CS 598CSC: Approximation Algorithms	Lecture date: February 23 and 25, 2011
Instructor: Alina Ene	Scribe: Alina Ene

1 Scheduling on Unrelated Parallel Machines

We have a set J of n jobs, and a set M of m machines. The processing time of job i is p_{ij} on machine j. Let $f : J \leftarrow M$ be a function that assigns each job to exactly one machine. The *makespan* of f is $\max_{1 \leq j \leq m} \sum_{i:f(i)=j} p_{ij}$, where $\sum_{i:f(i)=j} p_{ij}$ is the total processing time of the jobs that are assigned to machine j. In the Scheduling on Unrelated Parallel Machines problem, the goal is to find an assignment of jobs to machines of minimum makespan.

We can write an LP for the problem that is very similar to the routing LP from the previous lecture. For each job i and each machine j, we have a variable x_{ij} that denotes whether job i is assigned to machine j. We also have a variable λ for the makespan. We have a constraint for each job that ensures that the job is assigned to some machine, and we have a constraint for each machine that ensures that the total processing time of jobs assigned to the machines is at most the makespan λ.

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

The above LP is very natural, but unfortunately it has unbounded integrality gap. Suppose that we have a single job that has processing time T on each of the machines. Clearly, the optimal schedule has makespan T. However, the LP can schedule the job to the extend of $1/m$ on each of the machines, i.e., it can set $x_{1j} = 1/m$ for all j, and the makespan of the resulting fractional schedule is only T/m.

To overcome this difficulty, we modify the LP slightly. Suppose we knew that the makespan of the optimal solution is equal to λ, where λ is some fixed number. If the processing time p_{ij} of job i on machine j is greater than λ, job i is *not* scheduled on machine j, and we can strengthen the LP by setting x_{ij} to 0 or equivalently, by removing the variable. More precisely, let $\mathcal{S}_\lambda = \{(i, j) \mid i \in J, j \in M, p_{ij} \leq \lambda\}$. Given a value λ, we can write the following LP for the problem.

$$\begin{aligned}
& \underline{\mathbf{LP}(\lambda)} & \\
& \sum_{j:\,(i,j) \in \mathcal{S}_\lambda} x_{ij} = 1 & \forall i \in J \\
& \sum_{i:\,(i,j) \in \mathcal{S}_\lambda} x_{ij} p_{ij} \leq \lambda & \forall j \in M \\
& x_{ij} \geq 0 & \forall (i, j) \in \mathcal{S}_\lambda
\end{aligned}$$

Note that the LP above does *not* have an objective function. In the following, we are only interested in whether the LP is feasible, i.e, whether there is an assignment that satisfies all the constraints. Also, we can think of λ as a parameter and $\mathbf{LP}(\lambda)$ as a family of LPs, one for each value of the parameter. A useful observation is that, if λ is a lower bound on the makespan of the optimal schedule, $\mathbf{LP}(\lambda)$ is feasible and it is a valid relaxation for the SCHEDULING ON UNRELATED PARALLEL MACHINES problem.

Lemma 1. *Let λ^* be the minimum value of the parameter λ such that $\mathbf{LP}(\lambda)$ is feasible. We can find λ^* in polynomial time.*

Proof: For any fixed value of λ, we can check whether $\mathbf{LP}(\lambda)$ is feasible using a polynomial-time algorithm for solving LPs. Thus we can find λ^* using binary search starting with the interval $[0, \sum_{i,j} p_{ij}]$. □

In the following, we will show how to round a solution to $\mathbf{LP}(\lambda^*)$ in order to get a schedule with makespan at most $2\lambda^*$. As we will see shortly, it will help to round a solution to $\mathbf{LP}(\lambda^*)$ that is a *vertex* solution.

