Lecture 2: Scheduling on Parallel Machines

Loris Marchal

October 17, 2012

Parallel environment (*alpha* in Graham's notation):
- P parallel identical
- Q uniform machines: each machine has a given speed $speed_i$, and all jobs have a size $size_j$, the processing time is given by $size_j/speed_i$
- R unrelated machines: the processing time of job j on machine i is given by $p_{i,j}$, without any other constraints

Sometimes the number of processors is fixed: for example, P2, F2, or Pm.

1 Minimizing makespan on identical machines: $P||C_{\max}$

Even this simple parallel scheduling problem is NP-complete. There are straightforward reductions from classical NP-Complete problems:
- Partition: Given n integers a_i whose sum is S, is there a subset I of these numbers such that $\sum_{i\in I} a_i = \sum_{i\notin I} a_i = S/2$?
- 3-Partition: Given $3m$ integers a_i whose sum is mB, are there m subsets of 3 a_i each $S_1, \dots S_m$ such that $\forall k, \sum_{i\in S_k} a_i = B$? The problem remains NP-complete if the a_i are such that $B/4 < a_i < B/2$.

While Partition is only weakly NP-complete, 3-Partition is unary NP-complete (it remains NP-complete even if its input is encoded in unary).

A *List-Scheduling* algorithm is an algorithm that maps tasks to available resources without introducing idle times if it is not needed. Using m processors, Graham proves that any list-scheduling algorithm is a $2 - 1/m$-approximation algorithm.

Theorem 1. *A list-scheduling algorithm is a* $2-1/m$*-approximation algorithm for* $P||C_{\max}$.

Proof. Consider the last task k to finish, and t its starting time. Before time t, all machines were busy (otherwise k would have been started earlier). Thus, the total processing time of all tasks except k is larger than $t \times m$:

$$\sum_{i\neq k} p_i \geq m \times t$$

Thus,

$$t \leq \frac{\sum_i p_i - p_k}{m} \leq C_{\max}^{OPT} - \frac{p_k}{m}$$

And

$$C_{\max} = t + p_k \leq C_{\max}^{OPT} + \left(1 - \frac{1}{m}\right) p_k \leq \left(2 - \frac{1}{m}\right) C_{\max}^{OPT}. \quad \square$$

This bound is tight, which means that the inequality can be an equality in certains cases. Let us for example consider the problem of processing 3 tasks with running times 1,1,2 on 2 machines. If both task of duration 1 are scheduled first, the makespan will be 3 instead of 2.

We can improve this bound using a specific list-scheduling algorithm. LPT, which stands for Longest Processing Time first, considers a list of tasks sorted by non-increasing processing time, and schedule them on the available resources.

Theorem 2. *LPT is a* $(4/3 - 1/3m)$*-approximation algorithm for* $P||C_{\max}$.

Proof. Let S be the schedule output by LPT on a given instance of the problem. We consider l, the last task to finish in S. We shorten the instance to tasks $1, 2, \dots l$ (with $p_1 \geq p_2 \geq \dots \geq p_n$): this does not modify the solution of LPT, but only give some advantage to the optimal solution. The approximation ratio will only be better. On this instance, p_l is the smallest processing time, noted $p_{\min}$.

Lemma 1. *If* $p_{\min} > C_{\max}^{OPT}/3$*, then* $C_{\max} = C_{\max}^{OPT}$.

Proof. Assume that $p_{\min} > C_{\max}^{OPT}/3$ and focus on the optimal schedule. Since $C_{\max}^{OPT} < 3p_{\min}$, at most 2 tasks are processed on each machine. We denote by i_1 and i_2 the tasks processed on machine i, with $p_{i_1} \geq p_{i_2}$. We assume that machines are sorted such that $p_{i_1} \geq p_{i'_1}$ for $i < i'$.

