Chapter 9

Makespan Scheduling

In this chapter, we consider the classical MAKESPAN SCHEDULING problem. We are given m machines for scheduling, indexed by the set $M = \{1, \dots, m\}$. There are furthermore given n jobs, indexed by the set $J = \{1, \dots, n\}$, where job j takes $p_{i,j}$ units of time if scheduled on machine i. Let J_i be the set of jobs scheduled on machine i. Then $\ell_i = \sum_{j \in J_i} p_{i,j}$ is the *load* of machine i. The maximum load $\ell_{\max} = c_{\max} = \max_{i \in M} \ell_i$ is called the *makespan* of the schedule.

The problem is NP-hard, even if there are only two identical machines. However, we will derive several constant factor approximations and a PTAS for identical machines and a 2-approximation for the general case.

9.1 Identical Machines

In the special case of *identical machines*, we have that $p_{i,j} = p_j$ for all $i \in M$ and all $j \in J$. Here p_j is called the *length* of job j.

List Scheduling

As a warm-up we consider the following two heuristics for MAKESPAN SCHEDULING. The LIST SCHEDULING algorithm works as follows: Determine any ordering of the job set J, stored in a list L. Starting with all machines empty, determine the machine i with the currently least load and schedule the respective next job j in L on i. The load of i before the assignment of j is called the *starting time* s_j of job j and the load of i after the assignment is called the *completion time* c_j of job j. In the SORTED LIST SCHEDULING algorithm we execute LIST SCHEDULING, where the list L consists of the jobs in decreasing order of length.

Theorem 9.1. *The* LIST SCHEDULING *algorithm is a* 2*-approximation for* MAKESPAN SCHEDULING *on identical machines.*

Proof. Let T^* be the optimal makespan of the given instance. We show that $s_j \leq T^*$ for all $j \in J$. This implies $c_j = s_j + p_j \leq T^* + p_j \leq 2 \cdot T^*$ for all $j \in J$, since we clearly must have $T^* \geq p_j$ for all $j \in J$.

Assume that $s_j > T^*$ for some $j \in J$. Then we have that the load *before* the assignment of j is $\ell_i > T^*$ for all $i \in M$. Thus the jobs $J' \subseteq J$ scheduled before j by the algorithm have total length $\sum_{j' \in J'} p_{j'} > m \cdot T^*$. On the other hand, since the optimum solution schedules all jobs J until time T^* we have $\sum_{j \in J} p_j \leq m \cdot T^*$. A contradiction and the LIST SCHEDULING algorithm must start all jobs not later than time T^*. $\square$

Here we show that Sorted List Scheduling is a 3/2-approximation, but one can actually prove that the algorithm is a 4/3-approximation.

Theorem 9.2. *The* Sorted List Scheduling *algorithm is a* $3/2$*-approximation for* Makespan Scheduling *on identical machines.*

Proof. Let T^* be the optimal makespan of the given instance. Partition the jobs $J_L = \{j \in J : p_j > T^*/2\}$ and $J_S = J - J_L$, called *large* and *small* jobs. Notice that there can be at most m large jobs: Assume that there are more than m such jobs. Then, in any schedule, including the optimal one, there must be at least two such jobs scheduled on some machine. Since the length of a large job is more than $T^*/2$, this contradicts that T^* is the optimal makespan.

Since there are at most m large jobs and the algorithm schedules those first and hence on individual machines, we have that each large job completes not later than T^*, i.e., $c_j \le T^*$ for all $j \in J_L$. Thus, if a job completes later than T^* it must be a small job having length at most $T^*/2$. Since each job starts not later than T^* we have $c_j \le T^* + p_j \le 3/2 \cdot T^*$ for every small job $j \in J_S$. □

Polynomial Time Approximation Scheme

In this section we give a *polynomial time approximation scheme (PTAS)* for Makespan Scheduling on identical machines. This means, for any error parameter $\varepsilon > 0$, there is an algorithm which determines a $(1+\varepsilon)$-approximate solution with running time polynomial in the input size, but arbitary in $1/\varepsilon$. We will give a PTAS with running time $O\left(n^{2k} \cdot \lceil \log_2 1/\varepsilon \rceil\right)$, where $k = \lceil \log_{1+\varepsilon} 1/\varepsilon \rceil$.

