing that LP$_2$ has mn variables and only $m+n$ constraints not of the form $x_{ij} \geq 0$, we see that $G(\hat{x})$ has no more edges than nodes. We now show that each connected component of $G(\hat{x})$ has this property.

Let C be a connected component of $G(\hat{x})$. If M_C and J_C are the sets of machine and job nodes contained in C, let $\hat{x}_C$ denote the restriction of $\hat{x}$ to those $\hat{x}_{ij}$ for which $i \in M_C$ and $j \in J_C$, and let $\hat{x}_{\bar{C}}$ denote the remaining components of $\hat{x}$. For simplicity of notation, reorder the components so that $\hat{x} = (\hat{x}_C, \hat{x}_{\bar{C}})$. The connected component C induces a smaller scheduling problem which restricts attention to the subset of machines M_C and the subset of jobs J_C. We can formulate an analogous linear program to LP$_2$ for this subproblem, and denote it by LP$_C$.

We first prove that $\hat{x}_C$ is an extreme point of LP$_C$. Suppose not; then there exist distinct y_1 and y_2 such that $\hat{x}_C = (y_1 + y_2)/2$, where each y_i is a feasible solution of LP$_C$. But now, $\hat{x} = ((y_1, \hat{x}_{\bar{C}}) + (y_2, \hat{x}_{\bar{C}}))/2$ where each $(y_i, \hat{x}_{\bar{C}})$ is a feasible solution of LP$_2$, which contradicts the fact that $\hat{x}$ was chosen to be an extreme point of LP$_2$. Hence, $\hat{x}_C$ is an extreme point of LP$_C$, and it follows that $G(\hat{x}_C) = C$ has no more edges than nodes.

Since each component of $G(\hat{x}_C)$ has no more edges than nodes, and is, by definition, connected, each component must be either a tree or a tree plus one additional edge. We now use this fact to round the corresponding extreme point $\hat{x}$. As before, for each edge (M_i,J_j) with $\hat{x}_{ij} = 1$, we assign job J_j to machine M_i. These jobs correspond to job nodes of degree 1, so that by deleting all of these nodes we get a graph of the same type, $G'(\hat{x})$, with the additional property that each job node has degree at least 2.

We show that $G'(\hat{x})$ has a matching that covers all of the job nodes. For each component that is a tree, root the tree at any node, and match each job node with any one of its children. (Note that each job node must have at least one child and that, since each node has at most one parent, no machine is matched with more than one job.) For each component that contains a cycle, take alternate edges of the cycle in the matching. (Note that the cycle must be of even length.) If the edges of the cycle are deleted, we get a collection of trees which we think of as rooted at the node that had been contained in the cycle. For each job node that is not already matched, pair it with one of its children. This gives us the desired matching. If (M_i,J_j) is in the matching, assign job J_j to be processed on machine M_i.

It is straightforward to verify that the resulting schedule has $C_{\max} \leq 2d$. For each machine M_i, at most one job J_j is assigned to it based on the matching in $G'(\hat{x})$. Since the corresponding $\hat{x}_{ij}$ must be greater than 0, $p_{ij} \leq d$. For all of the remaining jobs assigned to M_i, the corresponding component of $\hat{x}$ is one, and so by (9.5), the total processing time of these jobs is at most d. Therefore, each machine is assigned jobs with total processing time at most $2d$. On the other hand, any schedule with $C_{\max} \leq d$ corresponds to an (integer) feasible solution to LP$_2$, and so if LP$_2$ is infeasible, we are justified in answering ‘no’.

By using this relaxed decision procedure within the usual framework, we have obtained the following result for this approximation algorithm based on linear programming (LP$'$).

Theorem 9.15. *The algorithm LP' is a polynomial-time algorithm for $R||C_{\max}$ and for any instance, $C_{\max}(LP')/C^*_{\max} \le 2$.*

It is not hard to construct a family of instances that show that this analysis cannot be improved to yield a better worst-case bound (see Exercise 9.13).

It is significant to note that the structure of $G(\hat{x})$ that was used in rounding $\hat{x}$ can also be used in finding an extreme point of LP$_2$. From a practical point of view, this characterization leads to a particularly efficient implementation of the simplex method. Alternatively, the structure of this linear program can be used to derive a special-purpose combinatorial algorithm that runs in polynomial time.

Consider again the special case when the number of machines is a fixed integer m. We will show that in this case, there is a fully polynomial approximation scheme. Clearly, this implies that such a scheme exists for the cases of a fixed number of identical machines or uniform machines. The heart of the scheme is a pseudopolynomial algorithm to find an optimal solution to $Rm||C_{\max}$. It is not hard to see that dynamic programming can be used to derive an algorithm that runs in $O(nmT^{m-1})$ time, where $T = \sum_j \min_i p_{ij}$ (see Exercise 9.18).

