CS-E400204: Approximation Algorithms **Spring (Fifth Period) 2022**

Lecture 7 (May 16): Parallel Machines Scheduling

Lecturer: Kamyar Khodamoradi *Scribe: Martino Ciaperoni*

The Parallel Machine Scheduling problem (also known as unrelated machine scheduling or for short $R \setminus \setminus C_{max}$) is a central problem in such fields as combinatorial optimization and operations research.

Broadly speaking, in what follows, first the problem is formalized, and then an approximation algorithm based on *Linear Programming (LP) relaxation* is discussed. Specifically, the approximation algorithm relies on a technique referred to as *parametric pruning*.

Roadmap More specifically, the notes are organized as follows.

- The Parallel Machine Scheduling problem formulation is presented in Section 7.1.
- In Section 7.2, we present an obvious formulation of the problem as an integer linear program.
- In Section 7.3, we explain how the integer linear program can be converted into a suitable linear program. As regards this linear program:
  - We first analyse its properties in Section 7.3.1.
  - We describe a rounding strategy which guarantees an approximation factor in Section 7.3.2.
  - We conclude by providing the details of two fundamental ingredients of the approximation scheme in Section 7.3.2.1 and 7.3.2.2.

7.1 Problem Formulation

In this problem, in plain words, we are given a set of machines and jobs to be processed (arriving all at the same time). The cost of processing jobs is allowed to vary over machines. In the simpler case of identical machines, a PTAS is known. Nevertheless, for the general problem considered in this notes whereby the cost is allowed to be machine-dependent, there is probably no PTAS. The goal is to assign jobs to machines such that the makespan (i.e., completion time) of *all* jobs is as short as possible. It is worth emphasizing that the makespan is the total time such that at least one machine is processing a job. Figure 7.1 shows a pictorial representation of the problem.

The problem is formalized next.

Problem 1 (Parallel Machine Scheduling) *Let $\mathcal{J} = \{J_1, ...J_i, ...J_n\}$ be a set of jobs and $\mathcal{M} = \{M_1, ...M_i, ...M_m\}$ be a set of machines. Furthermore let $p_{i,j} \in \mathbb{N}_{>0}$ be the processing time of job j scheduled on machine i. The* Parallel Machine Scheduling *problem asks for a schedule (or assignment) of jobs to machines:*

$$\sigma : \mathcal{J} \to \mathcal{M}$$

such that the total makespan t is minimized. In symbols, the goal is to minimize:

$$\max_{i \in M} \sum_{j:\sigma(j)=i} p_{i,j}$$

.

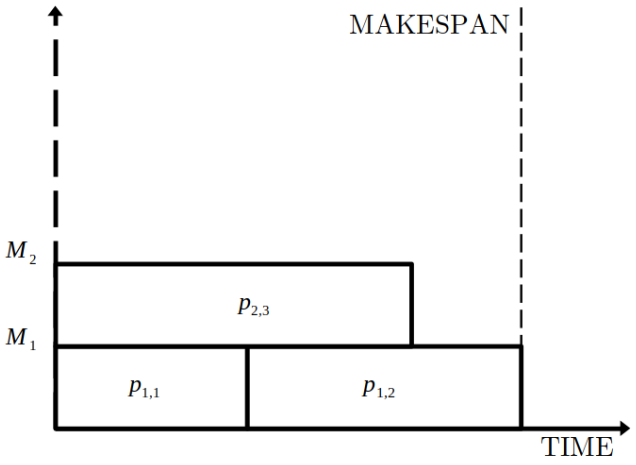

Figure 7.1: Simple diagram illustrating the PARALLEL MACHINE SCHEDULING problem. The x-axis represents time whilst the y-axis is associated with machines. 3 jobs are assigned to 2 machines M_1 and M_2. Each combination of job J_j and machine M_i is assigned a cost $p_{i,j}$. The makespan is additionally displayed as a dotted line.