Let x be a vertex solution to $\mathbf{LP}(\lambda^*)$. Let G be a bipartite graph on the vertex set $J \cup M$ that has an edge ij for each variable $x_{ij} \neq 0$. We say that job i is fractionally set if $x_{ij} \in (0, 1)$ for some j. Let F be the set of all jobs that are fractionally set, and let H be a bipartite graph on the vertex set $F \cup M$ that has an edge ij for each variable $x_{ij} \in (0, 1)$; note that H is the induced subgraph of G on $F \cup M$. As shown in Lemma 2, the graph H has a matching that matches every job in F to a machine, and we will use such a matching in the rounding algorithm.

Lemma 2. *The graph G has a matching that matches every job in F to a machine.*

We are now ready to give the rounding algorithm.

SUPM-Rounding
- Find λ^*
- Find a vertex solution x to $\mathbf{LP}(\lambda^*)$
- For each i and j such that $x_{ij} = 1$, assign job i to machine j
- Construct the graph H
- Find a maximum matching $\mathcal{M}$ in H
- Assign the fractionally set jobs according to the matching $\mathcal{M}$

Theorem 3. *Consider the assignment constructed by* SUPM-*Rounding. Each job is assigned to a machine, and the makespan of the schedule is at most $2\lambda^*$.*

Proof: By Lemma 2, the matching $\mathcal{M}$ matches every fractionally set job to a machine and therefore all of the jobs are assigned. After assigning all of the integrally set jobs, the makespan (of the partial schedule) is at most λ^*. Since $\mathcal{M}$ is a matching, each machine receives at most one additional job. Let i be a fractionally set job, and suppose that i is matched (in $\mathcal{M}$) to machine j. Since the pair (i, j) is in $\mathcal{S}_{\lambda^*}$, the processing time p_{ij} is at most λ^*, and therefore the total processing time of machine j increases by at most λ after assigning the fractionally set jobs. Therefore the makespan of the final schedule is at most $2\lambda^*$. □

Exercise: Give an example that shows that Theorem 3 is tight. That is, give an instance and a vertex solution such that the makespan of the schedule SUPM-Rounding is at least $(2 - o(1))\lambda^*$.

Since λ^* is a lower bound on the makespan of the optimal schedule, we get the following corollary.

Corollary 4. SUPM-*Rounding achieves a 2-approximation.*

Now we turn our attention to Lemma 2 and some other properties of vertex solutions to $\mathbf{LP}(\lambda)$.

Lemma 5. *If $\mathbf{LP}(\lambda)$ is feasible, any vertex solution has at most $m + n$ non-zero variables and it sets at least $n - m$ of the jobs integrally.*

Proof: Let x be a vertex solution to $\mathbf{LP}(\lambda)$. Let r denote the number of pairs in $\mathcal{S}_\lambda$. Note that $\mathbf{LP}(\lambda)$ has r variables, one for each pair $(i, j) \in \mathcal{S}_\lambda$. If x is a vertex solution, it satisfies r of the constraints of $\mathbf{LP}(\lambda)$ with equality. The first set of constraints consists of m constraints, and the second set of constraints consists of n constraints. Therefore at least $r - (m + n)$ of the tight constraints are from the third set of constraints, i.e., at least $r - (m + n)$ of the variables are set to zero.

We say that job i is set fractionally if $x_{ij} \in (0, 1)$ for some j; job i is set integrally if $x_{ij} \in \{0, 1\}$ for all j. Let I and F be the set of jobs that are set integrally and fractionally (respectively). Clearly, $|I| + |F| = n$. Any job i that is fractionally set is assigned (fractionally) to at least two machines, i.e., there exist $j \neq \ell$ such that $x_{ij} \in (0, 1)$ and $x_{i\ell} \in (0, 1)$. Therefore there are at least $2|F|$ distinct non-zero variables corresponding to jobs that are fractionally set. Additionally, for each job i that is integrally set, there is a variable x_{ij} that is non-zero. Thus the number of non-zero variables is at least $|I| + 2|F|$. Hence $|I| + |F| = n$ and $|I| + 2|F| \leq m + n$, which give us that $|I|$ is at least $n - m$. □

