CS5800 Algorithms

Recitations from Nov 18, 2019 to Nov 22, 2019

Ravi Sundaram

Problem 1 (Linear Programming)

1. A linear programming problem consists of some linear expression that we want to maximize/ minimize, and some inequalities involving the variables in the expression.

   We will run through an example. As a busy Northeastern student you would like to optimize your weekly schedule. You have 4 main activities, which involve studying (S), eating (E), napping (N), and having fun (F).

   Firstly, there are 168 hours a week. To keep your brain working you need to spend at least 50 hours each week napping. To not flunk out you need to spend at least 40 hours a week studying. To not go insane you need to spend at least 25 hours each week having fun and eating. But having fun distracts your brain, so the more fun you have the more you need to study. This relationship can be expressed as $2S - F \geq 75$.

   After some meditation, you've decided that your happiness can be expressed simply as the expression $2F + E + 2N$. So we are interested in maximizing this amount.

   This can be expressed as a linear programming problem as follows.

   $$\begin{array}{ll} \text{maximize} & 2F + E + 2N \\ \text{subject to} & N \geq 50 \\ & S \geq 40 \\ & F + E \geq 25 \\ & 2S - F \geq 75 \\ & S + E + F + N \leq 168 \\ & S, E, F, N \geq 0 \end{array}$$

   Here the variables are S, E, F, N and $2F + E + 2N$ is the objective function. All the inequalities are the constraints of the problem. One possible solution that satifies all the constraints is $N = 60, S = 50, F = 15, E = 15$. A linear programming algorithm returns the values of the variables that maximizes the value of the objective function in polynomial time. Many problems can be expressed as linear programming problems, and hence be done in polynomial time

2. Formulate the max flow problem as a linear programming problem. Think of what the constraints are and what needs to be maximized.

3. The 3SAT problem is determing whether a boolean expression of a special form has a satisfying assignment or not. The special form is called conjunctive normal form (CNF), where each clause has exactly 3 literals. Each literal is either a variable or a negated variable, like x_i or $\overline{x_j}$. Literals can be repeats of the same variable.

To be more concrete, each clause is of the form $(z_{i_1} \vee z_{i_2} \vee z_{i_3})$, and the whole expression consists of a conjunction of clauses, in the form $(z_1^{c_1} \vee z_2^{c_1} \vee z_3^{c_1}) \wedge (z_1^{c_2} \vee z_2^{c_2} \vee z_3^{c_2}) \wedge ... \wedge (z_1^{c_m} \vee z_2^{c_m} \vee z_3^{c_m})$, where m is the number of clauses in the expression.

An example of a 3SAT instance is given the boolean formula

$$\phi = (x_1 \vee \overline{x_4} \vee x_2) \wedge (\overline{x_1} \vee \overline{x_3} \vee x_4)$$

is there an assignment of the 4 variables x_1, x_2, x_3, x_4 so that the expression evaluates to true?

In this case, the answer is yes and there are multiple assignments that makes the expression evaluate to true. One possible assignment is

$$x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0$$

On the other hand the following assignment makes ϕ evaluate to false.

$$x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 1$$

Formulate 3SAT as a linear programming problem. Does this mean 3SAT can be done in polynomial time?

Solution:

1.

$$\begin{array}{lll} \text{maximize} & \sum_{(s,u)\in E} x_{s,u} & \\ \text{subject to} & x_{u,v} \le c(u,v) & \forall (u,v) \in E \\ & \sum_{(a,v)\in E} x_{a,v} = \sum_{(v,b)\in E} x_{v,b} & \forall v \in V, v \notin \{s,t\} \end{array}$$

We create the variable $x_{u,v}$ for each edge (u, v) whose value represents the flow going through edge (u, v). The first constraint is the amount of flow can't be more than the capacity of the edge, which we denote by $c(u, v)$. Now in a flow graph, each vertex besides s, t has to have the same amount of flow going in as flow coming out (conservation of flow), which is what the second constraint captures.

Finally, we want to maximize total flow from s to t, which is the same as maximizing the amount of flow coming out of s (or equivalently, the amount of flow going into t).