7.2 Parallel Machine Scheduling as an Integer Program

It is easy to model the PARALLEL MACHINE SCHEDULING problem as an integer program. This can be achieved as follows.

$$\min \ t$$

subject to:

$$\sum_{J_j \in \mathcal{J}} p_{i,j} x_{i,j} \leq t \quad \forall M_i \in \mathcal{M}$$

and:

$$x_{i,j} \in \{0,1\} \quad \forall M_i \in \mathcal{M} \wedge J_j \in \mathcal{J}$$

.

Here, $x_{i,j}$ is an indicator variable such that:

$$x_{i,j} = \begin{cases} 1 \text{ if } \sigma(J_j) = M_i \\ 0 \text{ otherwise} \end{cases}$$

The first set of constraints ensures that each machine has a processing time of at most t. The integer program above is straightforward. However, there is a caveat. The solution of the above integer program would be $x_{i,j} = 0 \quad \forall i, j$. Thus, additional constraints are required. Specifically, the following constraints are added:

$$\sum_{M_i \in \mathcal{M}} x_{i,j} = 1 \quad \forall J_j \in \mathbf{J} \tag{7.1}$$

ensuring that each job is assigned to one of the machines.

7.3 Parallel Machine Scheduling as a Linear Program

To relax the integer program and obtain an associated linear program, we need to relax $x_{i,j} \in \{0,1\}$ and require instead that $x_{i,j} \geq 0$, which in turn is equivalent to $0 \leq x_{i,j} \leq 1$, as values larger than 1 are redundant.

To understand whether the solution of such relaxed LP is close to the optimal solution, one needs to study the *integrality gap* δ.

It is easy to show with a counterexample that the integrality gap is not bounded by any small constant, rather it grows with the input size.

Example 1 *Assume we have one job J_1 and m machines with constant cost $p_{i,j} = m$, that is, m is the processing time any machine takes to process the only job. The LP can assign $\frac{1}{m}$ of the job to each machine which results in $OPT_{LP} = 1$, while for the integer program $OPT_{ILP} = m$. Hence, $\delta \geq m$ so that the integrality gap grows with m.*

Despite the unbounded integrality gap, this simple linear program is a starting point for a more refined one.

The reason why δ is unbounded is that the linear programming relaxation has an *unfair* advantage. To see why, consider the following simple observation.

Observation 1

$$p_{i,j} > t \rightarrow x_{i,j} = 0.$$

The above implication simply follows because if $p_{i,j} > t$, $x_{i,j}$ cannot be 1 otherwise the cost would be larger than t. Thus, the integer program sets to 0 all those variables for which the processing cost exceeds t. On the other hand, the linear programming relaxation is allowed to set these variables to fractional positive values, thus achieving significantly lower cost. We can avoid the scenario in the above *bad* example (Example 1) by making the LP unable to schedule jobs longer than the total makespan. One might be tempted to add a set of constraints encoding Observation 1. However those are not linear constraints. Nonetheless, while we cannot encode such constraints in a single linear program, we can use a family of linear programs, indexed by a parameter. This is the core idea underlying the technique known as *parametric pruning*. Therefore, we introduce a parameter T providing a lower bound on OLP_{ILP}. At the moment, we assume that T is given but, as we shall see, the parameter can be found in polynomial time by binary search in practice. The set of feasible assignments for T is:

$$S_T = \{(i,j) : J_j \in \mathcal{J}, M_i \in \mathcal{M}, p_{i,j} \leq T\}.$$

This is the set of pairs to be considered for the linear program. Therefore, a *pruned* linear program is obtained, which is written as follows.

$$\min\ 0$$

subject to:

C_1

$$\sum_{i:(i,j)\in S_T} x_{i,j} = 1 \quad \forall J_j \in \mathcal{J}$$

$\mathbf{C_2}$

$$\sum_{j:(i,j)\in S_T} p_{i,j}x_{i,j} \leq T \quad \forall M_i \in \mathcal{M}$$