Definition 1. *A connected graph is a **pseudo-tree** if the number of edges it most the number of vertices plus one. A graph is a **pseudo-forest** if each of its connected components is a pseudo-tree.*

Lemma 6. *The graph G is a pseudo-forest.*

Proof: Let C be a connected component of G. We restrict $\mathbf{LP}(\lambda)$ and x to the jobs and machines in C to get $\mathbf{LP}'(\lambda)$ and x'. Note that x' is a feasible solution to $\mathbf{LP}'(\lambda)$. Additionally, x' is a vertex solution to $\mathbf{LP}'(\lambda)$. If not, x' is a convex combination of two feasible solutions x'_1 and x'_2 to $\mathbf{LP}'(\lambda)$. We can extend x'_1 and x'_2 to two solutions x_1 and x_2 to $\mathbf{LP}(\lambda)$ using the entries of x that are not in x'. By construction, x_1 and x_2 are feasible solutions to $\mathbf{LP}(\lambda)$. Additionally, x is a convex combination of x_1 and x_2, which contradicts the fact that x is a vertex solution. Thus x' is a vertex solution to $\mathbf{LP}'(\lambda)$ and, by Lemma 5, x' has at most $n' + m'$ non-zero variables, where n' and m' are the number of jobs and machines in C. Thus C has $n' + m'$ vertices and at most $n' + m'$ edges, and therefore it is a pseudo-tree. □

Proof of Lemma 2: Note that each job that is integrally set has degree one in G. We remove each integrally set job from G; note that the resulting graph is H. Since we removed an equal number of vertices and edges from G, it follows that H is a pseudo-forest as well. Now we construct a matching $\mathcal{M}$ as follows.

Note that every job vertex has degree at least 2, since the job is fractionally assigned to at least two machines. Thus all of the leaves (degree-one vertices) of H are machines. While H has at least one leaf, we add the edge incident to the leaf to the matching and we remove both of its endpoints from the graph. If H does not have any leaves, H is a collection of vertex-disjoint cycles, since it is a pseudo-forest. Moreover, each cycle has even length, since H is bipartite. We construct a perfect matching for each cycle (by taking alternate edges), and we add it to our matching. □

Exercise: (Exercise 17.1 in [3]) Give a proof of Lemma 2 using Hall's theorem.

2 Generalized Assignment Problem

The Generalized Assignment problem is a generalization of the Scheduling on Unrelated Parallel Machines problem in which there are costs associated with each job-machine pair, in addition to a processing time. More precisely, we have a set J of n jobs, a set M of m machines, and a target λ. The processing time of job i is p_{ij} on machine j, and the cost of assigning job i to machine j is c_{ij}. Let $f : J \to M$ be a function that assigns each job to exactly one machine. The assignment f is *feasible* if its makespan is at most λ (recall that λ is part of the input), and its cost is $\sum_i c_{if(i)}$. In the Generalized Assignment problem, the goal is to construct a minimum cost assignment f that is *feasible*, provided that there is a feasible assignment.

In the following, we will show that, if there is a schedule of cost C and makespan at most λ, then we can construct a schedule of cost at most C and makespan at most 2λ.

As before, we let $\mathcal{S}_\lambda$ denote the set of all pairs (i, j) such that $p_{ij} \leq \lambda$. We can generalize the relaxation $\mathbf{LP}(\lambda)$ from the previous section to the following LP.

GAP-LP

$$
\begin{aligned}
\text{min} \quad & \sum_{(i,j)\in\mathcal{S}_\lambda} x_{ij} c_{ij} && \\
\text{subject to} \quad & \sum_{j:\,(i,j)\in\mathcal{S}_\lambda} x_{ij} = 1 && \forall i \in J \\
& \sum_{i:\,(i,j)\in\mathcal{S}_\lambda} x_{ij} p_{ij} \leq \lambda && \forall j \in M \\
& x_{ij} \geq 0 && \forall (i,j) \in \mathcal{S}_\lambda
\end{aligned}
$$

Since we also need to preserve the costs, we can no longer use the previous rounding; in fact, it is easy to see that the previous rounding is arbitrarily bad for the Generalized Assignment problem. However, we will still look for a matching, but in a slightly different graph.

