MS&E-319: Approximation Algorithms
Instructor: Professor Amin Saberi (saberi@stanford.edu)

Lecture 1: Introduction

This week we discussed three important problems in approximation algorithms:

1 Scheduling Jobs on Identical Machines

Suppose we have m identical machines and n jobs. For each job i, we are given t_i, the time it takes to process it on one of the machines. Our goals is to assign the jobs to machines in a balanced way.

Let A_j be the set of jobs assigned to machine j. Define $T_j = \sum_{i \in A_j} t_i$ to be the *load* of machine j. The *makespan* of an assignment is the maximum load on a machine (i.e. $\max_i T_i$). The goal of *load balancing* is to find an assignment of jobs to machines that minimizes the makespan.

A "greedy" approach, Algorithm 2 is to iteratively assign each job to the machine with the smallest load.

Algorithm 1 Greedy

$\forall j$, $A_j \leftarrow \emptyset$, $T_j \leftarrow 0$
for $i = 1$ to n **do**
  $j \leftarrow \text{argmin}_k T_k$
  $A_j = A_j \cup \{i\}$
  $T_j = T_j + t_i$
end for

Theorem 1 (Graham, 1966) *Greedy scheduling is a* 2*-approximation for the minimum makespan problem.*

To prove this result, we need to find a few lower bounds for the *optimal solution* T^*.

Lemma 1 *The optimal makespan* $T^* \geq \max_i t_i$.

Proof: The most time consuming job must be assigned to some machine. ■

Lemma 2 *The optimal makespan* $T^* \geq \frac{1}{m} \sum_i^n t_i$.

Proof: The maximum load must be larger than the average load. ■

Lemma 3 *The solution of the greedy makespan algorithm is at most*

$$\frac{1}{m} \sum_i^n t_i + \max_i t_i$$

Proof: Consider machine j with maximum load T_j. Let i be the *last* job scheduled on machine j. When i was scheduled, j had the smallest load, so j must have had smaller than the average load. We have,

$$T_j = (T_j - t_j) + t_j \leq \frac{1}{m} \sum_i^n t_i + \max_i t_i \ .$$

Therefore the total load on machine j is less than the average load plus the largest job. ■

Combining these three lemmas implies Theorem 1. Is this analysis tight? The following example shows that it essentially is.

Example: Consider m machines, with $m(m-1)$ jobs of length 1 and one job of length m. The optimal solution is to assign the largest jobs to one machine, and m of the small jobs to each of the remaining $m-1$ machines, resulting in an optimal makespan of m. The greedy algorithm may assign the largest job last, at which point each machine has load $m-1$, making a makespan of $2m-1$.

Now, consider a slightly modified greedy approach. First, we sort the jobs so that $t_1 \geq t_2 \geq \cdots \geq t_n$. Then, assign them iteratively to the machines, using Algorithm 2.

Algorithm 2 Greedy after sorting

$\forall j$, $A_j \leftarrow \emptyset$, $T_j \leftarrow 0$
Sort the jobs so that $t_1 \geq t_2 \geq \cdots \geq t_n$
for $i = 1$ to n **do**
  $j \leftarrow \text{argmin}_k T_k$
  $A_j = A_j \cup \{i\}$
  $T_j = T_j + t_i$
end for

Lemma 4 *The approximation factor of the modified greedy algorithm is at most* $3/2$.

Proof: If there are at most m jobs, the scheduling is optimal since we put each job on its own machine. If there are more than m jobs by the pigeonhole principle, at least one processor must get 2 of the first $m+1$ jobs. Each of these jobs is at least as big as t_{m+1}. Thus, $T^* \geq 2t_{m+1}$.

Consider machine j assigned maximum load T where $j > m$ since otherwise we are done. Let i be the last job assigned to j. Since the loads are sorted $t_i \leq t_{m+1} \leq T^*/2$ and as before

$$T = (T - t_i) + t_i \leq \frac{1}{m}\sum_{j=1}^{m} T_j + t_i \leq T^* + T^*/2 = \frac{3}{2}T^*$$

■

The above analysis is not tight. It is possible to show that the approximation factor of greedy after sorting is at most $4/3$. The proof of $4/3$ factor uses the same ideas as the above proof but it is more involved. We omit it in this lecture note but sketch the main idea: if the last job assigned to the machine with the highest load has index less than $2m$, then the algorithm has found the optimum solution. Otherwise, $T \leq \frac{1}{m}\sum_{j=1}^{m} T_j + t_{2m+1} \leq T^* + T^*/3 = \frac{4}{3}T^*$.