Given this pseudopolynomial-time algorithm, it is rather straightforward to complete the fully polynomial approximation scheme. For each positive integer k, we will construct a $(1+1/k)$-approximation algorithm. Approximate each processing time p_{ij} by $\lfloor p_{ij}/\delta \rfloor$, where $\delta = T/(kmn)$. Note that this rescales T to at most kmn. Find the optimal solution to this rescaled and rounded problem in $O(nm(kmn)^{m-1})$ time, and consider this schedule for the original processing times. The schedule obtained is also optimal for the instance with processing times given by $\delta\lfloor p_{ij}/\delta \rfloor$, $i = 1,\dots,m, j = 1,\dots,n$, and its value of $C_{\max}$ with these processing times is clearly a lower bound on the desired optimum. Since each C_j is the sum of at most n individual processing times, the difference between the value of $C_{\max}$ for this schedule with the original processing times and the value of $C_{\max}$ for this schedule with the rounded processing times is at most

$$n\delta = nT/kmn = \frac{1}{k}\frac{T}{m} \le \frac{1}{k}C^*_{\max}.$$

Therefore, this algorithm, which we shall call H_k, has the worst-case guarantee $C_{\max}(H_k)/C^*_{\max} \le 1+1/k$, and we have obtained the following result.

Theorem 9.16. *For any fixed m, the family of algorithms $\{H_k\}$ forms a fully polynomial approximation scheme for $Rm||C_{\max}$.*

Exercises

9.12. Give a family of examples that shows that for any number of machines, the analysis given for the greedy algorithm is tight.

9.13. Give a family of examples that shows that the analysis of LP$'$ given in Theorem 9.15 is tight. In addition, show that this bound is tight, even if $m = 2$.
9.14. Suppose that when job J_j is assigned to machine M_i a cost of c_{ij} is incurred. Let c^* denote the minimum total cost of any schedule with maximum completion time $C^*_{\max}$. Use the linear programming approach to give an algorithm that delivers a schedule with $C_{\max} \leq 2C^*_{\max}$ with total cost no more than c^*.
9.15. Suppose that when job J_j is scheduled on machine M_i, it may take anywhere between l_{ij} and u_{ij} time units. If job J_j is scheduled on machine M_i to be processed in u_{ij} time units, a cost of c_{ij} is incurred; job J_j may be processed faster on M_i by incurring an additional cost of s_{ij} per unit decrease. For example, the cost of scheduling J_j on M_i to take l_{ij} units is $c_{ij} + s_{ij}(u_{ij} - l_{ij})$. The objective is to schedule all jobs with minimum total cost, subject to the constraint that all jobs are completed within a deadline d. Use the linear programming approach to devise an algorithm that delivers a schedule that costs no more than this optimum, and completes all jobs within $2d$. (*Hint*: When job J_j is processed for any time in the range between l_{ij} and u_{ij}, this can be viewed as processing some fraction of the job on machine M_i at the speed needed to process the entire job in l_{ij} time units, and the complementary fraction is processed at the speed at which it would be processed in u_{ij} time units.)
9.16. Consider the variant of the decision version of $R||C_{\max}$ where each machine M_i has its own deadline d_i. Modify the approach used in algorithm LP$'$ to give an algorithm that either outputs 'no', or else outputs a schedule in which each machine M_i completes its assigned jobs by $d_i + \max_j p_{ij}$, where 'no' is output only when no schedule completes all jobs within the given deadlines.
9.17. The linear programming approach can also be used to design a polynomial approximation scheme for $Rm||C_{\max}$, where the space required does not depend exponentially on m. Recall that in designing a $(1+1/k)$-approximation algorithm it is sufficient to find a $(1+1/k)$-relaxed decision procedure. To use this approach, one can define a suitable notion of a *bad* assignment, where the job takes a constant fraction of the time prior to the deadline if assigned to that machine. As a result, there are only a constant number of ways to make bad assignments of the jobs. Combine complete enumeration of these possibilities with the algorithm of Exercise 9.16, to design a $(1+1/k)$-relaxed decision procedure.
9.18. Construct a dynamic programming algorithm to solve $Rm||C_{\max}$ in $O(nmT^{m-1})$ time, where $T = \sum_j \min_i p_{ij}$. (*Hint*: Recall Section 8.3.)

9.5. Unrelated machines: impossibilities

Unlike $P||C_{\max}$ and $Q||C_{\max}$, no polynomial approximation scheme is known for $R||C_{\max}$. It is highly unlikely that such a scheme exists, since this would imply that $P = NP$. In order to prove this assertion, we investigate the computational complexity of the decision version of $R||C_{\max}$ with small integral deadlines.