But before we give the rounding algorithm for the Generalized Assignment problem, we take a small detour into the problem of finding a minimum-cost matching in a bipartite graph. In the Minimum Cost Biparite Matching problem, we are given a bipartite graph $B = (V_1 \cup V_2, E)$ with costs c_e on the edges, and we want to construct a minimum cost matching $\mathcal{M}$ that matches every vertex in V_1, if there is such a matching. For each vertex v, let $\delta(v)$ be the set of all edges incident to v. We can write the following LP for the problem.

BipartiteMatching(B)

$$\begin{aligned} \min \quad & \sum_{e \in E(B)} c_e y_e & \\ \text{subject to} \quad & \sum_{e \in \delta(v)} y_e = 1 & \forall v \in V_1 \\ & \sum_{e \in \delta(v)} y_e \leq 1 & \forall v \in V_2 \\ & y_e \geq 0 & \forall e \in E(B) \end{aligned}$$

Theorem 7 (Edmonds [2]). *For any bipartite graph B, any vertex solution to **BipartiteMatching**(B) is an integer solution. Moreover, given a feasible fractional solution, we can find in polynomial time a feasible solution z such that z is integral and*

$$\sum_{e \in E(B)} c_e z_e \leq \sum_{e \in E(B)} c_e y_e.$$

Let x be an optimal vertex solution to **GAP-LP**. As before, we want to construct a graph G that has a matching $\mathcal{M}$ that matches all jobs. The graph G will now have costs on its edges and we want a matching of cost at most C. Recall that for Scheduling on Unrelated Parallel Machines we defined a bipartite graph on the vertex set $J \cup M$ that has an edge ij for every variable x_{ij} that is non-zero. We can construct the same graph for Generalized Assignment, and we can assign a cost c_{ij} to each edge ij. If the solution x was actually a fractional matching — that is, if x was a feasible solution to **BipartiteMatching**(G) — Theorem 7 would give us the desired matching. The solution x satisfies the constraints corresponding to vertices $v \in J$, but it does not necessarily satisfy the constraints corresponding vertices $v \in M$, since a machine can be assigned more than one job. To get around this difficulty, we will introduce several nodes representing the same machine, and we will use x to construct a fractional matching for the resulting graph.

The fractional solution x assigns $\sum_{i \in J} x_{ij}$ jobs to machine j; let $k_j = \lceil \sum_{i \in J} x_{ij} \rceil$. We construct a bipartite graph G as follows. For each job i, we have a node i. For each machine j, we have k_j nodes $(j,1), \cdots, (j,k_j)$. We can think of the nodes $(j,1), \cdots, (j,k_j)$ as *slots* on machine j. Since now we have multiple slots on each of the machines, we need a fractional assignment y that assigns a job to slots on the machines. More precisely, y has an entry $y_{i,(j,s)}$ for each job i and each slot (j,s) that represents the fraction of job i that is assigned to the slot. We give the algorithm that constructs y from x below. Once we have the solution y, we add an edge between any job i and any machine slot (j,s) such that $y_{i,(j,s)}$ is non-zero. Additionally, we assign a cost $c_{i,(j,s)}$ to each edge $(i,(j,s))$ of G that is equal to c_{ij}.