We can assume that $p_{i_2} \leq p_{i'_2}$ for $i < i'$ (otherwise we exchange them). We prove that this schedule is the one given by LPT:

- If LPT gives at most 2 tasks per machine, this is the schedule of LPT.
- If LPT gives 3 tasks to a given machine, let j be the third task put on a machine with 2 tasks. Since $n \leq 2m$, there exists one machine with a task k which is alone on a machine in LPT but not in OPT. Since LPT put task j on the least loaded machine, it means that k is longer than 2 other tasks: $p_k \geq 2p_{\min} > 2C_{\max}^{OPT}/3$. However, in OPT, k is processed with another task on a single machine, whose running time is thus larger than $p_k + p_{\min} > C_{\max}^{OPT}$, which contradicts its optimality. □

We now have to consider the case $p_{\min} \leq C_{\max}^{OPT}/3$. Then, we refine the bound from the previous proof:

$$C_{\max} \leq C_{\max}^{OPT} + \left(1 - \frac{1}{m}\right) p_{\min} \leq \left(\frac{4}{3} - \frac{1}{3m}\right) C_{\max}^{OPT}. \quad \square$$

2 Adding precedence constraints: $P|prec|C_{\max}$

We now introduce precedence constraint between tasks:

- precedence constraints: $i \to j$ means that j cannot start before i completes
- often modeled using a Directed Acyclic Graph
- any path of precedence is a lower bound on the optimal makespan
- critical path: (one of) the longest path
- precedence may have special structure:
  - prec : arbitrary precedence constraints

– intree: (outtree) intree (or outtree) precedences
– chains: chain precedences

The Graham list-scheduling approximation ratio can be adapted in this case.

Theorem 3. *A list-scheduling algorithm is a $2-1/m$-approximation algorithm for $P|prec|C_{\max}$.*

Proof. Let l be the tasks which finishes last and t_l its starting time. Let $l-1$ be the predecessor of l which finishes the last one. Because of precedence constraint, we have $t_l \geq t_{l-1} + p_{l-1}$. We construct a series of jobs preceding each other, starting at 1 (which has no predecessor) such that $1 \to 2 \to \cdots \to l-1 \to l$. Since this is a precedence path, $C_{\max}^{OPT} \geq \sum_{i=1}^{l} p_i$.

We now state the important observation of the proof: between the finish time $t_i + p_i$ of one task of this chain and the starting time of the next task t_{i+1}, all machines are busy (otherwise task $i+1$ would have been started earlier). The same is true between time 0 and t_1. The overall amount of work processed during these busy times is:

$$m(t_1 + \sum_{i=1}^{l-1}(t_{i+1} - (t_i + p_i)) \leq \sum p_i - \sum_{i=1}^{l-1} p_i \leq mC_{\max}^{OPT} - \sum_{i=1}^{l-1} p_i$$

Thus,

$$\begin{aligned} t_1 + \sum_{i=1}^{l-1}(t_{i+1} - (t_i + p_i)) \leq C_{\max}^{OPT} - \frac{1}{m}\sum_{i=1}^{l} p_i + \sum_{i=1}^{l} p_i \\ t_l \leq C_{\max}^{OPT} + \left(1 - \frac{1}{m}\right)\sum_{i=1}^{l-1} p_i \end{aligned}$$

Since $C_{\max} = t_l + p_l$, we get the expected result. □

3 A 2-approximation algorithm for unrelated machines: $R||C_{\max}$

3.1 Linear Programming formulation and integrality gap

The results stated in this section come from the article "Approximation Algorithms for Scheduling Unrelated Parallel Machines", published in 1990 by Lenstra, Shmoys and Tardos.

The problem $R||C_{\max}$ is NP-complete as a generalization of $P||C_{\max}$. It can be formulated as an integer program (IP). We use variable $x_{i,j} \in \{0,1\}$ to describe the schedule: $x_{i,j} = 1$ if and only if task i is schedule on machine j. Variable C denotes the makespan:

$$(IP) \begin{cases} \text{Minimize } C \text{ under the constraints:} \\ \forall i, \sum_j x_{i,j} p_{i,j} \leq C \\ \forall j, \sum_i x_{i,j} p_{i,j} = 1 \\ \forall i,j, x_{i,j} \in \{0,1\} \end{cases}$$

The following straightforward result states that it is equivalent to solve the integer program or the scheduling problem.