Theorem 9.17. *For $R||C_{\max}$, the question of deciding if $C^*_{\max} \leq 3$ is NP-complete.*

Proof. We prove this result by a reduction from the 3-dimensional matching problem. We are given an instance of this problem, consisting of a family of triples $\{T_1, T_2, ..., T_m\}$ over the ground set $A \cup B \cup C$, where A, B and C are disjoint sets such that $|A| = |B| = |C| = n$; each T_i satisfies $|T_i \cap A| = |T_i \cap B| = |T_i \cap C| = 1$. We construct an instance of the scheduling problem with m machines and $2n + m$ jobs. Machine M_i corresponds to the triple T_i, for $i = 1, ..., m$. There are $3n$ 'element jobs' that correspond to the $3n$ elements of $A \cup B \cup C$ in the natural way. In addition, there are $m - n$ 'dummy jobs'. (If $m < n$, we construct some trivial 'no' instance of the scheduling problem.) Machine M_i corresponding to $T_i = (a_j, b_k, c_\ell)$ can process each of the jobs corresponding to a_j, b_k and c_ℓ in one time unit and each other job in three time units. Note that the dummy jobs require three time units on each machine.

It is quite simple to show that $C^*_{\max} \leq 3$ if and only if there is a 3-dimensional matching. Suppose there is a matching. For each $T_i = (a_j, b_k, c_\ell)$ in the matching, schedule the element jobs corresponding to a_j, b_k and c_ℓ on machine M_i. Schedule the dummy jobs on the $m - n$ machines corresponding to the triples that are not in the matching. This gives a schedule with $C_{\max} = 3$. Conversely, suppose that there is such a schedule. Each of the dummy jobs requires three time units on any machine and is thus scheduled by itself on some machine. Consider the set of n machines that are not processing dummy jobs. Since these are processing all of the $3n$ element jobs, each of these jobs is processed in one time unit. Each three jobs that are assigned to one machine must therefore correspond to elements that form the triple corresponding to that machine. Since each element job is scheduled exactly once, the n triples corresponding to the machines that are not processing dummy jobs form a matching. □

As an immediate corollary of this theorem, we get the following result.

Corollary 9.18. *For every $\rho < 4/3$, there does not exist a polynomial-time ρ-approximation algorithm for $R||C_{\max}$, unless $P = NP$.*

Proof. Suppose there were such an algorithm. We will show that it yields a polynomial-time algorithm for the 3-dimensional matching problem. Given an instance I of the 3-dimensional matching problem, map it into an instance of $R||C_{\max}$ using the reduction given above, and then apply the presumed approximation algorithm. We have just seen that I is a 'yes' instance if and only if the instance of $R||C_{\max}$ does have a schedule of length 3. Therefore, when I is a 'yes' instance, the approximation algorithm must output a schedule of length less than $(4/3)C^*_{\max}$. But this length must be an integer, and so it must be at most 3. When I is a 'no' instance, the algorithm produces a schedule of length at least 4. Therefore, the algorithm outputs a schedule of length at most 3 if and only if I is a 'yes' instance. □

The technique employed in Theorem 9.17 can be refined to yield a stronger result.

Theorem 9.19. *For $R||C_{\max}$, the question of deciding if $C^*_{\max} \leq 2$ at most 2 is NP-complete.*

Proof. We again start from the 3-dimensional matching problem. We call the triples that contain a_j *triples of type* j. Let t_j be the number of triples of type j, for $j = 1,\ldots,n$. As before, machine M_i corresponds to the triple T_i, for $i = 1,\ldots,m$. There are now only $2n$ element jobs, corresponding to the $2n$ elements of $B \cup C$. We refine the construction of the dummy jobs: there are $t_j - 1$ dummy jobs of type j, for $j = 1,\ldots,n$. (Note that the total number of dummy jobs is $m-n$, as before.) Machine M_i corresponding to a triple of type j, say, $T_i = (a_j, b_k, c_\ell)$, can process each of the element jobs corresponding to b_k and c_ℓ in one time unit and each of the dummy jobs of type j in two time units; all other jobs require three time units on machine M_i.

Suppose there is a matching. For each $T_i = (a_j, b_k, c_\ell)$ in the matching, schedule the element jobs corresponding to b_k and c_ℓ on machine M_i. For each j, this leaves $t_j - 1$ idle machines corresponding to triples of type j that are not in the matching; schedule the $t_j - 1$ dummy jobs of type j on these machines. This completes a schedule with $C_{\max} = 2$. Conversely, suppose that there is such a schedule. Each dummy job of type j is scheduled on a machine corresponding to a triple of type j. Therefore, there is exactly one machine corresponding to a triple of type j that is not processing dummy jobs, for $j = 1,\ldots,n$. Each such machine is processing two element jobs in one time unit each. If the machine corresponds to a triple of type j and its two unit-time jobs correspond to b_k and c_ℓ, then (a_j, b_k, c_ℓ) must be the triple corresponding to that machine. Since each element job is scheduled exactly once, the n triples corresponding to the machines that are not processing dummy jobs form a matching. □

Corollary 9.20. *For every* $\rho < 3/2$*, there does not exist a polynomial-time* ρ*-approximation algorithm for* $R||C_{\max}$*, unless* $P = NP$.

