Optimization I **Summer 2011**
Lecture 23

Dealing with NP-hard Optimization Problems

Our goal so far in developing algorithms for optimization problems has been to find algorithms that

(a) find the optimal solution;

(b) run in polynomial time;

(c) have property (a) and (b) for *any* input.

We have seen that we can indeed do this for optimization problems that can be formulated as a *linear program*. For problems that can be formulated as an integer linear program, we are not so lucky. In fact, unless P=NP, we cannot find algorithms that satisfy (a), (b) and (c) for a general integer linear program. Therefore, unless P=NP, we are going to have to focus on developing algorithms that have just two of the three properties (and do the best we can with respect to the other property).

In this lecture, we will illustrate these three approaches using vertex cover as an example.

Definition 1. *Given an undirected graph $G = (V, E)$, the Minimum Vertex Cover problem asks for a vertex cover of minimum size, i.e., a set of nodes $S \subseteq V$ of minimum size $|S|$ such that every edge $e \in E$ has at least one endpoint in S.*

The Minimum Vertex Cover problem is known to be NP-hard, so we don't expect to find a polynomial time algorithm that finds the optimal solution for every possible input.

If we drop requirement (c), and restrict our attention to certain classes of inputs, we can however have algorithms that solve the problem in polynomial time: for example if we restrict the input to bipartite graphs.

1 Fixed Parameter Tractability

Suppose we drop the requirement that we satisfy property (b), and we just try to find a minimum vertex cover with a "good" but not polynomial time algorithm.

Suppose that the size of the minimum vertex cover is a fixed constant k, say $k = 2$ or $k = 3$. For such inputs, it is not hard to find a minimum vertex cover in polynomial time. If we know the value k, we just try all subsets of size k, and check whether they are a vertex cover. There are $\binom{n}{k}$ subsets of size k, and checking them takes $O(m)$ time, where $n = |V|, m = |E|$. If we don't know k in advance, we can find the minimum vertex cover by checking whether there exists a vertex cover of size ℓ, for $\ell = 0, \ell = 1, \ldots, \ell = k$. Therefore, if the minimum vertex cover has size k, we can find it in time

$$O(\left(\binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \ldots + \binom{n}{k}\right) m) = O(kn^k m)$$

This is polynomial for fixed k, so for some inputs, this gives a polynomial time algorithm. However, this algorithm is far from practical (according to Kleinberg-Tardos, it will take longer than the age of the universe for an input with $n = 1000, k = 10$). Can we do something smarter?

Observation 1. *If G has n nodes and a vertex cover of size k, then G has at most $k(n-1)$ edges.*

Observation 2. *Let $e = \{u, v\}$ be any edge in G. If G has a vertex cover of size k, then either $G - \{u\}$ or $G - \{v\}$ has a vertex cover of size $k - 1$.*

These two observations give rise to a simple recursive algorithm for finding a vertex cover of size k if it exists:

- If G has no edges, return the empty set.
- If G has more than $k(|V|-1)$ edges, then no vertex cover of size k exists.
- Else, let $e = \{u, v\}$ be an edge of G.
  - Find a vertex cover of size $k-1$ in $G - \{u\}$ and in $G - \{v\}$.
    If neither of those exists, then G has no vertex cover of size k.
    Else, if T is a vertex cover of size $k-1$ of $G - \{u\}$ (respectively, $G - \{v\}$), then return $T \cup \{u\}$ (respectively, $T \cup \{v\}$).

Theorem 1. *There exists an algorithm to check if a graph has a vertex cover of size k that runs in $O(2^k n)$ time.*

Proof. We can view the execution of the algorithm as a tree, in which each node is a different recursive call. Note that the tree is binary, and has (at most) $k+1$ levels. Hence, there are at most 2^{k+1} nodes in this tree. In each recursive call, we need to check if the number of edges in the current graph is more than $k(|V|-1)$, and we need to make (at most) two recursive calls for which we need to remove a node and its incident edges. We can assume that we can check the number of edges in constant time (by using an appropriate data structure). Removing an edge from the graph can be done in constant time and a node has at most n incident edges. Hence, in each node of the tree we do $O(n)$ work. □

Note that this is pretty good: If $k = O(\log n)$, then this is still a polynomial time algorithm. This is an example of a fixed parameter algorithm.