Theorem 4. *The value of the objective function in an optimal solution of IP is equal to the optimal makespan $C_{\max}^{OPT}$.*

Solving an integer problem is an NP-complete problem. However, solving a linear program (where all variables are rational) can be done in polynomial time, and efficient algorithm exists such as the simplex. We use a relaxation of this integer program as a basis for a 2-approximation.

$$(LP) \begin{cases} \text{Minimize } C \text{ under the constraints:} \\ \forall i, \sum_j y_{i,j} p_{i,j} \leq C \\ \forall j, \sum_i y_{i,j} p_{i,j} = 1 \\ \forall i, j, 0 \leq y_{i,j} \leq 1 \end{cases}$$

The usual way of deriving approximation algorithms based on linear programming relaxation is the following. Based on the optimal solution of LP, we aim a constructing a schedule for the original problem. Since the solution of the relaxed problem is usually not feasible, the performance of the obtained schedule is reduced compared to the LP relaxation. If we denote by $C_{\max}^*$ the value of the objective in the LP relaxation and $C_{\max}$ the makespan of the constructed schedule, we hope that we can construct a schedule with $C_{\max} \leq \alpha C_{\max}^*$.

However, as the relaxed linear program does not capture all the constraints of the original problem, there may exist an instance I for which the ratio $C_{\max}^{OPT}(i)/C_{\max}^*(I)$ is large. Thus, as $C_{\max} \leq C_{\max}^{OPT}$, we have:

$$\alpha \geq \frac{C_{\max}(I)}{C_{\max}^*(I)} \geq \frac{C_{\max}^{OPT}(I)}{C_{\max}^*(I)}$$

Since this is true for all instances, the best approximation that we can expect is lower bounded:

$$\alpha \geq \max_{\text{instances } I} \frac{C_{\max}^{OPT}(I)}{C_{\max}^*(I)}$$

This maximum is called the *integrality gap* of the LP relaxation. In particular in our problem this gap is an issue: consider the problem of scheduling a single task on m machines. The running time of the task is m on any machine. The optimal makespan is $C_{\max}^{OPT} = m$, but in the LP relaxation, it is possible to allocate a fraction $1/m$ of the task on each machine, leading to a objective value $C_{\max}^* = 1$. Thus, $\alpha = 1$ and it is not possible to derive a constant factor approximation algorithm with this method.

3.2 Approximation algorithm

On of the problem of using the relaxed linear program is that it is oblivious to a very simple lower bound on the makespan: if task i is processed on machine j, the makespan is not smaller than $p_{i,j}$. However, we do not know a priori which machine will process each job.

We use this bound in a different way. Assume on the contrary that we know the optimal value for the makespan C. Then, we know that a task i can only be processed on machines j such that $p_{i,j} \le C$. We denote by S_C the set of possible mappings with makespan C:

$$S_C = \{(i,j), p_{i,j} \le C\}.$$

The following polytope defines the solutions of the relaxed LP with this additional constraints for a makespan not larger than C:

$$(LP_C) \begin{cases} \forall i, \displaystyle\sum_{j,(i,j)\in S_C} y_{i,j}p_{i,j} \le C \\ \forall j, \displaystyle\sum_{i,(i,j)\in S_C} y_{i,j}p_{i,j} = 1 \\ \forall (i,j) \in S_C, y_{i,j} \ge 0 \end{cases}$$

As previously with LP, if $C \ge Cmaxopt$, then LP_C is feasible. We will show later that from a feasible solution of LP_C, it is possible to build a schedule with makespan $C_{\max} \le 2C$.

Theorem 5. *For a given C, if LP_C is feasible, then one can find a schedule with makespan at most $2C$ in polynomial time.*

We first how to solve the original problem: computing a 2-approximation algorithm for $R||C_{\max}$ using this result. The solution is based on a dichotomic search using the following algorithm:

1. Initialize $L = 1$ and $U = n \max_{i,j} p_{i,j}$
2. While $U - L > 1$ do
   (a) Let $C = (L + U)/2$
   (b) If LP_C then $U \leftarrow C$ otherwise $L \leftarrow C$.
3. Let $C^* \leftarrow U$. Note that this is the minimum C^* for which LP_{C^*} is feasible. We build a schedule with makespan at most $2C^*$ using Theorem 5.

Theorem 6. *This algorithm produces a 2-approximation to $R||C_{\max}$ in polynomial time.*

Proof. The algorithm runs in time $\log(n \max_{i,j} p_{i,j}) \times T_1 + T_2$ where T_1 is the time needed to solve the linear program and T_2 the time needed to build a solution. Since both are polynomial, the algorithm runs in polynomial time.