The two main ingredients of the algorithm are these:

(1) Firstly, assume that we are given the optimal makespan T^* at the outset. Then we can try to construct a schedule with makespan at most $(1+\varepsilon) \cdot T^*$. But how do we determine the number T^*? It turns out that we can perform *binary search* in an interval $[\alpha, \beta]$, where α is any lower bound on T^* and β any upper bound on T^*. This binary search will enable us to eventually find a number B, which is within $(1+\varepsilon)$ times T^* and where the number of binary search iterations depends on the error parameter ε.

(2) Secondly, assume that the number of *distinct* values of job lengths is a constant k, say. Then we can determine all configurations of jobs that do not violate a load bound of t if scheduled on a single machine. This is the basis of a dynamic programming scheme to determine a schedule on m machines. Of course, this approach involves rounding the original job lengths to constantly many values, which introduces some error. The error can be controled by adjusting the constant k of distinct job lengths at the expense of running time and space requirement for the dynamic programming table.

Consider the instance J with jobs of lengths $p_1, \ldots, p_n$. Let t be a parameter and let $m(J,t)$ be the smallest number of machines required to schedule the jobs J having makespan at most t. With this definition, the minimum makespan T^* is given by $T^* = \min\{t : m(J,t) \le m\}$. We will later perform binary search on the parameter t, in the interval $[\alpha, \beta]$, where $\alpha = \max\{\max_{j \in J} p_j, 1/m \cdot \sum_{j \in J} p_j\}$ and $\beta = 2 \cdot \alpha$. Notice that α is a lower bound on T^* and β an upper bound on T^*.

Dynamic Programming. Assume for now that $|\{p_1, \ldots, p_n\}| = k$, i.e., there are k distinct job lengths. Fix an ordering of the jobs lengths. Then a k-tuple $(i_1, \ldots, i_k)$ describes for any $\ell \in \{1, \ldots, k\}$ the number i_ℓ of jobs having the respective length. For any k-tuple $(i_1, \ldots, i_k)$ let $m(i_1, \ldots, i_k, t)$ be the smallest number of machines needed to schedule these jobs having makespan at most t. For a given parameter t and an instance $(n_1, \ldots, n_k)$ with $\sum_{\ell=1}^{k} n_\ell = n$, we first compute the set Q of all k-tuples $(q_1, \ldots, q_k)$ such that $m(q_1, \ldots, q_k, t) = 1$, $0 \leq q_\ell \leq n_\ell$ for $\ell = 1, \ldots, k$, i.e., all sets of jobs that can be scheduled on a single machine with makespan at most t. Clearly, Q contains at most $O\left(n^k\right)$ elements. Having these numbers computed, we determine the entries $m(i_1, \ldots, i_k, t)$ for every $(i_1, \ldots, i_k) \in \{0, \ldots, n_1\} \times \cdots \times \{0, \ldots, n_k\}$ of a k-dimensional table as follows: The table is initialized by setting $m(q, t) = 1$ for every $q \in Q$. Then we use the following recurrence to compute the remaining entries:

$$m(i_1, \ldots, i_k, t) = 1 + \min_{q \in Q} m(i_1 - q_1, \ldots, i_k - q_k, t).$$

Computing each entry takes $O\left(n^k\right)$ time. Thus the entire table can be computed in $O\left(n^{2k}\right)$ time, thereby determining $m(n_1, \ldots, n_k, t)$ in polynomial time provided that k is a constant.