When we construct y, we consider each machine in turn. Let j be the current machine. Recall that we want to ensure that y assigns at most one job to each slot; as such, we will think of each slot on machine j as a bin with capacity 1. We "pack" jobs into the bins greedily. We only consider jobs i such that p_{ij} is at most λ; let h denote the number of such jobs. We assume without loss of generality that these are labeled as $1, 2, \cdots, h$, and $p_{1j} \geq p_{2j} \geq \cdots \geq p_{hj}$. Informally, when we construct y, we consider the jobs $1, 2, \cdots, h$ in this order. Additionally, we keep track of the bin that has not been filled and the amount of space s available on that bin. When we consider job i,

```
GreedyPacking(x)
  y = 0 ⟨⟨initialize y to 0⟩⟩
  s = 1 ⟨⟨s is the current bin⟩⟩
  R = 1 ⟨⟨R is the space available on bin s⟩⟩
  for i = 1 to h
  ⟨⟨pack x_ij into the bins⟩⟩
  if x_ij ≤ R
       y_{i,(j,s)} = x_ij
       R = R − x_ij
       if R = 0
            s = s + 1
            R = 1
  else
       y_{i,(j,s)} = R
       y_{i,(j,s+1)} = x_ij − R ⟨⟨pack x_ij − R in the next bin⟩⟩
       R = 1 − y_{i,(j,s+1)}
       s = s + 1
  return y
```

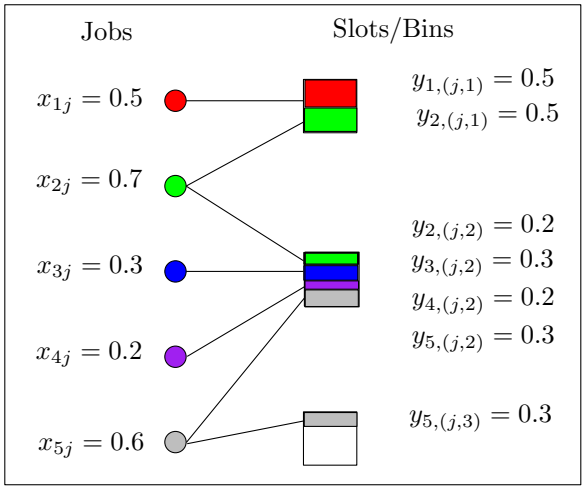

Figure 1. Constructing y from x.

we try to pack x_{ij} into the current bin: if there is at least x_{ij} space available, i.e., $x_{ij} \leq s$, we pack the entire amount into the current bin; otherwise, we pack as much as we can into the current bin, and we pack the rest into the next bin. (See Figure 1 for an example.)

Lemma 8. *The solution y constructed by* GreedyPacking *is a feasible solution to* BipartiteMatching(G). *Moreover,*

$$\sum_{(i,(j,s))\in E(G)} y_{i,(j,s)} c_{i,(j,s)} = \sum_{(i,j)\in\mathcal{S}_\lambda} x_{ij} c_{ij}.$$

Proof: Note that, by construction, $x_{ij} = \sum_{s=1}^{k_j} y_{i,(j,s)}$. Therefore, for any job i, we have

$$\sum_{(i,(j,s))\in\delta(i)} y_{i,(j,s)} = \sum_{j:\ (i,j)\in\mathcal{S}_\lambda} \sum_{s=1}^{k_j} y_{i,(j,s)} = \sum_{j:\ (i,j)\in\mathcal{S}_\lambda} x_{ij} = 1$$

Additionally, since we imposed a capacity of 1 on the bins associated with each slot, it follows that, for any slot (j,s),

$$\sum_{(i,(j,s)\in\delta((j,s)))} y_{i,(j,s)} \leq 1$$

Therefore y is a feasible solution to BipartiteMatching(G). Finally,

$$\sum_{(i,(j,s))\in E(G)} = \sum_{i=1}^{n} \sum_{j:\ (i,j)\in\mathcal{S}_\lambda} \sum_{s=1}^{k_j} y_{i,(j,s)} c_{ij} = \sum_{(i,j)\in\mathcal{S}_\lambda} x_{ij} c_{ij}$$

□

Theorem 7 gives us the following corollary.

