CME 305: Discrete Mathematics and Algorithms

1 Approximation Algorithms

In light of the apparent intractability of the problems we believe not to lie in P, it makes sense to pursue ideas other than complete solutions to these problems. Three standard approaches include:

- *Exploiting special problem structure:* perhaps we do not need to solve the general case of the problem but rather a tractable special version.
- *Heuristics:* procedures that tend to give reasonable estimates but for which no proven guarantees exist.
- *Approximation algorithms:* procedures which are proven to give solutions within a factor of optimum.

Of these approaches, approximation algorithms are arguably the most mathematically satisfying, and will be the subject of discussion for this section.

An algorithm is a **factor α approximation** (α-approximation algorithm) for a problem iff for every instance of the problem it can find a solution within a factor α of the optimum solution.

If the problem at hand is a minimization then $\alpha > 1$ and this definition implies that the solution found by the algorithm is at most α times the optimum solution. If the problem is a maximization, $\alpha < 1$ and this definition guarantees that the approximate solution is at least α times the optimum.

1.1 Minimum Vertex Cover

Given a graph $G(V, E)$, the **minimum vertex cover problem** is to find a subset $S \subseteq V$ with minimum cardinality such that every edge in E has at least one endpoint in S.

Algorithm 1. 2-Approximation Algorithm for Minimum Vertex Cover

Find a maximal matching M in G.
Output the endpoints of edges in M: $S = \bigcup_{e \in M} e$.

Claim 1. *The output of algorithm 1 is feasible.*

Proof: We prove this by contradiction: suppose there exists an edge $e = (u, v)$ such that $u, v \notin S$. Since e does not share an endpoint with any of the vertices in M, $M \cup \{e\}$ is a larger matching, which contradicts M being a maximal matching. ■

Lemma 1. *Algorithm 1 gives a 2-approximation for minimum vertex cover regardless of the choice of M.*

Proof: The edges of M are independent; thus any feasible cover must take at least one vertex from every edge in M. This means that $|M| \leq OPT$ and then we have:

$$|M| \leq OPT \leq |S| = 2|M| \leq 2OPT.$$

■

This technique of lower bounding the optimum is often key in proving approximation factors, as we are usually unable to compute the value of OPT.

1.2 Minimum Weight Vertex Cover

Given a graph $G(V, E)$, and a weight function $W : V \mapsto \mathbb{R}^+$, the **minimum weight vertex cover problem** is to find a subset $S \subseteq V$ that covers all the edges and has minimum weight $W(S) = \sum_{v \in S} w(v)$.

We may formulate this problem as an integer program as follows: associate variable x_v with node v, and solve:

$$\begin{aligned} \textbf{minimize:} \quad & \sum_{v \in V} w(v) x_v \,; \\ \textbf{s.t.} \quad & x_u + x_v \geq 1, \ \forall \ (u, v) \in E \,; \\ & x_v \in \{0, 1\} \ \forall \ v \in V. \end{aligned}$$

Solving this integer program is equivalent to solving min weight vertex cover, an NP-complete problem. We instead attempt to solve a simpler problem for which polynomial-time algorithms exists. We modify the second constraint to read: $0 \leq x_v \leq 1 \ \forall \ v \in V$. This is known as a **linear programming (LP) relaxation** of the integer program. It is well known that linear programs can be solved in polynomial time.

Let $\{x_v^*, \ v \in V\}$ be the solution of the LP. We need to convert this fractional solution to an integral one; we use the following rounding policy: if $x_v^* < 1/2$ then we set $\widehat{x}_v = 0$, otherwise we set $\widehat{x}_v = 1$. Note that $\{\widehat{x}_v, \ v \in V\}$ is a feasible solution. Because $\{x_v^*, \ v \in V\}$ is a feasible solution of the LP, $x_v^* + x_u^* \geq 1$, and thus $x_v \geq 1/2$ or $x_u \geq 1/2$ which implies that $\widehat{x}_v + \widehat{x}_u \geq 1$.

Lemma 2. *Let $S = \{v \in V : \widehat{x}_v = 1\}$. Then S gives a 2-approximation to min weight vertex cover, i.e.*

$$\sum_{v \in S} w(v) \leq 2w(S^*),$$

where S^* is the optimum solution.