2.

$$\begin{array}{lll} \text{maximize} & \sum_{j=1}^{m} c_j & \\ \text{subject to} & c_j \le x_{j_1} + x_{j_2} + x_{j_3} & \text{for } 1 \le j \le m, \text{if } c_j = (x_{j_1} \vee x_{j_2} \vee x_{j_3}) \\ & c_j \le x_{j_1} + x_{j_2} + (1 - x_{j_3}) & \text{if } c_j = (x_{j_1} \vee x_{j_2} \vee \overline{x_{j_3}}) \\ & c_j \le x_{j_1} + (1 - x_{j_2}) + x_{j_3} & \text{if } c_j = (x_{j_1} \vee \overline{x_{j_2}} \vee x_{j_3}) \\ & c_j \le x_{j_1} + (1 - x_{j_2}) + (1 - x_{j_3}) & \text{if } c_j = (x_{j_1} \vee \overline{x_{j_2}} \vee \overline{x_{j_3}}) \\ & \vdots & \\ & x_i \in \{0, 1\} & \text{for } 1 \le i \le n \\ & c_j \in \{0, 1\} & \text{for } 1 \le j \le m \end{array}$$

(Where n is the number of variables in ϕ and m is the number of clauses in ϕ).

For each variable x_i in the formula we create a variable for our linear program under the same name. If x_i is 1, then that's saying x_i is set to true. So if there's a literal x_i it will evaluate to

true, and if there's a $\overline{x_i}$ literal it will evaluate to false. And for each clause j in the formula, we create the variable c_j. If the clause evaluates to true, then c_j will take the value 1, otherwise it takes the value 0.

Now, we need to come up with the constraint that ties a clause to the actual variables inside of it. For each c_j, we define the constraints based on the literals that appear in it. For example, say $c_j = (x_{j_1} \vee \overline{x_{j_2}} \vee x_{j_3})$. Then the constraint we add is $c_j \leq x_{j_1} + (1 - x_{j_2}) + x_{j_3}$. The general rule is we add a x_{j_1} to the right hand side if x_{j_1} appears as a positive literal in c_j, and add $(1 - x_{j_2})$ if x_{j_2} appears as a negative literal in c_j. Now note $x_{j_1} + (1 - x_{j_2}) + x_{j_3} = 0$ only when $x_{j_1}, x_{j_3} = 0$ and $x_{j_2} = 1$, which is exactly the setting of variables that makes c_j evaluate to false. Otherwise, the right hand side will be between 1 and 3. Note that c_j is either 1 or 0 (as it just represents whether c_j is true or not). So when we maximize over all c_j, if a clause is satisfied it takes the value 1, otherwise it takes the value 0. So the original formula is satisfiable if the objective function hits the value m (where m is the number of clauses in the boolean formula).

Note that this is an integer linear program. Integer Linear Programming is an NP-Complete problem, as opposed to regular linear programming (where the variables take values in a continuous range). The above formulation can be relaxed, with instead of $x_i \in \{0, 1\}$ we can say $x_i \in [0, 1]$. We will be able to get some fractional solution in polynomial time, but it doesn't really give us a full answer, since setting x_i to $1/2$ doesn't make sense in the boolean setting.

...

■

Problem 2 (Reductions for NP-Complete Problems)

1. 3SAT is the canonical NP-complete problem, and it used often to show different problems are NP-complete as well. This is done by reductions, where if we want to show some problem $\mathcal{A}$ is NP-complete, we show that if there is some hypothetical algorithm for solving $\mathcal{A}$, then that algorithm can be used to solve 3SAT.

   We are going to show that the Independent Set problem is NP-hard. The Independent Set problem is defined as the problem of determining whether a given graph $G = (V, E)$ has a subset of vertices $V \in V'$ of size k such that for all $u, v \in V'$, the edge $(u, v) \notin E$. In other words, no two vertices in the independent set V' are connected to each other by an edge.

   Show that the Independent Set problem is NP-hard, by giving a reduction from 3SAT.

   (Hint: Suppose you are given some 3SAT instance ϕ. Convert this boolean expression into a graph $G = (V, E)$ such that ϕ has a satisfying assignment if and only if G has an independent set of some size k (the k needs to be thought about as well). This implies that if we had an algorithm to solve Independent Set, we can use it to solve 3SAT).

Solution:

Suppose we are given some arbitrary instance of 3SAT ϕ. We need to convert it into an instance (G, k) of Independent set such that $\text{3SAT}(\phi) = YES$ if and only if $\text{independent set}(G, k) = YES$.

In other words, if there exists some satisfying assignment for ϕ, then there exists some independent set of size k in G.

And if there exists some independent set of size k in G, then there exists some satisfying assignment for ϕ.