$\mathbf{C_3}$

$$x_{i,j} \geq 0 \quad \forall (i,j) \in S_T$$

This linear program is denoted by $LP(T)$. Note that min 0 simply means that there is no objective function, that is, we have a *feasibility* linear program. The goal is to find the smallest value of T, say T^*, for which the linear program $LP(T)$ is feasible. As already mentioned, binary search allows to do this in polynomial time since $p_{i,j} \in \mathbb{N}_{>0}$. Finally, note that in the situation of Example 1 previously considered, this linear program would not have a feasibile solution for $T < m$.

7.3.1 Properties of Extreme Point Solutions

Here, the pruned linear program introduced above is characterized via a number of properties that define its structure, and in particular, the structure of the corresponding extreme point solutions. This characterization will be used to prove the existence of a 2-approximation algorithm for the problem under consideration. The algorithm rounds an extreme point solution of $LP(T^*)$ to find a schedule having makespan not larger than $2T^*$. The rest of the notes is dedicated to the description of extreme point solutions of the linear program and the algorithm which provides the approximation guarantees.

Notice that, concerning T^*, the following simple inequality holds.

Observation 2

$$T^* \leq OPT_{ILP}$$

because T as mentioned above provides a lower bound on PT_{ILP}, so that the same holds for $T = T^*$.

There is an important structural lemma suggesting that the support of the linear program is remarkably sparse, which usually leads to easier rounding.

Lemma 1 *Any extreme point solution of $LP(T)$ has at most $|\mathcal{J}| + |\mathcal{M}|$ positive (or nonzero) variables.*

Proof. First, note that the space identified by $LP(T)$ has dimension $|S_T|$. A feasible solution to $LP(T)$ is an extreme point solution iff it corresponds to setting $|S_T|$ linearly independent constraints of $LP(T)$ to equality. Therefore, we need to intersect $|S_T|$ inequalities. Among S_T variables, $|\mathcal{J}|$ are fixed by $\mathbf{C_1}$, and similarly $|M|$ variables are fixed by $\mathbf{C_2}$. Thus, $|S_T| - |\mathcal{J}| - |\mathcal{M}|$ is the lower bound on the number of $\mathbf{C_3}$ constraint that must be tight (i.e., the amount of variables that should be precisely 0), whereas $|\mathcal{J}| + |\mathcal{M}|$ is an upper bound on the number of variables that may not be tight. This concludes the proof. ■

Let $\mathbf{x}$ be an extreme point solution. We refer to a job j as integrally set in $\mathbf{x}$ if it is entirely scheduled on one machine. Otherwise, job j is said to be fractionally set.

it is possible to prove another important structural lemma, which bounds the number of variables that can have fractional values. Of course, all other variables that have not fractional values, clearly have integral value.

Lemma 2 *Any extreme point solution $\mathbf{x}$ for $LP(T)$ fractionally sets at most $|\mathcal{M}|$ jobs. Similarly any extreme point solution for $LP(T)$ sets at least $|\mathcal{J}| - |\mathcal{M}|$ jobs integrally.*

Proof. Let α be the number of jobs $\mathcal{J}_\alpha$ that the extreme point solution sets integrally (namely to 1) and let β be the number of jobs $\mathcal{J}_\beta$ that the extreme point solution sets fractionally. Clearly:

$$\alpha + \beta = |\mathcal{J}|.$$

If a job J_j is set integrally, it means that there exists a machine M_i such that $x_{i,j} = 1$ whilst all other machines are set to 0 for the same job, $x_{h,j} = 0$ for $h \neq i$. For the other jobs set fractionally, the jobs can be split into multiple machines, but the machine-specific weights have to sum to 1. Hence, at least two variables have to be set fractionally. For any job in $\mathcal{J}_\alpha$ exactly one job is set to 1. On the other hand, for any job in $\mathcal{J}_\beta$, at least two jobs are set to be non-zero. As a consequence the number of jobs $\mathcal{J}^+$ of positive value is bounded by $\alpha + 2\beta$. By Lemma 1, we have:

$$|\mathcal{J}| + |\mathcal{M}| \geq |\mathcal{J}^+| \geq \alpha + 2\beta$$

and since $\alpha + 2\beta = |\mathcal{J}| + \beta$ it follows: $\beta \leq |\mathcal{M}|$ and $\alpha \geq |\mathcal{J}| - |\mathcal{M}|$, which concludes the proof. ■