Proof: Since the feasible region of the IP is a subset of the feasible region of the LP, the optimum of the LP is a lower bound for the optimum of the IP. Moreover, note that our rounding procedure ensures that $\widehat{x}_v \leq 2x_v^*$ for all $v \in V$, thus:

$$OPT_{IP} \leq w(S) = \sum_{v \in V} w(v)\widehat{x}_v \leq 2 \sum_{v \in V} w(v) x_v^* = 2OPT_{LP} \leq 2OPT_{IP}.$$

■

1.3 Job Scheduling

Suppose we have n jobs each of which take time t_i to process and m identical machines on which to schedule their completion. Jobs cannot be split between machines. For a given scheduling, let A_j be the set of jobs assigned to machine j. Let $L_j = \sum_{p \in A_j} t_p$ be the load of machine j. The **minimum makespan scheduling problem** is to find an assignment of jobs to machines that minimizes the makespan, defined as the maximum load over all machines (i.e. $\max_j L_j$).

We consider the following greedy algorithm for this problem which sorts the jobs so that $t_1 \geq t_2 \geq ... \geq t_n$, and iteratively allocates the next job to the machine with the least load.

We note that algorithm 2 need not return an optimal solution. Considering jobs of sizes $\{3, 3, 2, 2, 2\}$ to be assigned to two machines we see that it is no better than a 7/6-approximation algorithm. We prove in these

Algorithm 2. Greedy Approximation Algorithm for Job Scheduling

$\forall j$, $A_j \leftarrow \emptyset$, $T_j \leftarrow 0$
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

notes that algorithm 2 has an approximation factor of no worse than 3/2; we leave as an exercise to the reader to prove that it is actually a 4/3-approximation algorithm.

Let T^* denote the optimal makespan. The following two facts are self-evident:

$$T^* \geq \max_r t_r\,;$$
$$T^* \geq \frac{1}{m}\sum_{p=1}^{n} t_p\,.$$

Claim 2. *The solution of the greedy makespan algorithm is at most*

$$\frac{1}{m}\sum_{p=1}^{n} t_p + \max_r t_r\,.$$

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 load smaller than the average load. Then,

$$T_j = (T_j - t_i) + t_i \;\leq\; \frac{1}{m}\sum_{p=1}^{n} t_p + \max_r t_r\;.$$

■

This claim shows immediately that algorithm 2 is a 2-approximation algorithm. Slightly more careful analysis proves $\alpha = 3/2$.

Lemma 3. *The approximation factor of the greedy makespan 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, then by the pigeonhole principle at least one processor in the optimal scheduling must get 2 of the first $m+1$ jobs. Each of these jobs is at least as big as t_{m+1}. Thus, $T^* \geq 2t_{m+1}$.

Take the output of algorithm 2 and consider machine j assigned maximum load T_j. Let i be the last job assigned to j. We may assume $i > m$ or else the algorithm's output is optimal. Since the jobs are sorted, $t_i \leq t_{m+1} \leq T^*/2$ and we have

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

■

1.4 Non-Uniform Job Scheduling

Consider a more general version of the minimum makespan scheduling problem in which job i can have different processing time on different machines. Let t_{ij} be the time it takes machine j to process job i. We want to minimize the makespan $T = \max_j \sum_i x_{ij} t_{ij}$ where the variable $x_{ij} \in \{0, 1\}$ indicates whether job i is assigned to machine j. We may 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.

$$\begin{aligned} \textbf{minimize:} \quad & T; \\ \textbf{s.t.} \quad & \sum_j x_{ij} = 1 \qquad \forall\, i\,; \\ & \sum_i x_{ij} t_{ij} \le T \qquad \forall\, j\,; \\ & x_{ij} \in \{0, 1\}. \end{aligned}$$

To approximate a solution to this integer program, one might first 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 $t_{1j} = m$ for every machine j, then OPT$= m$ but the solution of the LP is 1 by assigning $x_{1j} = \frac{1}{m}$.

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 $t_{ij} > T$ we assign $x_{ij} = 0$. We do this by defining a series of feasibility LPs with a makespan parameter T as follows:

$$\begin{aligned} & \sum_j x_{ij} = 1 \qquad \forall\, i\,; \\ & \sum_i x_{ij} t_{ij} \le T \qquad \forall\, j\,; \\ & 0 \le x_{ij} \le 1 \qquad \forall\, i, j\,; \\ & x_{ij} = 0 \qquad \forall\, i, j \text{ s.t. } t_{ij} > T\,. \end{aligned}$$

Using binary search we can obtain the smallest value of T, T^*, for which the above feasibility linear program (FLP) has a solution. We outline a procedure for rounding the solution of the FLP with $T = T^*$.