Rounding. Let $\varepsilon > 0$ be an error parameter and let $t \in [\alpha, \beta]$ as defined above. We say that a job j is small if $p_j < \varepsilon \cdot t$. Small jobs are removed from the instance for now. The rest of the job lengths are rounded down as follows: If a job j has length $p_j \in [t \cdot \varepsilon \cdot (1+\varepsilon)^i, t \cdot \varepsilon \cdot (1+\varepsilon)^{i+1})$ for $i \geq 0$, it is replaced by $p'_j = t \cdot \varepsilon \cdot (1+\varepsilon)^i$. Thus there can be at most $k = \lceil \log_{1+\varepsilon} 1/\varepsilon \rceil$ many distinct job lengths. Now we invoke the above dynamic programming scheme and determine a the optimal number of machines for scheduling these jobs if the makespan is at most t. Since the rounding reduces the length of each job by a factor of at most $(1+\varepsilon)$, the computed schedule has makespan at most $(1+\varepsilon) \cdot t$ when considering the original job lengths. Now we schedule the small jobs greedily in leftover space and open new machines if needed. Clearly, whenever a new machine is opened, all previous machines must be loaded to an extent of at least t. Denote by $a(J, t, \varepsilon)$ the number of machines used by this algorithm. Recall that the makespan is at most $(1+\varepsilon) \cdot t$.

Lemma 9.3. *We have that $a(J, t, \varepsilon) \leq m(J, t)$.*

Proof. If the algorithm does not open any new machines for small jobs, then the assertion clearly holds since the rounded down jobs have been scheduled optimally with makespan t. In the other case, all but the last machine are loaded to the extent of t. Hence, the optimal schedule of J having makespan t must also use at least $a(J, t, \varepsilon)$ machines. □

Corollary 9.4. *We have that $T = \min\{t : a(J, t, \varepsilon) \leq m\} \leq \min\{t : m(J, t) \leq m\} = T^*$.*

Binary Search. If T could be determined with no additional error during the binary search, then clearly we could use the above algorithm to obtain a schedule with makespan at most $(1+\varepsilon) \cdot T^*$. Next, we will specify the details of the binary search and show how to control the error it introduces. The binary search is performed in the interval $[\alpha, \beta]$ as defined above. Thus, the length of the available interval is $\beta - \alpha = \alpha$ at the start of the search and it reduces by a factor of two in each iteration. We continue the search until it drops to a length of at most $\varepsilon \cdot \alpha$. This will require $\lceil \log_2 1/\varepsilon \rceil$ many iterations. Let B be the right endpoint of the interval $[A, B]$ we terminate with.

Lemma 9.5. *We have that $B \leq (1+\varepsilon) \cdot T^*$.*

Proof. Clearly $T = \min\{t : a(J, t, \varepsilon) \leq m\}$ must be in the interval $[B - \varepsilon \cdot \alpha, B]$. Hence

$$B \leq T + \varepsilon \cdot \alpha \leq (1+\varepsilon) \cdot T^*,$$

where we have used $T^* \geq \alpha$ and Corollary 9.4. □

For $t \leq B$ this directly gives the result we wanted to show:

Theorem 9.6. *For any $0 < \varepsilon \leq 1$ the algorithm produduces a schedule with makespan at most $(1+\varepsilon)^2 \cdot T^* \leq (1+3\varepsilon) \cdot T^*$ within running time $O\left(n^{2k} \cdot \lceil \log_2 1/\varepsilon \rceil\right)$, where $k = \lceil \log_{1+\varepsilon} 1/\varepsilon \rceil$.*

9.2 Unrelated Machines

Here we give a 2-approximation algorithm for Makespan Scheduling on *unrelated machines*, which means that job j takes time $p_{i,j}$ if scheduled on machine i. The algorithm is based on a suitable LP-formulation and a procedure for rounding the LP.