7.3.2 Rounding Algorithm

To design of the rounding algorithm, we rely on a graph structure, as well as on the notion of perfect matching. The algorithm is illustrated next.

- Let $G = (\mathcal{J} \cup \mathcal{M}, E)$ be a bipartite graph where $(i,j) \in E \iff x_{i,j} > 0$.
- Given G, let $F \subseteq \mathcal{J}$ be the set of jobs assigned fractionally. Define:

  $$H = G[F \cup \mathcal{M}].$$

  In this restriction of the original graph, by construction, $(i,j) \in E[H] \iff 0 < x_{i,j} < 1$.
- Find a perfect matching in H. However, note that it is necessary to prove that such a matching always exist. The proof is given in Section 7.3.2.1. For the time being, assume that such matching exists.
- Finally, given the extreme point solution vector of $LP(T)$, assign integral jobs as indicated by variables $x_{i,j}$ and assign fractional ones as indicated by the matching P.

The rounding algorithm comes with approximation guarantees, as formalized by the following crucial result.

Theorem 1 *The rounding algorithm is a* 2*-approximation algorithm.*

Proof. Recall that $T^* \leq OPT$ since $LP(OPT)$ has a feasible solution. For jobs assigned integrally, by $\mathbf{C_2}$, every machine has at most completion time T^*. As for fractional jobs, every machine only gets at most 1 extra job when assignment is carried out based on matching P. Furthermore, $\forall (i,j) \in S_{T^*}, p_{i,j} \leq T^*$. This implies that if we assign J_j to M_i completion time of M_i increases by at most T^*. Then, after the rounding (or assigning of the fractional jobs) we have that the total makespan is bounded by $2T^* = 2OPT$, proving the approximation factor claimed in the theorem. ■

7.3.2.1 Perfect Matching

So far, we have assumed the existence of the perfect matching. Again, the structure of the extreme point solution of the linear program is exploited to prove that such matching is guaranteed to exist.

Preliminarily, however, it is necessary to introduce the notions of *pseudo-tree* and *pseudo-forest*.

Definition 1 (Pseudo-tree) *A connected graph is called a pseudo-tree if it is either a tree or a tree with an additional edge. More formally, it is a connected graph $G = (V, E)$ with $|E| \leq |V|$.*

A pseudo-forest, as the name suggest, is a collection of pseudo-trees.

Definition 2 (Pseudo-forest) *A graph is called a pseudo-forest if its connected components are pseudo-trees.*

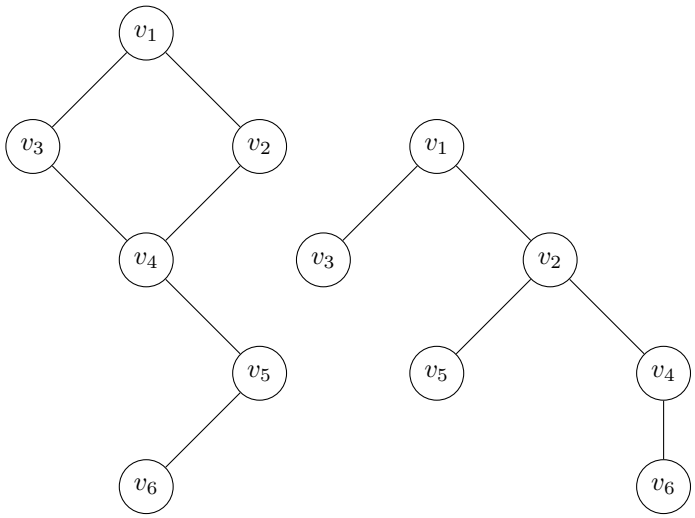

Figure 7.2: An example of pseudo-forest composed of two trees. Note that a pseudo-tree can be a tree (right pseudo-tree) or a connected graph composed of a cycle and one or more trees (left pseudo-tree).