Exercises

9.19. Prove for any integers $p < q$ such that $2p \neq q$, $R||C_{\max}$ is NP-hard even in the case that all $p_{ij} \in \{p, q\}$.

9.20. In contrast to Exercise 9.19, give a polynomial-time algorithm to solve the special case of $R||C_{\max}$ when each $p_{ij} \in \{1, 2\}$.

9.6. Two identical machines: probabilistic analysis

The results presented in the previous sections provide ample illustration of the power of a worst-case approach to the analysis of heuristics. While necessarily pessimistic, the outcome of a worst-case analysis at least yields an ironclad performance guarantee that will always be valid. This safety belt often comes at the expense of realism, in that computational experiments might indicate that the worst-case behavior is rarely registered in practice. On the contrary, the average performance of an approximation algorithm is usually strikingly better than its worst-case behavior would suggest. In this section, we shall consider a mathematical framework in which these empirical results can be analyzed, and reconsider several of the approximation algorithms already discussed, but from this perspective. As a caveat, however, the reader should note that we might have little understanding of what an average instance really is in practice; the results presented in this section might be no more realistic than our assumptions about the instances considered.

We first must outline the precise mathematical definitions that will capture the notion of the average performance of a heuristic. Probability theory provides an appropriate setting for this approach. We shall assume that the reader is familiar with the basic terminology of this area. In the spirit of empirical computational work, a problem instance will be regarded as being generated by a *random mechanism*. For example, for the scheduling problem $P||C_{\max}$, one would typically assume that the processing times p_j, $j = 1,...,n$, are *random variables* whose joint distribution is given in advance. Given a particular *realization* of these random variables, the heuristic solution is computed; its value is obviously a random variable as well, whose distribution can be analyzed and whose *expected value* informs us about the average behavior of the heuristic in question, especially when compared to the *expected value of the optimal solution*.

The *probabilistic analysis* of algorithms, then, starts from a probability distribution over the class of all problem instances, and focuses on the random variables describing algorithmic behavior on a randomly generated instance. The analysis can be technically demanding; frequently, it is *asymptotic* in nature, in that precise statements are only possible if the problem size is allowed to go to infinity. Hence, it is appropriate at this point to introduce the three modes of *stochastic convergence* that typically arise in such a situation.

If $\mathbf{y}_1, \mathbf{y}_2, ...$ is a sequence of random variables, then *almost sure (a.s.) convergence* of the sequence to a constant c means that

$$\Pr\{\lim_{n\to\infty} \mathbf{y}_n = c\} = 1.$$

Sometimes this is referred to as convergence *with probability 1*. It implies the weaker condition *convergence in probability*, which requires that, for every $\varepsilon > 0$,

$$\lim_{n\to\infty} \Pr\{|\mathbf{y}_n - c| > \varepsilon\} = 0. \tag{9.9}$$

Finally, *convergence in expectation* means that

$$\lim_{n\to\infty} |Ex[\mathbf{y}_n] - c| = 0,$$

and also implies (9.9).

We shall illustrate some of these notions on the problem $P2||C_{\max}$. In this case, a problem instance corresponds to the processing times $p_1, p_2, ..., p_n$; let us assume these to be independent random variables that are *uniformly distributed* over the interval [0,1], which is a model favored by analysts and experimenters alike.

Let us first consider the performance of the list scheduling (LS) algorithm from a probabilistic perspective. From (9.1), we deduce that in the case of $P2||C_{\max}$,

$$C_{\max}(\mathrm{LS}) \le \sum_j p_j/2 + p_{\max}/2.$$

Since the first term on the right-hand side is a lower bound on $C_{\max}*$, we see that

$$\mathbf{C}_{\max}(LS)/\mathbf{C}^*_{\max} \le 1 + \mathbf{p}_{\max}/\sum_j \mathbf{p}_j. \tag{9.10}$$

Obviously, $\mathbf{p}_{\max} \le 1$. In addition, the *strong* law of large numbers says that

$$(\sum_j \mathbf{p}_j)/(n/2) \to 1(\text{a.s.}).$$

Hence, the ratio $\mathbf{C}_{\max}(LS)/\mathbf{C}^*_{\max}$ itself also converges to 1 with probability 1. Put differently, the relative error of this simple heuristic almost surely vanishes for large problem sizes, and its depressing worst-case error of 50 percent occurs rather infrequently.

As we have seen above, *almost sure convergence* is just one of several ways to capture the asymptotic behavior of a sequence of random variables. Thus, there are other ways to capture the asymptotic optimality of these simple heuristics. They do, however, require a different probabilistic starting point.