Let $G(J, M, E)$ be a bipartite graph defined on the set of jobs and machines where edge $\{i, j\}$ between job $i \in J$ and machine $j \in M$ has weight x_{ij}. Our claim is that this allocation graph can be transformed into a forest in such a way that the load of every machine remains the same or decreases. Suppose $G=(J,M,E)$ is not a forest, thus it has cycle $c = j_1, i_1, j_2, i_2, \ldots, j_r, i_r, j_1$; suppose we update $x_{i_1 j_1}$ to $x_{i_1 j_1} - \epsilon^1$, we proceed around cycle c and update the weight of edges in the following way:

Since the total weight of edges incident to i_1 must add up to one, if we decrease $x_{i_1 j_1}$ by ϵ^1, we need to increase $x_{i_1 j_2}$ by ϵ^1, thus we update $x_{i_1 j_2}$ to $x_{i_1 j_2} + \epsilon^1$. Now to keep the load of machine i_2 less than or equal to T^*, we decrease $x_{i_2 j_2}$ by $\epsilon^2 = \frac{t_{i_1 j_2}}{t_{i_2 j_2}} \epsilon^1$, repeating this procedure, we modify the weights keeping the constraints satisfied. The only constraint that may become unsatisfied is the load of machine j_1; we decrease the load of j_1 by ϵ^1 via edge $\{i_1, j_1\}$ and at the end, we may increase the load by $\frac{t_{i_1 j_2}}{t_{i_2 j_2}} \frac{t_{i_2 j_3}}{t_{i_3 j_3}} \cdots \frac{t_{i_{r-1} j_r}}{t_{i_r j_r}} \epsilon^1$. If $\frac{t_{i_1 j_2}}{t_{i_2 j_2}} \frac{t_{i_2 j_3}}{t_{i_3 j_3}} \cdots \frac{t_{i_{r-1} j_r}}{t_{i_r j_r}} > 1$ we use the simple observation that if we start from $\{j_1, i_r\}$ and go around the cycle in that direction, then we would need to increase the weight of $\{i_1, j_1\}$ by $\left(\frac{t_{i_1 j_2}}{t_{i_2 j_2}} \frac{t_{i_2 j_3}}{t_{i_3 j_3}} \cdots \frac{t_{i_{r-1} j_r}}{t_{i_r j_r}} \right)^{-1} < 1$ thus the total load of i_1 would become less than T^*.

Using the above scheme, we are able to decrease the weight of $\{i_1, j_1\}$ by ϵ^1 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 our forest by breaking all the cycles.

Suppose now that we have a solution of the FLP with $T = T^*$ whose allocation graph is a forest, call it x. Note that if job $i \in J$ is a leaf of the tree with parent $j \in M$, then $x_{ij} = 1$, thus there is no J leaf with fractional weight. However, we can have fractional edge $\{i, j\}$ between leaf $j \in M$ and its parent $i \in J$. Suppose i has k leaves $j_1, j_2, \ldots, j_k$. Afterwards we choose one of the leaf machines at random and assign job i to it with probability x_{ij}. Then we remove i from the graph. We repeat this procedure of assigning fractional load of a parent to one of its children and removing the job from the graph until all the jobs are assigned.

Claim 3. *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 the forest G. Any machine j's load before this addition is $\sum_{i \in J} x_{ij} t_{ij} \leq T^*$, so the new load of machine j is less than $2T^*$. Hence, the final makespan is at most $2T^*$. ■

1.5 Maximum Satisfiability

We return to the setting of boolean formulas and consider a problem related to satisfiability: for a given formula in CNF, what is the maximum number of its clauses that can be satisfied by assigning truth values to its variables? More concretely, suppose we have n variables $x_1, ..., x_n$ and m clauses $C_1, ..., C_m$, where

$$C_i = \left(\bigvee_{j \in S_i^+} x_j \right) \vee \left(\bigvee_{j \in S_i^-} \bar{x}_j \right).$$

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. Then the probability that any C_i is satisfied is $1 - 2^{-|C_i|}$. 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:

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

An expected approximation ratio is sometimes undesirable because it does not shed much light on the probability that the randomized 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 $\frac{1}{2}\mathrm{E}[Z \mid x_j = 1, x_{j-1}, ..., x_1]$ and $\frac{1}{2}\mathrm{E}[Z \mid x_j = 0, x_{j-1}, ..., x_1]$.
  Set $x_j = 1$ if the first expression is larger than the second, set $x_j = 0$ 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 no longer an expectation, but merely an evaluation.

The approximation ratio of algorithm 3 depends on all of the clauses having K or more 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_j. We relax the last condition to be $0 \leq q_i, y_j \leq 1$ in order to get a linear program.

Algorithm 4. LP Relaxation 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; WLOG we

consider $C_1 = x_1 \vee ... \vee x_k$. We have $q_1^* = \min\{y_1^* + ... + y_k^*, 1\}$ and by the AM-GM inequality:

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

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 better of the two assignments. 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:

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