It is possible to show that G is a pseudo-forest, as proved next.

Lemma 3 *$G = (\mathcal{J} \cup \mathcal{M}, E)$ is a pseudo-forest.*

Proof. We need to show that every connected component of G is a pseudo-tree. Observe that, by Lemma 1:

$$|E_G| \leq |V_G| = |\mathcal{J}| + |\mathcal{M}|.$$

This is not sufficient to complete the proof because we may have some components that are trees and some components where instead $|E| > |V|$. However, we can show that this is not possible. Given extreme point solution $\mathbf{x}$ and an arbitrary component C, define $\mathbf{x}_C$ to be the restriction of $\mathbf{x}$ to C. We claim that $\mathbf{x}_C$ is an extreme point solution of $LP(T^*)$ on C. To prove this by contradiction, assume it is not. It means that it would be possible to write $\mathbf{x}_C = \lambda y_C + (1-\lambda)y'_C$ for $0 < \lambda < 1$ since $\mathbf{x}$ would be a convex combination of two feasible solutions for the restricted linear program. Let $\mathbf{x}_{\bar{C}} = \mathbf{x} \setminus \mathbf{x}_C$ be the set of coordinates of the vector $\mathbf{x}$ that are not in $\mathbf{x}_C$. Therefore we can write:

$$\mathbf{x} = \lambda'(\mathbf{x}_{\bar{C}} \odot y_C) + (1-\lambda')(\mathbf{x}_{\bar{C}} \odot y'_C)$$

which means that $\mathbf{x}$ is a convex combination of two feasible solutions to $LP(T)$, which is a contradiction because $\mathbf{x}$ is an extreme point solution. Thus, each component must be a pseudo-tree and we have proved the statement that G is a pseudo-forest. ■

Based on Lemma 3, we can prove the following crucial result, which guarantees that a perfect matching in H exists.

Lemma 4 *Given $G = (\mathcal{J} \cup \mathcal{M}, E)$ defined as above, any $H = G[F \cup \mathcal{M}]$ has a perfect matching.*

Proof. A simple intuitive proof is as follows. Every component is either a tree or a composition of a unique cycle and trees. The degree of the nodes suggests that the leaves are machines. Each job that is integrally set in $\mathbf{x}$ has exactly one edge incident at it in G. Moreover, as already noticed, all jobs that are assigned fractionally will have at least 2 positive values. Hence, all such jobs in H have degree at least 2: $deg_c(J_j) \geq 2 \ \forall J_j$. In light of this consideration, we devise the following procedure which retrieves a matching in H. Given a machine and the job it is connected to, we match them and remove the job and the connected machines. This can be repeated iteratively, until the three becomes a simple even cycle, for which finding a perfect matching is trivial. ■

7.3.2.2 Binary Search

Finally, we briefly describe how binary search is performed to find the value of T^*. This is the last ingredient that we need to complete the discussion of the approximation algorithm for the Parallel Machine Scheduling problem. A *naïve* (or greedy) scheduling is obtained by assigning any job J_j to the machine minimizing $p_{i,j}$. It is easy to see that this is a feasible assignment. Using the *naïve* scheduling, we get an associated makespan τ which is an upper bound on OPT. Moreover, as it is possible to prove, it holds that $OPT \geq \frac{\tau}{|\mathcal{M}|}$. As a consequence, binary search is performed in the interval $[\frac{\tau}{|\mathcal{M}|}, \tau]$.