Definition 2. *A problem is fixed parameter tractable (FPT) with respect to parameter k if there is an algorithm with running time at most $f(k)n^{O(1)}$.*

If we consider the example with $n = 1000, k = 10$, it now only takes a few seconds to check whether a vertex cover of size k exists. On the other hand, the running time still grows exponentially: if $k = 40$ instead of $k = 10$, the algorithm will take a significant number of years to terminate...

2 Approximation algorithms

Another approach for dealing with NP-hardness is dropping the requirement (a) that the algorithm has to find the optimal solution. This is called a heuristic.

For the minimum vertex cover example, a reasonable heuristic seems to be the following:

- Repeat until E is empty
  - Pick a vertex v with highest degree,
  - Add v to S, and remove v and its incident edges from G.

Unfortunately, the solution returned by this heuristic can be pretty bad – there exists a family of examples for which the minimum vertex cover has size $k!$ and the vertex cover found by the heuristic has size $k! \log k$.

Definition 3. *For a minimization problem, an α-approximation algorithm is an algorithm that runs in polynomial time and is guaranteed to output a solution of cost at most α times the value of the optimal solution.*

One popular approach to developing approximation algorithms is to use linear programming. We will see two examples: LP rounding and primal-dual algorithms.

2.1 LP rounding

We can formulate the minimum vertex cover problem as an integer linear program as follows. We slightly generalize the problem, and allow each vertex to have a nonnegative weight $w_v \geq 0$ that is part of the input. The problem in Definition 1 is then just the special case when $w_v = 1$ for all $v \in V$.

For every vertex $v \in V$, we introduce a variable $x_v \in \{0, 1\}$. We think of $x_v = 1$ as representing that $v \in S$. Then we want to minimize $\sum_{v \in V} w_v x_v$, subject to the constraint that $x_u + x_v \geq 1$ for every $e = \{u, v\} \in E$. Let OPT be the optimal value of this integer linear program (which is the same as the optimal value of the minimum vertex cover problem).

If we relax this integer program, we get the following LP:

$$
\begin{aligned}
& \min && \sum_{v \in V} w_v x_v \\
(P_{VC}) \quad & \text{s.t.} && x_u + x_v \geq 1 \text{ for every } e = \{u, v\} \in E \\
& && x_v \geq 0 \text{ for } v \in V.
\end{aligned}
$$

(Note that we don't need to require that $x_v \leq 1$, because this will automatically be true for any optimal solution!)

Theorem 2. *There exists a 2-approximation algorithm for the minimum vertex cover problem.*

Proof. Let x^* be an optimal solution to (P_{VC}). Note that we can find x^* in polynomial time. Also, now that $\sum_{v \in V} w_v x_v^* \leq OPT$, since the optimal integer solution gives a feasible solution to the LP with objective value OPT. Now, we just round up the variables x^* that are greater than or equal to $\frac{1}{2}$. Let $\bar{x}$ be this rounded solution. Then $\bar{x}_u + \bar{x}_v \geq 1$ for every $\{u, v\} \in E$, since at least one of x_u^* and x_v^* must be at least $\frac{1}{2}$, so $\bar{x}$ is a feasible solution for the Vertex Cover problem. Also $\sum_{v \in V} w_v \bar{x}_v \leq 2 \sum_{v \in V} w_v x_v^* \leq 2OPT$. □

Although this LP rounding algorithm is nice and seems simple, in some sense it is not that simple: it needs us to solve the LP relaxation, and - although we can do this in polynomial time - it is not "easy". However, our knowledge of linear programming can also help us develop a very simple and fast algorithm.

2.2 Primal-dual algorithm

We begin by taking the dual of (P_{VC}):

$$
\begin{aligned}
& \max && \sum_{e \in E} y_e \\
(D_{VC}) \quad & \text{s.t.} && \sum_{u : \{u,v\} \in E} y_{\{u,v\}} \leq w_v \text{ for every } v \in V \\
& && y_e \geq 0 \text{ for } e \in E.
\end{aligned}
$$

We have the following lemma which follows from weak duality, and the fact that P_{VC} is a relaxation of the vertex cover problem.