By way of example, consider the inequality

$$\Pr\{\frac{\sum_j \mathbf{p}_j}{n} - \frac{1}{2} \ge t\} \le e^{-2nt^2},$$

which is a special case of Hoeffding's inequality. From (9.10), with $t = 1/4$, one can easily deduce that

$$\Pr\left\{\frac{\mathbf{C}_{\max}(LS) - \mathbf{C}^*_{\max}}{\mathbf{C}^*_{\max}} \le \frac{4(m-1)}{3n}\right\} \ge 1 - e^{-n/8},$$

which, of course, confirms error convergence to 0 in probability, but which also provides precise information on the rate at which the relative error of list scheduling converges to 0. We shall later return to the third mode of convergence, convergence in expectation.

The entire analysis above has its implications for the optimal solution value $\mathbf{C}^*_{\max}$ itself. Indeed, a trivial byproduct is that

$$\mathbf{C}^*_{\max}/(n/4) \to 1(\text{ a.s.}).$$

This is a typical example of what is usually referred to as *probabilistic* value analysis; for large n, the optimal solution value can be guessed with increasing relative accuracy. The *structure* of the optimal solution itself does not lend itself to a similar probabilistic convergence analysis, and so it is much easier to predict the solution value that to say anything about the solution itself.

Now that asymptotic optimality in the relative sense turns out to be within easy reach, it is tempting to examine the asymptotic behavior of the absolute error for $\mathbf{C}_{\max}(A) - \mathbf{C}^*_{\max}$ for several algorithms A. In the case of list scheduling, it converges to a value strictly greater than 0. Can we do better? The insight gained from the worst-case analysis leads us to consider the LPT rule, with the hope that this might demonstrate superior probabilistic performance too. Let us investigate this possibility from a probabilistic perspective by focusing on the absolute difference $\mathbf{d}_j(LPT)$ between the total processing time assigned to M_1 and to M_2 after j jobs have been allocated. Clearly, $\mathbf{d}_n(LPT)/2$ is an upper bound on the absolute error.

Let $\mathbf{p}^{(1)} \leq \mathbf{p}^{(2)} \leq \cdots \leq \mathbf{p}^{(n)}$ denote the sorted list of processing times. The structure of the LPT rule immediately implies that

$$\begin{aligned}\mathbf{d}_n(LPT) &\leq \max\{\mathbf{d}_{n-1}(LPT) - \mathbf{p}^{(1)}, \mathbf{p}^{(1)}\} \\ &\leq \max\{\mathbf{d}_{n-2}(LPT) - (\mathbf{p}^{(1)} + \mathbf{p}^{(2)}), \mathbf{p}^{(2)} - \mathbf{p}^{(1)}, \mathbf{p}^{(1)}\}.\end{aligned}$$

Continuing recursively, we see that

$$\mathbf{d}_n(LPT) \leq \max_{k=1,\ldots,n} \{\mathbf{p}^{(k)} - \sum_{j=1}^{k-1} \mathbf{p}^{(j)}\}.$$

It is trivial to see that, for any fixed $\varepsilon \in (0,1)$,

$$\max_{k=1,\ldots,\lfloor \varepsilon n \rfloor} \{\mathbf{p}^{(k)} - \sum_{j=1}^{k-1} \mathbf{p}^{(j)}\} \leq \mathbf{p}^{(\lfloor \varepsilon n \rfloor)} \tag{9.11}$$

and

$$\max_{k=\lfloor \varepsilon n \rfloor+1,\ldots,n} \{\mathbf{p}^{(k)} - \sum_{j=1}^{k-1} \mathbf{p}^{(j)}\} \leq \mathbf{p}^{(n)} - \sum_{j=1}^{\lfloor \varepsilon n \rfloor} \mathbf{p}^{(j)}. \tag{9.12}$$

For the uniform distribution, it is known that $\mathbf{p}^{(\lfloor \varepsilon n \rfloor)}/\varepsilon \rightarrow 1$ (a.s.). In addition, it is not difficult to show that $\sum_{j=1}^{(\lfloor \varepsilon n \rfloor)} \mathbf{p}^{(j)}/n$ almost surely converges to a positive constant (which depends on ε). Since $\mathbf{p}^{(n)}$ is obviously at most 1, this implies that the right-hand side of (9.12) tends to $-\infty$, and hence the maximum of the right-hand sides of (9.11) and (9.12) tends to ε. This establishes the following theorem.

Theorem 9.21. *For $P2||C_{\max}$, $\boldsymbol{d}_n(LPT) \rightarrow 0$ almost surely.*

Corollary 9.22. *For $P2||C_{\max}$, $\boldsymbol{C}_{\max}(LPT) - \boldsymbol{C}^*_{\max} \rightarrow 0$ almost surely.*

We can use the same bounding technique to bound $\mathbf{d}_n(LPT)$ in expectation. From probability theory, we know that

$$Ex[\mathbf{p}^{(\lfloor \varepsilon n \rfloor)}] = \frac{\lfloor \varepsilon n \rfloor}{n+1}.$$

and

$$Ex\left[\mathbf{p}^{(n)} - \sum_{j=1}^{\lfloor \varepsilon n \rfloor} \mathbf{p}^{(j)}\right] = \frac{n}{n+1} - \sum_{j=1}^{\lfloor \varepsilon n \rfloor} \frac{j}{n+1}. \qquad (9.13)$$

For every fixed $\varepsilon \in (0,1)$, the right-hand side of (9.13) converges to $-\infty$, and hence the absolute error of the LPT rule also converges to 0 in expectation.