Corollary 9. *The graph G has a matching $\mathcal{M}$ that matches every job and it has cost at most $\sum_{(i,j)\in\mathcal{S}_\lambda} x_{ij}c_{ij}$. Moreover, we can find such a matching in polynomial time.*

GAP-Rounding
let x be an optimal solution to **GAP-LP**
$y = \text{GREEDYPACKING}(x)$
construct the graph G
construct a matching $\mathcal{M}$ in G such that $\mathcal{M}$ matches every job
    and the cost of $\mathcal{M}$ is at most $\sum_{(i,j)\in\mathcal{S}_\lambda} x_{ij}c_{ij}$
for each edge $(i,(j,s)) \in \mathcal{M}$
    assign job i to machine j

Theorem 10. *Let $C = \sum_{(i,j)\in\mathcal{S}_\lambda} x_{ij}c_{ij}$. The schedule returned by **GAP-Rounding** has cost at most C and makespan at most 2λ.*

Proof: By Corollary 9, the cost of the schedule is at most C. Therefore we only need to upper bound the makespan of the schedule.

Consider a machine j. For any slot (j,s) on machine j, let

$$q_{js} = \max_{i:y_{i,(j,s)}>0} p_{ij}$$

That is, q_{js} is the maximum processing time of any pair ij such that job i is assigned (in y) to the slot (j,s). It follows that the total processing time of the jobs that $\mathcal{M}$ assigns to machine j is at most $\sum_{s=1}^{k_j} q_{js}$.

Since **GAP-LP** has a variable x_{ij} only for pairs (i,j) such that p_{ij} is at most λ, it follows that q_{j1} is at most λ. Therefore we only need to show that $\sum_{s=2}^{k_j} q_{js}$ is at most λ as well. Consider a slot s on machine j such that $s > 1$. Recall that we labeled the jobs that are relevant to machine j — that is, jobs i such that p_{ij} is at most λ — as $1, 2, \cdots, h$ such that $p_{1j} \geq p_{2j} \geq \cdots \geq p_{hj}$. Consider a job ℓ that is assigned to slot s. Since GREEDYPACKING considers jobs in non-increasing order according to their processing times, the processing time $p_{\ell j}$ of job ℓ is at most the processing time of any job assigned to the slot $s-1$. Therefore $p_{\ell j}$ is upper bounded by any convex combination of the processing times of the jobs that are assigned to the slot $s-1$. Since the slot $s-1$ is full, $\sum_i y_{i,(j,s-1)} = 1$ and thus $p_{\ell j}$ is at most $\sum_i y_{i,(j,s-1)}p_{ij}$. It follows that

$$\sum_{s=2}^{k_j} q_{js} \leq \sum_{s=2}^{k_j}\sum_i y_{i,(j,s-1)}p_{ij} \leq \sum_{s=1}^{k_j}\sum_i y_{i,(j,s)}p_{ij}$$

By construction, $\sum_s y_{i,(j,s)} = x_{ij}$, and therefore

$$\sum_{s=1}^{k_j}\sum_i y_{i,(j,s)}p_{ij} = \sum_i p_{ij}\sum_{s=1}^{s} y_{i,(j,s)} = \sum_i p_{ij}x_{ij}$$

Since x is a feasible solution to the **GAP-LP**,

$$\sum_{s=2}^{k_j} q_{js} \leq \sum_i p_{ij}x_{ij} \leq \lambda$$

which completes the proof. □

References

[1] Chandra Chekuri, *Approximation Algorithms Lecture Notes*, Lecture 12, http://www.cs.uiuc.edu/~chekuri/teaching/fall2006/lect12.pdf, 2006.

[2] Alexander Schrijver, *Combinatorial Optimization*, Chapter 17, Springer Verlag, 2003.

[3] Vijay Vazirani, *Approximation Algorithms*, Chapter 17, Springer Verlag, 2001.

[4] David Williamson and David Shmoys, *The Design of Approximation Algorithms*, Chapter 11, http://www.designofapproxalgs.com/, 2010.