Lemma 5 *The approximation factor of the modified greedy algorithm is* $4/3$.

Note that $4/3$ is essentially tight. Consider an instance with m machines, $n = 2m+1$ jobs, $2m$ jobs of length $m+1, m+2, \cdots, 2m-1$ and one job of length m.

2 Maximum Satisfiability

We return to the setting of boolean formulas and consider a problem related to satisfiability: for a given formula in Conjunctive Normal Form (CNF), what is the maximum number of its clauses that can be satisfied

by assigning true and false value to variables? More concretely, suppose we have n variables $x_1, ..., x_n$ and m clauses $C_1, ..., C_m$ where

$$C_i = \left(\bigvee_{i \in S_i^+} x_i \right) \vee \left(\bigvee_{i \in S_i^-} \bar{x}_i \right).$$

The **maximum satisfiability problem** is to find the maximum number of clauses that may be satisfied by an assignment x.

We first propose a simple randomized algorithm to approximate a solution to this problem. Set each x_i independently to be 0 or 1 with probability 1/2. The probability that C_i is satisfied by this assignment is $1 - 2^{|C_i|}$ for every i. If we let Z_i denote the event that clause C_i is satisfied by this random assignment and $Z = \sum_{i=1}^m Z_i$ be the total number of satisfied clauses, we may compute:

$$\mathrm{E}[Z] = \sum_{i=1}^m \mathrm{E}[Z_i] = \sum_{i=1}^m \left(1 - 2^{|C_i|}\right).$$

In the case that all of our clauses are large, i.e. $|C_i| \geq K$ for each i, then this randomized algorithm has an approximation ratio of $\geq 1 - 2^{-K}$ in expectation:

$$m(1 - 2^{-K}) \leq \mathrm{E}[Z] \leq OPT \leq m.$$

Having a bound on the approximation ratio of the algorithm in expectation is often unsatisfactory. This is because a bound on expected value does not bound the probability that the algorithm returns a good solution. Concentration inequalities can help us estimate such probabilities, but in some cases we may do even better. This algorithm may be **derandomized** using conditional expectation as follows.

Algorithm 3 Derandomized Approximation Algorithm for Maximum Satisfiability

for $i = 1$ to n **do**
  Compute $\frac{1}{2}\mathrm{E}[Z \mid x_i = 1, x_{i-1}, ..., x_1]$ and $\frac{1}{2}\mathrm{E}[Z \mid x_i = 0, x_{i-1}, ..., x_1]$.
  Set $x_i = 1$ if the first expression is larger than the second, set $x_i = 0$ otherwise.
end for
return x

For motivation, we consider the first step of the algorithm. Note that

$$\mathrm{E}[Z] = \frac{1}{2}\mathrm{E}[Z \mid x_1 = 1] + \frac{1}{2}\mathrm{E}[Z \mid x_1 = 0].$$

Both terms on the right hand side may be computed simply by summing over all clauses the probability that C_i is satisfied given the information on x_1. The equation above implies that $\mathrm{E}[Z \mid x_1 = 1] \geq \mathrm{E}[Z]$ or $\mathrm{E}[Z \mid x_1 = 0] \geq \mathrm{E}[Z]$. Thus if we choose the greater expectation in each step of the algorithm, we will deterministically build up an assignment x such that

$$\mathrm{E}[Z|x] \geq \mathrm{E}[Z] \geq m(1 - 2^{-K})$$

where $\mathrm{E}[Z|x]$ is the number of clauses x satisfies.

The approximation ratio of algorithm 3 is good only if the clauses have a large number of variables. We present a different algorithm for dealing with the possibility that some of the clauses may be small. It is based on the now familiar concept of LP relaxation. We write down an integer program for maximum

satisfiability.

$$\begin{aligned} \textbf{maximize:} \quad & \sum_{i=1}^{m} q_i & \\ \textbf{s.t.} \quad & q_i \leq \sum_{j \in S_i^+} y_j + \sum_{j \in S_i^-} (1 - y_j) & \forall\, i \\ & q_i, y_j \in \{0, 1\} & \forall\, i, j \end{aligned}$$

The variables q_i correspond to the truth value of each clause C_i, and the variables y_j correspond to the values of each boolean variable x_i. We relax the last condition to be $0 \leq q_i, y_j \leq 1$ in order to get a linear program.