Lemma 3. *Let y be a feasible solution to (D_{VC}), then $\sum_{e \in E} y_e \leq OPT$.*

We now have another lower bound on the optimal value, given by any solution to (D_{VC}). To get a 2-approximation algorithm, what we will do is (1) create a feasible solution to the vertex cover problem; (2) create a feasible solution to (D_{VC}); (3) show that our vertex cover solution costs at most twice as much as our solution to (D_{VC}). The algorithm works by uniformly increasing the dual variable y_e for all e that is not yet covered by the current solution. We will say that the vertex v *goes tight* if $\sum_{e = \{u,v\} \in E} y_e$ becomes equal to w_v.

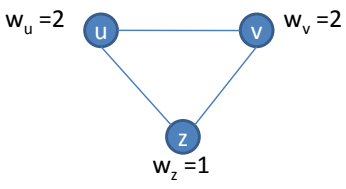

Figure 1: The primal dual algorithm outputs $y_{\{u,v\}} = \frac{3}{2}, y_{\{u,z\}} = \frac{1}{2}, y_{\{v,z\}} = \frac{1}{2}$ and $S = \{z, u, v\}$.

VC-Primal-Dual algorithm:

- $y \leftarrow 0; S \leftarrow \emptyset$
- While E is not empty
  - Increase y_e for all $e \in E$ at the same rate until some vertex v goes tight.
  - Add v to S and remove v and its incident edges from G.
- Return S, y.

Lemma 4. *The VC-Primal-Dual algorithm outputs a feasible vertex cover S. Also, it outputs a vector y which is feasible for (D_{VC}).*

Proof. At any point in the algorithm, E contains all the edges e that are not covered by S. Since the algorithm terminates when E is empty, the algorithm outputs a feasible vertex cover. The vector y is feasible initially. Once a vertex goes tight, we remove all its adjacent edges from E, hence y remains feasible throughout the course of the algorithm. □

Theorem 5. *The Primal-Dual algorithm is a 2-approximation algorithm for Vertex Cover.*

Proof. Let S, y be the output of the algorithm, then

$$\begin{aligned}
\sum_{v \in S} w_v &= \sum_{v \in S} \sum_{u:\{u,v\} \in E} y_{\{u,v\}} \\
&= \sum_{e \in E} \sum_{v \in S : v \text{ is an endpoint of } e} y_e \\
&\leq \sum_{e \in E} \sum_{v \in V : v \text{ is an endpoint of } e} y_e \\
&= \sum_{e \in E} y_e |\{v \in V : v \text{ is an endpoint of } e\}| \\
&= 2 \sum_{e \in E} y_e \\
&\leq 2OPT
\end{aligned}$$

□

Note that we did not do better than the previous LP rounding algorithm. In fact, no α-approximation algorithm for Vertex Cover is known for $\alpha < 2$ (and under a conjecture called the Unique Games Conjecture, it is hard to approximate vertex cover to within $2 - \epsilon$ for any $\epsilon > 0$ (which was shown by Khot and Regev.)) One reason one could prefer the primal-dual algorithm to the LP rounding algorithm is the fact that it is very fast, combinatorial and easy to implement. If you use a datastructure called Fibonacci heaps, you can implement this algorithm to run in $O(n \log n + m)$ time! Recall also the complementary slackness conditions:

Definition 4. *Let x be a solution to P_{VC} and let y be a solution to D_{VC}. Then x and y satsify complementary slackness if*

- $x_v > 0 \Rightarrow \sum_{u:\{u,v\} \in E} y_{\{u,v\}} \leq w_v$;
- $y_{\{u,v\}} > 0 \Rightarrow x_u + x_v = 1$.

Note that if we let $x_v = 1$ if v is in the vertex cover created by VC-Primal-Dual, then y and x satisfy the first type of Complementary Slackness conditions. This is a key ingredient of primal-dual *approximation* algorithms! Note that we cannot expect x and y to always satisfy the second type of Complementary Slackness conditions, as this would mean that x is an integer optimal solution to P_{VC}. If this were always the case, then P_{VC} would have integer extreme points (and Minimum Vertex Cover would be solvable in polynomial time).