An obvious integer program for this problem is the following: Let $x_{i,j}$ be a variable indicating if job j is assigned to machine i. The objective is to minimize the makespan. The first set of constraints ensures that each job is scheduled on one of the machines and the second ensures that each machine has a load of at most t.

$$\begin{aligned} \text{minimize} \quad & t \\ \text{subject to} \quad & \sum_{i \in M} x_{i,j} = 1 \quad j \in J, \\ & \sum_{j \in J} p_{i,j} x_{i,j} \leq t, \quad i \in M, \\ & x_{i,j} \in \{0,1\}. \end{aligned}$$

If we relax the constraints $x_{i,j} \in \{0,1\}$ to $x_{i,j} \in [0,1]$, it turns out that this formulation has unbounded integrality gap. (It is left as an exercise to show this.) The main cause of the problem is an "unfair" advantage of the LP-relaxation: If $p_{i,j} > t$, then we must have $x_{i,j} = 0$ in any feasible integer solution, but we might have $x_{i,j} > 0$ in feasible fractional solutions. However, we can not formulate the statement "if $p_{i,j} > t$ then $x_{i,j} = 0$" in terms of linear constraints.

Parametric Pruning. We will make use of a technique called *parametric pruning* to overcome this difficulty. Let the parameter t be a "guess" of a lower bound for the actual makespan T^*. Of course, we will do binary search on t in order to determine a suitable value in an outside loop. However, having a value for t fixed, we are now able to enforce constraints $x_{i,j} = 0$ for all machine-job pairs i, j for which $p_{i,j} > t$. Define $S_t = \{(i,j) : p_{i,j} \leq t\}$. We now define a family LP(t) of linear programs, one for each value of the parameter t. LP(t) uses the variables $x_{i,j}$ for which $(i,j) \in S_t$ and asks if there is a feasible solution using the restricted assignment possibilities, only.

$$\begin{aligned} \text{minimize} \quad & 0 & \qquad (\textsc{lp}(t)) \\ \text{subject to} \quad & \sum_{i:(i,j)\in S_t} x_{i,j} = 1 \quad j \in J, \\ & \sum_{j:(i,j)\in S_t} p_{i,j} x_{i,j} \leq t, \quad i \in M, \\ & x_{i,j} \geq 0 \quad (i,j) \in S_t. \end{aligned}$$

Extreme Point Solutions. With a binary search, we find the smallest value for t such that $\textsc{lp}(t)$ has a feasible solution. Let T be this value and observe that $T^* \geq T$, i.e., the actual makespan is bounded from below by T. Our algorithm will "round" an extreme point solution of $\textsc{lp}(T)$ to yield a schedule with makespan at most $2 \cdot T^*$. Extreme point solutions to $\textsc{lp}(T)$ have several useful properties.

Lemma 9.7. *Any extreme point solution to* $\textsc{lp}(T)$ *has at most* $n + m$ *many non-zero variables.*

Proof. Let $r = |S_T|$ represent the number of variables on which $\textsc{lp}(T)$ is defined. Recall that a feasible solution is an extreme point solution to $\textsc{lp}(T)$ if and only if it sets r many linearly independent constraints to equality. Of these r linearly independent constraints, at least $r - (n + m)$ must be chosen from the third set of constraints, i.e., of the form "$x_{i,j} \geq 0$". The corresponding variables are set to zero. So, any extreme point solution has at most $n + m$ many non-zero variables. □

Let x be an extreme point solution to $\textsc{lp}(T)$. We will say that job j is *integrally set* if $x_{i,j} \in \{0, 1\}$ for all machines i. Otherwise, i.e., $x_{i,j} \in (0, 1)$ for some machine i, job j is said to be *fractionally set.*

Corollary 9.8. *Any extreme point solution to* $\textsc{lp}(T)$ *must set at least* $n - m$ *many jobs integrally.*

Proof. Let x be an extreme point solution to $\textsc{lp}(T)$ and let α and β be the number of jobs that are integrally and fractionally set by x, respectively. Each job of the latter kind is assigned to at least 2 machines and therefore results in at least 2 non-zero entries in x. Hence we get $\alpha + \beta = n$ and $\alpha + 2\beta \leq n + m$. Therefore $\beta \leq m$ and $\alpha \geq n - m$. □

Algorithm. The algorithm starts by computing the range in which it finds the right value for T. For this it constructs a greedy schedule, in which each job is assigned to the machine on which it has the smallest length. Let α be the makespan of this schedule. Then the range is $[\alpha/m, \alpha]$ (and it is an exercise to show that α/m is indeed a lower bound on T^*).