Algorithm 4 An LP Rounding Algorithm for Maximum Satisfiability

Solve the LP given above.
for $j = 1$ to n **do**
Independently set $x_j = \begin{cases} 1 & : \text{with probability } y_j^* \\ 0 & : \text{with probability } 1 - y_j^* \end{cases}$
end for

In order to analyze algorithm 4 we consider the probability that a particular clause is satisfied; Let u focus on one clause, say C_1 and assume without loss of generality that $C_1 = x_1 \vee ... \vee x_k$. We have $q_1^* = \min\{y_1^* + ... + y_k^*, 1\}$ and by the inequality of arithmetic and geometric means:

$$\begin{aligned} Pr[C_1] &= 1 - \prod_{j=1}^{k} (1 - y_j^*) \\ &\geq 1 - \left(\frac{1}{k} \sum_{j=1}^{k} (1 - y_j^*) \right)^k \\ &\geq 1 - \left(1 - \frac{q_1^*}{k} \right)^k \\ &\geq q_1 \left(1 - \left(1 - \frac{1}{k} \right)^k \right) \\ &\geq q_1 (1 - 1/e) \end{aligned}$$

This last line implies, through the linearity of expectation, that this rounding procedure gives a $(1 - 1/e)$-approximation for maximum satisfiability regardless of the size of the smallest clause. Algorithm 4 may also be derandomized by the method of conditional expectations.

These two derandomized algorithms may be combined to give a factor $3/4$ approximation algorithm for maximum satisfiability. We simply run both algorithms on a given problem instance and output the solution with the better value. This procedure itself may be viewed as a derandomization of an algorithm that flips a fair coin to decide which randomized sub-algorithm to run. That algorithm has approximation factor $3/4$:

$$\mathrm{E}[Z] = \sum_{i=1}^{m} \mathrm{E}[Z_i] \geq \sum_{i=1}^{m} \frac{1}{2} \left((1 - 2^{-|C_i|}) + \left(1 - \left(1 - \frac{1}{|C_i|} \right)^{|C_i|} \right) \right) \geq (3/4)m.$$

3 Minimum Makespan Revisited

Recall the minimum makespan problem. We have set J of jobs to schedule on set M of machines and we want to minimize the makespan, the maximum load on a machine. In the previous lecture, we assumed that each job has the same processing time on all machines. Here we consider a more general version of the problem in which job j can have different processing time on different machines.

Let p_{ij} be the time it takes machine i to process job j. We want to minimize the makespan $T = \max_i \sum_j x_{ij}p_{ij}$ where the variable $x_{ij} \in \{0,1\}$ indicates whether job j is assigned to machine i. We can write this as an integer program with constraints ensuring that each job is assigned to exactly one machine and that the load of each machine does not exceed T, the makespan.

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

First, one may try to relax variable x_{ij} and let $x_{ij} \in [0,1]$, however, the solution of the corresponding LP may be too far from the solution of the IP. Thus solution of LP does not serve as a tight lower bound for OPT. For instance, if we have only 1 job, m machines, and $p_{i1} = m$, $i \in M$, then OPT$= m$, however, solution of LP is 1 by assigning $x_{i1} = \frac{1}{m}$.

The maximum ratio of the optimal IP and LP solutions is called the *integrality gap*. We need to define the relaxed problem in such a way that the integrality gap is small.

The idea is to ensure that if $p_{ij} > T$ we assign $x_{ij} = 0$. We do this by defining a series of feasibility LPs with a makespan parameter T as follows.

$$\begin{aligned} & \sum_{i\,:\,p_{ij} \leq T} x_{ij} = 1 \qquad \forall\, j \in J \\ & \sum_{j \in J} x_{ij} p_{ij} \leq T \qquad \forall\, i \in M \\ & x_{ij} \geq 0 \qquad \forall i \in M \;\; j \in J \end{aligned}$$

Using binary search we can obtain the smallest value of T, T^*, for which the above feasibility linear program (FLP) has a solution. It is easy to show that in such a solution we assign $x_{ij} = 0$ if $p_{ij} > T^*$.

Claim 1 *FLP assigns at most $n + m$ (fractional) jobs to the machines.*

Proof: Recall that a solution x is a vertex of the polyhedron formed by the constraints. Note that there are $|x| + n + m$ total constraints, where $|x|$ of them are of the from $x_{ij} \geq 0$. At a vertex, $|x|$ linearly independent constraints must be satisfied; at least $|x| - (m+n)$ of them are of the form $x_{ij} = 0$, therefore the number of non-zero x_{ij}'s is at most $m + n$. ■

We now proceed to rounding the solution of FLP with a minimum value of T given by T^*. Let $G(J, M, E)$ be a bipartite graph defined on the set of jobs and machines where edge (i, j) between machine i and job j has weight x_{ij}.