Theorem 9.23. *For* $P2||C_{\max}$, $Ex[\boldsymbol{C}_{\max}(LPT) - \boldsymbol{C}^*_{\max}] \to 0$.

This result can be extended for much more general models, such as $Q||C_{\max}$ (see Exercise 9.21).

Can we be still more ambitious? The above analysis for $P2||C_{\max}$ reveals that $\mathbf{d}_n(LPT)$ converges to 0 (a.s.), but does not provide any information on the *rate* at which this occurs. It is possible to estimate this rate; one finds that $\mathbf{d}_n(LPT)/(\log\log n/n)$ is almost surely bounded by a constant. At the same time, the smallest possible difference $\mathbf{d}^*_n$ is known to converge much faster to 0; $\mathbf{d}^*_n/(n^2 2^{-n})$ is almost surely bounded by a constant, so that here convergence occurs at an exponential rate. From that perspective, the best possible heuristic would be one that achieves a similarly fast convergence. Does such a heuristic exist?

The answer to this question is unknown, but an impressive improvement on the rate of convergence of the LPT rule is obtained by a modified version of the differencing method. The differencing method was introduced in studying the probabilistic performance of algorithms for this problem, but there is no known theoretical analysis of this method. As is often the case in the probabilistic analysis of algorithms, it seems to be easier to modify this algorithm to obtain one that is similar, and yet more amenable to analysis. The typical problem is one of dependencies among the data used at various stages of the algorithm. Initially, we have assumed that the processing requirements of the jobs are independent random variables, and this assumption is extremely important to the analyses that we have given. However, after even one iteration of the differencing method, we have lost this essential property. The basic idea is to modify the algorithm so that iterations of the algorithm can be analyzed as essentially independent steps. The technical details of this result are far too complicated to present here; however, we shall give an outline of the modified algorithm and try motivate the modifications made.

The modified differencing method (MD) works in a series of phases, where in each phase most of the jobs are paired, and each pair is replaced by a new job with processing requirement equal to the difference of their processing requirements. Suppose that in the current phase, each $p_j \in (0,U]$; we can subdivide this interval into a number of equal-length subintervals. For each pair, both jobs will be selected from the same interval, and so the size of the new job created will be bounded by the length of the subinterval. The crucial modification is to introduce *randomization* into the algorithm itself. For each subinterval, the algorithm randomly pairs all jobs (assuming that there are an even number of jobs). As a result, the processing requirements of the jobs created in this way will be independent random variables, and will be distributed according to a triangular distribution. Of course, not all subintervals will have an even number of jobs, and something more complicated must be introduced to handle this situation. The algorithm terminates when there are only a specified constant number of jobs remaining, and this small instance can be solved to optimality.

The crux of the analysis is to calculate the way in which the length of the upper bound U changes from iteration to iteration, as compared to the number of jobs that remain. The value of U at any stage is an upper bound on the value of $d_n(MD)$; thus, if we can show that with high probability, the value of U at the end is very small, then we obtain the corresponding bound for the solution given by algorithm. Roughly speaking, it can be shown that the number of jobs in the $(i+1)$th phase is at least $c^{-i}n$ and the upper bound U is at most $c^{(2^i)}n^{-i}$, where c is a constant greater than 2. When the number of remaining jobs reaches the specified constant, the bound on the length of the interval is $n^{-O(\log n)}$.

Theorem 9.24. *For $P2||C_{\max}$, $\boldsymbol{C}_{\max}(MD) - \boldsymbol{C}^*_{\max} \leq \boldsymbol{d}_n(MD)/2 = n^{-O(\log n)}$.*

In the probabilistic analysis of algorithms, it is often the case that the modified algorithm is justified by showing that, in some stochastic sense, the performance of the original algorithm dominates the modified one. It would be nice if this were true in this case, but this remains an important open question.

We have dealt with $P2||C_{\max}$ at length, since this example exhibits many of the ingredients typically encountered in a probabilistic analysis:

- a *combinatorial problem* which is NP-hard and hence difficult to solve;

- a *probability distribution* over all problem instances to generate problem data as realizations of independent and identically distributed random variables;

- a *probabilistic value analysis* that yields an asymptotic characterization of the optimal solution value as a simple function of the problem data;

- a *probabilistic error analysis* of a fast heuristic to prove that its relative or absolute error tends to 0 with increasing problem size in some stochastic sense; and

- a *rate of convergence analysis* that yields some indication of how large the problem size must be in order to demonstrate asymptotic behavior in practice, and this, moreover, allows for further differentiation among the heuristics.