Given a ϕ, this is how we convert it into a graph. For each clause $c_j = (x_{j_1} \vee \overline{x_{j_2}} \vee x_{j_3})$, we create the vertices $x_{j_1}, \overline{x_{j_2}}, x_{j_3}$ and create an edge between each pair of vertices. This creates a triangle. As a note, even if a literal appears in multiple clauses we still create a fresh vertex for each clause the literal appears in. Now for the vertex x_{j_1} in clause j, we also connect it to all vertices $\overline{x_{j_1}}$ in different clauses. Likewise, we create an edge between $\overline{x_{j_2}}$ and all appearances of x_{j_2} in different clauses. In other words, for each literal we connect it to all negations of it (if there are any) in different clauses.

We are done with creating G. The last thing is we need to define the k parameter. In this case it is simply m, where m is the number of clauses in ϕ. Note that is also the number of triangles we just created. Also note that the largest independent set in a triangle is of size 1. This is important. In this reduction, if a vertex $\overline{x_{j_2}}$ is in the independent set in triangle j, we think of $\overline{x_{j_2}}$

Now we prove the correctness of the reduction.

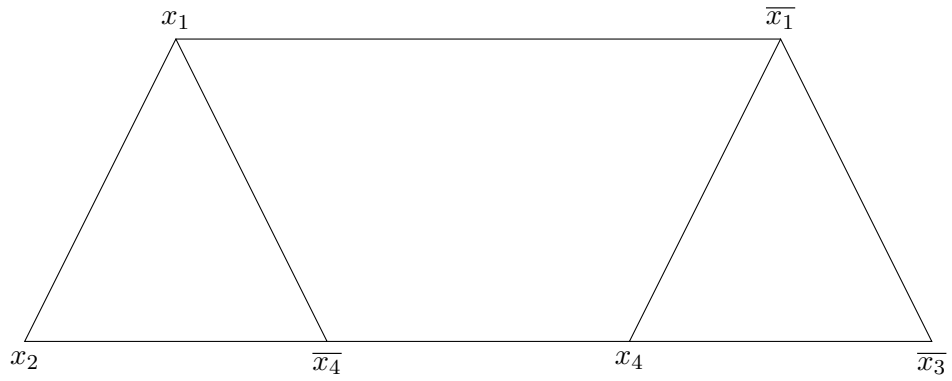
First suppose ϕ has some satisfying assignment x. Then we want to show that G has an independent set of size m.

So we know that under this hypothetical x, every clause evaluates to true. So for each clause j we pick the literal that makes the clause evaluate to true (if there are multiple, then we pick one arbitrarily) and put it in our independent set. Since each clause has at least one literal that evaluates to true, we put m vertices in our independent set. They are all in different triangles. And note that if say x_{j_1} make clause j true, that means $\overline{x_{j_1}}$ will never make some other clause k true. So none of the vertices we picked will share any edges, so we created an independent set of size m.

Now for the second direction, suppose G has some independent set of size m. For some triangle j, suppose $\overline{x_{j_2}}$ is in the independent set. Then we say that $\overline{x_{j_2}}$ makes clause j true, so that means we set $x_{j_2} = 0$. Note that if $\overline{x_{j_2}}$ is in the independent set, that means we can never have some x_{j_2} vertex in a different triangle also be in the independent set. Which is good, since if we did that would mean we set $x_{j_2} = 1$ which is contradictory. By the nature of the graph, if there's an independent set of size m there's exactly one vertex in the independent set in each triangle. So have a way to set the variables to make each clause true in ϕ, which is equivalent to finding a satisfying assignment for ϕ. Now there might be some variables that are left unassigned, which we can then assign arbitrarily, since all the clauses are already satisfied.

Now we are done.

As an example, below is the graph G for the formula $\phi = (x_1 \vee \overline{x_4} \vee x_2) \wedge (\overline{x_1} \vee \overline{x_3} \vee x_4)$.



■

Problem 3 (Extras...)

1. The Hamilton Cycle problem is defined as the problem of determining whether a given graph $G = (V, E)$ has a Hamiltonian cycle or not. A Hamiltonian cycle is defined as a cycle that visits every vertex in G exactly once. We've seen Eulerian paths before, which is similar except its a path, and the path has to visit every edge exactly once (vertices can be repeated).

   Show that the Hamilton Cycle problem is NP-hard by giving a reduction from 3SAT.

2. Show that Vertex Cover is NP-hard by a reduction from 3SAT.