Since there exists a solution to IP with objective $C_{\max}^{OPT}$, in particular $LP_{C_{\max}^{OPT}}$ is feasible. The algorithm the smallest value C^* for which LP_{C^*} is feasible, $C^* \le C_{\max}^{OPT}$ and the produces schedule has a makespan $C_{\max} \le 2C^* \le 2C_{\max}^{OPT}$. □

We now move to the construction of a schedule with makspan smaller than $2C$ from a solution of LP_C.

Proof of Theorem 5. Let v the number of variables in the linear program LP_C ($v = |S_C|$). The linear program has v variables and $n+m+v$ constraints. A vertex of the polyhedron is defined by v constraints. Thus, there exists a point y^* in the polytope for which v

among the $n + m + v$ constraints are equalities. At most, there are $n + m$ of the last constraints ($y_{i,j} \geq 0$) which are not equalities for y^*, and thus at most $n + m$ non-zero variables in y^*.

We construct a bipartite graph G representing the non-zero variables: the vertices are the machines and the tasks, and there is an edge between i and j if and only if $y^*_{i,j} > 0$. We assume that G is connected (if G is not connected, we process each connected component as follows). The number of edges in G is at most $n + m$ while the number of vertices is $n + m$. Thus, G is either a tree or a tree plus one edge. In particular, G has at most one cycle.

We call *leaf task* a task with degree 1 in G . Not that for each such leaf task i, there is a unique machine j with $y^*_{i,j} \geq 0$ and thus, $y^*_{i,j} = 1$. Let T_j be the set of leaf tasks connected to machine j. Note that we have

$$\sum_{i \in T_j} p_{i,j} = \sum_{i \in T_j} y^*_{i,j} \leq C^*$$

since y^* is a point of LP_{C^*}.

We first delete all these leaf tasks from G. In the remaining graph, all tasks have degree at least 2. We map the remaining tasks as follows:

1. $M \leftarrow \emptyset$
2. While G is not a cycle or is not empty, do
   (a) Find a machine j with degree 1. Let i be a task connected to j. Map i to i and supress both vertices in G. If any machine has degree 0, delete it as well.
3. If G is a cycle, find a matching M' in the cycle, which maps each task to a unique machine, and a single task per machine, and $M \leftarrow M \cup M'$

Note that pruning the graph always ends with an empty graph or a unique cycle. This is why the graph G originally contains a single cycle. Moreover, since all tasks of degree 1 have been deleted before, the only remaining vertices with degree 1 are machines. When removing such a machine and the corresponding task, it only affects the degree of other machines, not tasks. At the end, a single taks has been mapped to each machine in this step. Thus, the additionnal workload for each machine is smaller than C^* and the obtained mapping has makespan smaller than $2C^*$. □

4 Shop scheduling problems

4.1 Notations for shop and job-shop scheduling

- jobs: $J_1, \ldots, J_m$
- processors: $P_1, \ldots, P_m$
- job J_j consists in operations $J_{1,j}, J_{2,j}, \ldots, J_{n_j,j}$
- operation $J_{i,j}$ takes time $p_{i,j}$ and must be processed by machine $\mu_{i,j}$

	no precedence	precedence chains
arbitrary structure		J: job-shop
$\forall j, n_j = m, \mu_{i,j} = P_i$	O: open-shop	F: flow-shop

4.2 List-scheduling for O$||C_{\max}$

We prove that any list-scheduling algorithm is a 2-approximation for O$||C_{\max}$ first, two simple bounds on the optimal makespan:

- $\forall j, \quad OPT \geq \sum_{i=1}^{m} p_{i,j}$ (all operations of each job must be processed)
- $\forall i, \quad OPT \geq \sum_{j=1}^{n} p_{i,j}$ (each machine must process all required operations)

Let (M, l) be the pair machine/job which finishes the last one. At any time step before $C_{\max}$, either M is busy processing something or l is busy being processed by some machine, so:

$$C_{\max} \leq \sum_{j=1}^{n} P_{M,j} + \sum_{i=1}^{m} P_{i,l} \leq 2OPT$$