ORIE 6300 Mathematical Programming I October 28, 2014

Lecture 18

Lecturer: David P. Williamson *Scribe: Venus Lo*

1 Good algorithms

1.1 What is a good algorithm?

We want to consider provably efficient algorithms for solving linear programs. What do we mean by this? We have said that the simplex method works very well in practice. Why isn't this enough? We'd like to get some mathematical sense of which problems have good algorithms and which do not, as long as there is roughly a correspondence between this and reality.

One of the first definitions of a good algorithm was given in the context of the perfect matching problem. In the perfect matching problem, we are given an undirected graph $G = (V, E)$, where $|V|$ is even, and edge costs $c_e \geq 0$ for all $e \in E$, find a subset $M \subseteq E$ of minimum cost such that each vertex has exactly one edge of M incident to it.

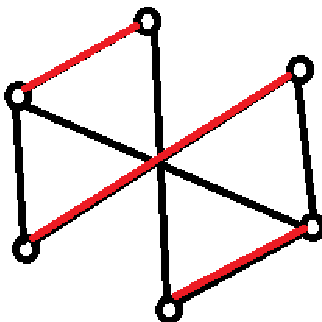

Figure 1: Example of a perfect matching depicted by the red edges

Definition 1 (Edmonds (1965)) *An algorithm for this problem is* good *if its running time can be bounded by a polynomial in the size of its inputs.*

Edmonds gave an $O(n^5)$ time algorithm for the minimum cost perfect matching problem, where $n = |V|$. Note that this is much better than consider all possible matchings. This has been improved on subsequently. However, the simplex method is not known to be "good" since no pivot rule can be proven to use less than $(2^n - 1)$ pivots in the worst case (recall the Klee-Minty cubes).

To get at some of the subtleties of this definition, we recall Euclid's algorithm for computing the GCD.

Example 1 *Input: gcd(a,b)*
If $b = 0$, return a.
Else gcd($b, a \mod b$)

Calling gcd(1071, 1029) $\rightarrow$ gcd(1029,42) $\rightarrow$ gcd(42,21) $\rightarrow$ gcd(21,0), so we return 21.

We claim that Euclid's algorithm is a good algorithm. Consider the input size to the problem.

Definition 2 Input size - the number of bits needed to encode input numbers in binary. So to encode x, we would need at most $\lceil \log_2(x+1) \rceil + 1$ bits.

Claim 1 *After two recursive calls, b drops by a factor of at least 2.*

This implies that after approximately $2 \log_2 b$ calls (for the initial b), the algorithm terminates, so its running time is polynomial in the size of the inputs.

Question: Are there good algorithms for solving LPs?
Answer: Yes! The ellipsoid method (not used in practice) and interior point methods (sometimes competitive to Simplex)

Definition 3 *A* strongly polynomial *algorithm for LP would have running time bounded by a polynomial in m and n. In order words, only the number of inputs (size of matrix, etc) matters, and not their magnitude.*

Important research question: Does a strongly polynomial algorithm exists for solving LPs?

Another interesting question is to know when there does *not* exist a good algorithm for a problem.

Question: Is there a good algorithm for the knapsack problem?
Answer: Previously we gave an algorithm with running time $O(mW^2)$. The inputs to the knapsack problems were the item sizes $\{s_1, s_2, \ldots, s_m\}$, the item values $\{y_1, y_2, \ldots, y_m\}$, and the knapsack size W. So the size of the inputs is $\lceil \log_2(W+1) \rceil + 1$. Thus the running time of $O(mW^2)$ is exponential in the input size of W since its size is $O(\log_2 W)$. In fact, we do not know if good algorithms exists for the knapsack problem. The knapsack problem is in the class of NP-hard problems, a class for which there are no known good algorithms.

1.2 Determining the input and output size of an LP

Consider the following linear program:

$$
\begin{aligned}
\min \quad & c^T x \\
& Ax \leq b \\
& x \geq 0
\end{aligned}
$$

Assume A, b, c are integer. What is the input size of this linear program?