Exercises

9.21. Using the approach suggested in the text, show that the absolute error of the LPT rule applied to $Q||C_{\max}$ a.s. converges to 0. (*Hint*: Rather than looking at the largest difference between the amounts of processing assigned to the various machines, consider the difference between largest and average amount.)

Notes

In these notes, a dagger (†) will be used to indicate that a performance bound is tight or asymptotically tight.

9.1. *Identical machines: classical performance guarantees.* The seminal paper of Graham [1966] contains the performance analysis of the list scheduling rule. In a later paper, Graham [1969] also analyzed the LPT rule as well as the polynomial approximation schemes for $Pm||C_{\max}$ given in Theorem 9.3 and Exercise 9.2. Sahni [1976] gave the first fully polynomial approximation scheme for $Pm||C_{\max}$. Garey, Graham, and Johnson [1978] give a tutorial introduction into the early work on performance guarantees for scheduling parallel machines. Karmarkar and Karp [1982] invented the differencing method; its worst-case analysis is due to Fischetti and Martello [1987].

9.2. *Identical machines: using bisection search for guarantees.* The FFD algorithm and other approximation algorithms for bin packing are reviewed by Coffman, Garey, and Johnson [1984]. Coffman, Garey, and Johnson [1978] proposed the multifit algorithm and proved that $C_{\max}(\mathrm{MF})/C^*_{\max} \leq 1.22 + 2^{-\ell}$. Friesen [1984] improved the constant to 1.2 and gave instances that achieve a ratio of 13/11. Yue [1990] claimed that 13/11 is also an upper bound, but it appears that a complete proof was obtained later by Cao [1995]. Friesen and Langston [1986] gave a refined version of multifit which runs in time $O(n\log n + \ell \cdot n\log m)$ (where the constant embedded within the 'big Oh' notation is big indeed) and has a tight performance bound of $72/61 + 2^{-\ell}$.

The framework of using a relaxed decision procedure as well as the first polynomial approximation scheme for $P||C_{\max}$ are due to Hochbaum and Shmoys [1987]. They also gave the 5/4- and 6/5-relaxed decision procedures of Exercises 9.5 and 9.6, and a 7/6-relaxed decision procedure that runs in $O(n(m^4 + \log n))$ time. Exercise 9.7 is based on the work of Kafura and Shen [1977, 1978].

Several bounds are available that take into account the processing times of the jobs. Note that the probabilistic result of Theorem 9.21 relies on such a (worst-case) bound for list scheduling. Achugbue and Chin [1981] prove two results relating the performance ratio of list scheduling to the value of $\pi = \max_j p_j / \min_j p_j$. If $\pi \leq 3$, then

$$C_{\max}(\mathrm{LS})/C^*_{\max} \leq \begin{cases} 5/3 & \text{if } m = 3,4, \\ 17/10 & \text{if } m = 5, \\ 2 - \frac{1}{3\lfloor m/3 \rfloor} & \text{if } m \geq 6, \end{cases}$$

and if $\pi \leq 2$,

$$C_{\max}(\mathrm{LS})/C^*_{\max} \leq \begin{cases} 3/2 & \text{if } m = 2,3, \\ 5/3 - \frac{1}{3\lfloor m/2 \rfloor} & \text{if } m \geq 4. \end{cases}$$

For the case of LPT, Ibarra & Kim [1977] prove that

$$C_{\max}(\text{LPT})/C^*_{\max} \leq 1 + \frac{2(m-1)}{n} \text{ for } n \geq 2(m-1)\pi.$$

Much less is known about the worst-case performance of approximation algorithms for other minmax criteria. For $P|r_j|L_{\max}$, Exercise 9.8(a) is due to Gusfield [1984]; the result of Exercise 9.8(b) was observed by Hall and Shmoys [1989], who also developed a polynomial approximation scheme for this problem. For the case of equal processing times, $P|r_j, p_j = p|L_{\max}$, Simons [1983] extended Theorem 4.10 to obtain a polynomial algorithm, and Simons and Warmuth [1989] reduced the running time to $O(mn^2)$ by generalizing the approach of Garey, Johnson, Simons, and Tarjan [1981].

As for enumerative optimization methods, Bratley, Florian, and Robillard [1975] proposed an algorithm for $P|r_j, \bar{d}_j|C_{\max}$ and tested it on problems with up to 3 machines and 25 jobs. Carlier [1987] gave an algorithm for $P|r_j|L_{\max}$, which performs well on problems with up to 8 machines and 100 jobs. Dell'Amico and Martello [1995] developed an algorithm for $P||C_{\max}$ and reported good results for random problems with up to 15 machines and 10,000 jobs; relatively small problems ($n \leq 50$ for $m = 10$ or 15) appear to be the hardest ones.