The LP-rounding algorithm is based on several interesting properties of extreme point solutions of $\textsc{lp}(T)$, which we establish now. For any extreme point solution x for $\textsc{lp}(T)$ define a bipartite graph $G = (M \cup J, E)$ such that $(i, j) \in E$ if and only if $x_{i,j} > 0$. Let $F \subseteq J$ be the fractionally set jobs in x and let H be the subgraph of G induced by the vertex set $M \cup F$. Clearly $(i, j) \in E(H)$ if $0 < x_{i,j} < 1$. A matching in H is called *perfect* if it matches every job $j \in F$. We will show and use that the graph H admits perfect matchings.

We say that a connected graph on a vertex set V is a *pseudo tree* if it has at most $|V|$ many edges. Since the graph is connected, it must have at least $|V| - 1$ many edges. So,

Algorithm 9.1 SCHEDULE UNRELATED

```
Input.   $J$, $M$, $p_{i,j}$ for all $i \in M$ and $j \in J$

Output.  $x_{i,j} \in \{0,1\}$ for all $i \in M$ and $j \in J$

Step 1. By binary search in $[\alpha/m, \alpha]$ compute smallest value $T$ of the parameter $t$ such
        that LP$(t)$ has a feasible solution.

Step 2. Let $x$ be an extreme point solution for LP$(T)$.

Step 3. Construct graph $H$ and find perfect matching $P$.

Step 4. Round in $x$ all fractionally set jobs according to the matching $P$.
```

it is either a tree or a tree with an additional single edge (closing exactly one cycle). A graph is a *pseudo forrest* if each of its connected components is a pseudo tree.

Lemma 9.9. *We have that G is a pseudo forrest.*

Proof. We will show that the number of edges in each connected component of G is bounded by the number of vertices in it. Hence, each connected component is a pseudo tree.

Consider a connected component G_c. Restrict LP(T) and the extreme point solution x to the jobs and machines of G_c, only, to obtain LP$_c(T)$ and x_c. Let $x_{\bar{c}}$ represent the rest of x. The important observation is that x_c must be an extreme point solution for LP$_c(T)$. Suppose that this is not the case. Then, x_c is a convex combination of two feasible solutions to LP$_c(T)$. Each of these, together with $x_{\bar{c}}$ form a feasible solution for LP(T). Therefore x is a convex combination of two feasible solutions to LP(T). But this contradicts the fact that x is an extreme point solution. With Lemma 9.7 G_c is a pseudo tree. □

Lemma 9.10. *Graph H has a perfect matching P.*

Proof. Each job that is integrally set in x has exactly one edge incident at it in G. Remove these jobs together with their incident edges from G. The resulting graph is clearly H. Since an equal number of edges and vertices have been removed from the pseudo forrest G, H is also a pseudo forrest.

In H, each job has a degree of at least two. So, all leaves in H must be machines. Keep matching a leaf with the job it is incident to and remove them both from the graph. (At each stage all leaves must be machines.) In the end we will be left with even cycles (since we started with a bipartite pseudo forrest.) Match alternating edges of each cycle. This gives a perfect matching P. □

Theorem 9.11. *Algorithm* SCHEDULE UNRELATED *is a 2-approximation for* MAKESPAN SCHEDULING *on unrelated machines.*

Proof. Clearly $T \leq T^*$ since LP(T^*) has a feasible solution. The extreme point solution x to LP(T) has a fractional makespan of at most T. Therefore, the restriction of x to integrally set jobs has an integral makespan of at most T. Each edge (i,j) of H satisfies $p_{i,j} \leq T$. The perfect matching found in H schedules at most one extra job on each machine. Hence, the total makespan is at most $2 \cdot T \leq 2 \cdot T^*$ as claimed. The algorithm clearly runs in polynomial time. □

It is an exercise to show that the analysis is tight for the algorithm.