The graph G has $n+m$ vertices and by the above claim there are at most $n+m$ edges. Thus if G were connected, we would need to remove one edge to obtain a tree; roughly speaking, graph G is "close" to a tree (or a forrest if it has more than one connected components). Note that for each feasible solution, $\sum_{i:\ p_{ij}\leq T^*} x_{ij} = 1$, $j \in J$, and $\sum_{j\in J} p_{ij}x_{ij} \leq T^*$, $i \in M$. We modify the weights x to obtain x' in such a way that the resulting graph $G(J, M, E')$ is a tree (forrest) and the constraints on sides J and M remain satisfied, i.e., $\sum_{i:\ p_{ij}\leq T^*} x'_{ij} = 1$, $j \in J$, and $\sum_{j\in J} p_{ij}x'_{ij} \leq T^*$, $i \in M$.

Suppose x' is given. Note that if node $j \in J$ is a leaf of the tree with parent $i \in M$, then $x_{ij} = 1$, thus there is no j leaf with fractional weight. However, we can have fractional edge (i, j) between leaf $i \in M$ and its parent $j \in J$. Suppose j has k leaves $i_1, i_2, \ldots, i_k$, we choose one of the leaf machines uniformly at random and assign job j to it. Then we remove j from the graph. We repeat this procedure of assigning fractional load of a parent to one its children and removing the job from the graph until all the jobs are assigned.

Claim 2 *The above rounding procedure produces a factor* 2 *approximation.*

Proof: Let OPT be the makespan of the optimal assignment and let T^* be the minimum value of T found using binary search on FLP. Then, $T^* \leq OPT$ since the FLP is clearly feasible using the optimal assignment. x' is a feasible solution therefore load of each machine is at most T^*. During the rounding procedure, we add the load of at most one job to each machine because a node i can only have one parent in G'. Suppose machine i is a leaf of job j_p, $L'_i = \sum_{j\in J} x_{ij}p_{ij} \leq T^*$, and the load of job j_p is assigned to machine i. Thus the new load of machine i is less than $L'_i + p_{ij_p}$; since $x_{ij_p} \neq 0$ we know that $p_{ij_p} \leq T^*$, thus $L'_i + p_{ij_p} \leq 2T^*$. Hence, the final makespan is at most $2\,T^*$. ■

The remaining task is to show that we can covert x to x' such that the underlying graph become a tree. Suppose $G = (J, M, E)$ is not a tree, thus it has cycle $c = i_1, j_1, i_2, j_2, \ldots, i_r, j_r, i_1$; suppose we update $x_{i_1j_1}$ to $x_{i_1j_1} - \epsilon^1$, we proceed around cycle c and update the weight of edges in the following way:

Since the total weight of edges incident to j_1 must add up to one, if we decrease $x_{i_1j_1}$ by ϵ^1, we need to increase $x_{i_2j_1}$ by ϵ^1, thus we update $x_{i_2j_1}$ to $x_{j_1i_2} + \epsilon^1$. Now to keep the load of machine i_2 less than or equal to T^*, we decrease $x_{i_2j_2}$ by $\epsilon^2 = \frac{p_{i_2j_1}}{p_{i_2j_2}}\epsilon^1$, repeating this procedure, we modify the weights keeping the constraints satisfied. The only constraint that may become unsatisfied is the load of machine i_1; we decrease the load of i_1 by ϵ^1 via edge (i_1, j_1) and at the end, we may increase the load by $\frac{p_{i_2j_1}}{p_{i_2j_2}}\frac{p_{i_3j_2}}{p_{i_3j_3}}\ldots\frac{p_{i_rj_{r-1}}}{p_{i_rj_r}}\epsilon$. if $\frac{p_{i_2j_1}}{p_{i_2j_2}}\frac{p_{i_3j_2}}{p_{i_3j_3}}\ldots\frac{p_{i_rj_{r-1}}}{p_{i_rj_r}} > 1$ we use the simple observation that if we start from (i_1, j_r) and go around the cycle in that direction, then we would need to increase the weight of (i_1, j_1) by $\left(\frac{p_{i_2j_1}}{p_{i_2j_2}}\frac{p_{i_3j_2}}{p_{i_3j_3}}\ldots\frac{p_{i_rj_{r-1}}}{p_{i_rj_r}}\right)^{-1} < 1$ thus the total load of i_1 would become less that T^*.

Using the above scheme, we are able to decrease the weight of (i_1, j_1) by ϵ keeping the solution feasible. Repeating this reduction, we can make the weight of one of the edges zero, thus we can remove cycle c. We obtain x' by breaking all the cycles.