9.3. *Uniform machines: performance guarantees.* Theorem 9.8 is due to Liu and Liu [1974A, 1974B, 1974C], as are the observations given as Exercises 9.9 and 9.10. Morrison [1988] showed that LPT is better that LS, in that

$$C_{\max}(\text{LPT})/C^*_{\max} \leq \max\{\max_i s_i/(2\min_i s_i), 2\}.$$

Cho and Sahni [1980] considered the variant of list scheduling, LS$'$, which puts the next job in the list on the machine on which it will finish earliest, and proved that

$$C_{\max}(\text{LS}')/C^*_{\max} \leq \begin{cases} (1+\sqrt{5})/2 & \text{for } m = 2, \\ (1+(\sqrt{2m-2})/2 & \text{for } m > 2. \end{cases}$$

The bound is tight for $m \leq 6$ but, in general, the worst known examples have a performance ratio of $\lfloor(\log_2(3m-1)+1)/2\rfloor$. This approach followed the work of Gonzalez, Ibarra, and Sahni [1977], who presented the LPT$'$ algorithm and the result given in Theorem 9.9. The improved upper and lower bounds for LPT$'$ were obtained by Dobson [1984] and Friesen [1987].

Friesen and Langston [1983] extended the multifit approach to uniform processors and proved that its performance bound is in between 1.4 and 1.341. They also showed that the decision to order the bins by increasing size is the correct one, since for decreasing bin sizes there exist examples with performance ratio 3/2. Chen [1991] proved the performance bound of Theorem 9.11.

Extending the work of Sahni [1976], Horowitz and Sahni [1976] gave a family of algorithms A_k with running time $O(n^{2m}k^{m-1})$ and performance bound $1+1/k$, which is a fully polynomial approximation scheme for any fixed value of m. The 3/2-approximation algorithm implied by Theorem 9.10 and the polynomial approximation scheme for $Q||C_{\max}$ mentioned at the end of the section are due to Hochbaum and Shmoys [1988]. Williamson (private communication) observed the algorithm given in Exercise 9.11 and its analysis.

9.4. *Unrelated machines: performance guarantees.* The first paper on this subject was by Ibarra and Kim [1977], who considered the greedy method of Theorem 9.12 and Exercise 9.12 and other m-approximation algorithms. Davis and Jaffe [1981] analyzed a number of algorithms and obtained the bound for the RS rule given in Theorem 9.13. The linear programming approach of Theorem 9.14 is due to Potts [1985A]; it was extended by Lenstra, Shmoys, and Tardos [1990] to the algorithm LP$'$ of Theorem 9.15 and Exercise 9.13. Trick [1990] proposed the model described in Exercise 9.15; Tardos (private communication) observed the results given in Exercises 9.14 and 9.15. For $Rm||C_{\max}$, the fully polynomial approximation scheme of Theorem 9.16 is due to Horowitz and Sahni [1976], and the polynomial approximation scheme derived in Exercises 9.16 and 9.17 is due to Lenstra, Shmoys, and Tardos [1990].

Hariri and Potts [1991] performed computational tests with approximation algorithms for $R||C_{\max}$ on problems with up to 50 machines and 100 jobs. Among five constructive heuristics, an implementation of the Lenstra-Shmoys-Tardos algorithm produced the best solutions but required nontrivial running times; a simple iterative improvement scheme, applying job reassignments and interchanges, was still able to achieve substantial improvements, in almost negligible amounts of time. Van de Velde [1993] developed optimization and approximation algorithms for $R||C_{\max}$, based on surrogate duality relaxation of the constraints (9.2). His branch-and-bound method performs reasonably well. His 'duality-based heuristic search' algorithm outperforms the constructive heuristics but, again, the method should be supplemented with some form of iterative improvement.

9.5. *Unrelated machines: impossibilities.* The results in this section are due to Lenstra, Shmoys, and Tardos [1990].

9.6. *Two identical machines: probabilistic analysis.* A basic text in probability theory is Feller [1968, 1971]. For the probabilistic analysis of scheduling algorithms, we refer to the monograph by Coffman and Lueker [1991], the surveys by Coffman, Lueker, and Rinnooy Kan [1988] and by Rinnooy Kan and Stougie [1989], and the annotated bibliography by Karp, Lenstra, McDiarmid, and Rinnooy Kan [1985]. The inequality of Hoeffding [1963] was applied by Coffman et al. in their survey paper to bound the rate of convergence of the relative error of the LS algorithm. Frenk and Rinnooy Kan [1987] proved that the absolute error of the LPT rule converges to 0 almost surely and in expectation, even for $Q||C_{\max}$; see also Coffman, Flatto, and Lueker [1984] and Loulou [1984]. The rate of convergence of this error was investigated by Boxma [1984] and Frenk and Rinnooy Kan [1986]. Karmarkar and

Karp [1982] proposed and analyzed the MD method.