- Encoding a vector v in binary takes $\sum_{j=1}^{n}$ size (v_j) bits $\equiv$ size(v) bits.
- Encoding an matrix A in binary takes $\sum_{j=1}^{n}\sum_{i=1}^{m}$ size (a_{ij}) bits $\equiv$ size(A) bits.

Define the input size of the LP to be $L = \text{size}(A) + \text{size}(b) + \text{size}(c)$.

We need to be able to bound the size of the output of LP solutions by a polynomial to have any hope of achieving a polynomial-time algorithm, otherwise we would not be able to write down the output in polynomial time. Let U be the maximum size of a_{ij}, b_i, c_j, so that $L = O(mnU)$.

What is the size of the LP solution? To determine the size of the output, we look at the size of writing down a basic solution to $\bar{A}x = \bar{b}$, where $\bar{A}$ and $\bar{b}$ are part of the inputs A and b. Then by Cramer's rule,

$$x_j = \frac{\det(\overline{A}_j)}{\det(\overline{A})},$$

where $\bar{A}_j$ is $\bar{A}$ with the jth column replaced by $\bar{b}$.

Now we can bound the determinant of $\bar{A}$:

$$\det(\bar{A}) = \sum_{\sigma \in S_n} (-1)^{sign(\sigma)} a_{1\sigma(1)} a_{2\sigma(2)} ... a_{n\sigma(n)}$$

where S_n is the set of permutations of $\{1, 2, ..., n\}$.

Since each $a_{i\sigma(i)}$ is at most U bits, the product has $\leq nU$ bits. This means the product is at most $\leq 2^{nU}$. There are $n!$ products which we need to sum over, so $x_j \leq n! 2^{nU}$. (Aside: The sum of two m-bits numbers has at most $(m+1)$-bits.) The output size of each x_j is at most:

$$
\begin{aligned}
\log_2(n! 2^{nU}) =& \log_2 n! + \log_2 2^{nU} \\
\leq& \log_2 n^n + \log_2 2^{nU} \\
=& n \log n + nU.
\end{aligned}
$$

Since we have m nonzero x_j in the output, the output size is $O(m(n \log n + nU))$, which is polynomial in the input size of $O(mnU)$.

2 The Ellipsoid Method for LP

The Ellipsoid Method: Given a bounded polyhedron, $P = \{x \in \mathbb{R}^n : Cx \leq D\}$, it either finds some $x \in P$ or or states "infeasible" if $P = \emptyset$.

How can we use this oracle to solve an optimization problem such as $\{\min c^T x : Ax \leq b, x \geq 0\}$? We can find the optimal solution by making three calls to it.

1. Check for primal feasibility: $\exists\, x$ s.t. $Ax \leq b, x \geq 0$? If not, done, the primal is infeasible.
2. Check dual feasibility: $\exists\, y$ s.t. $A^T y \leq c, y \geq 0$? If not, done, the primal is unbounded.
3. Find optimal solution by running ellipsoid method on:

$$\begin{aligned} c^T x &\leq b^T y \\ Ax &\leq b \\ x &\geq 0 \\ A^T y &\leq c \\ y &\geq 0 \end{aligned}$$

By our proof of strong duality, we know that if both the primal and dual are feasible, then there will exist feasible solutions to the primal and dual such that their objective functions are equal. So if we get to Step 3, then the algorithm should return optimal (x, y).

Suppose our oracle only gives "feasible" or "infeasible" responses (i.e. it does not give us a point x or y). We can still find a vertex of $Cx \leq d$ as follows:

Set $I \leftarrow \emptyset$.
For $i = 1, \ldots, m$:

- Check system :

$$\begin{aligned} C_j x &\leq d_j & \forall j \notin I \\ C_i x &= d_i & \\ C_j x &= d_j & \forall j \in I \end{aligned}$$

- If system is feasible, $I \leftarrow I \cup \{i\}$.

When this terminates, we can solve $\{C_j x = d_j, \forall j \in I\}$ via Gaussian elimination. This gives us a vertex of $\{Cx \leq d\}$.