18.415/6.854 Advanced Algorithms		October 10, 2001
	Lecture 8	
Lecturer: Michel X. Goemans		*Scribe: Jelena Spasojevic*

Interior Point Algorithm Wrap Up

In last lecture we were discussing the Interior Point Algorithm. Our goal was to optimize barrier problem $BP(\mu)$:

$$\min\ c^T x - \mu \sum \ln x_j \ \text{ subject to } \ Ax = b, x > 0$$

To find μ-center one had to solve some quadratic constraints, thus the algorithm discussed in class was using full Newton steps. In this way we get approximate μ-center. In order to asses the run time of algorithm let us take a look at how does our proximity measure, $\sigma(x, s, \mu)$ change with: $x \leftarrow x + \Delta x$, $s \leftarrow s + \Delta s$. We proved the following theorem.

Theorem 1 *If $\sigma(x, s, \mu) < \sigma < 1$, then*

$$\sigma(x + \Delta x, s + \Delta s, \mu) \leq \frac{\sigma^2}{2(1-\sigma)}.$$

Thus, as a corollary we get:

$$\sigma(x, s, \mu) < \frac{2}{3} \Rightarrow \sigma(x + \Delta x, s + \Delta s, \mu) < \sigma(x, s, \mu) < \frac{2}{3},$$

i.e., proximity measure decreases after the Newton step. Therefore, if we continue this process, we will converge to the μ-center.

However, in order to find the solution of the linear program, we need to find the μ-center for $\mu = 0$. Therefore, after each step, we also change the value of μ: $\mu \leftarrow \mu(1-\theta)$. We showed the following:

Theorem 2 *If $\sigma(x, s, \mu) = \sigma$ and $x^T s = n\mu$, then*

$$\sigma(x, s, \mu(1-\theta)) = \frac{\sqrt{\sigma^2 + n\theta^2}}{(1-\theta)}.$$

Therefore, the algorithm is as follows:

- Initialization: Start with $(x^{(0)}, s^{(0)}, \mu^{(0)})$ such that $\sigma(x^{(0)}, s^{(0)}, \mu^{(0)}) \leq \frac{1}{2}$. Refer to the handout to see in more detail how we can do this. The point is that we can choose this starting point in such a way that $\mu^{(0)} \leq 2^{6L}$, where L is the length of the program.
- While $\mu \geq \epsilon$ do:
  - Find $\Delta x, \Delta s$ by solving the Newton step equations.

– Let $x \leftarrow x + \Delta x$, $s \leftarrow s + \Delta s$, $\mu \leftarrow \mu(1-\theta)$ where $\theta = \frac{1}{4\sqrt{n}}$.

We can prove the following.

Theorem 3 *In the above algorithm, if before the Newton step the centrality measure σ is at most $\frac{1}{2}$, then after the Newton step and changing μ, we still have $\sigma < \frac{1}{2}$.*

Proof: If $\sigma(x, s, \mu) \leq \frac{1}{2}$ then by Theorem 1:

$$\sigma(x + \Delta x, s + \Delta s, \mu) \leq \frac{(1/2)^2}{2(1 - 1/2)} = \frac{1}{4}.$$

Now let us change $\mu \leftarrow \mu(1-\theta)$, where $\theta = \frac{1}{4\sqrt{n}}$. Using Theorem 2 we get $\sigma(x+\Delta x, s+\Delta s, \mu(1-\theta)) \leq \frac{\sqrt{1/16+1/16}}{1-\frac{1}{4\sqrt{n}}} \leq \frac{4}{3}\frac{\sqrt{2}}{4} < \frac{1}{2}$. $\square$

What is the number of iterations that we have to make? Suppose we start at $\mu^{(0)} = 2^{6L}$. After k iterations we have $\mu \leq (1-\theta)^k 2^{6L}$. Therefore, in order to have $\mu \leq \epsilon$, we need $k = \Theta(\sqrt{n}(L + \ln \frac{1}{\epsilon}))$.

How do we choose ϵ? We know that at any point because of duality gap: $x^T s = n\mu$. Given that we stop when $\mu < \epsilon$ then we have $x^T s < n\epsilon$, i.e., the duality gap is smaller than $n\epsilon$ at the end of the algorithm. The following theorem allows us to stop as soon as duality gap is smaller than $\frac{1}{2^{2L}}$ and take any vertex below as the solution. Suppose that when we stop there are two vertices of the original linear program for which the above duality gap condition holds. These two vertices, $x^{(1)}$, $x^{(2)}$, both satisfy $Ax = b, x \geq 0$.

Theorem 4 *If $c^T x^{(1)} \neq c^T x^{(2)}$ then*

$$|c^T x^{(1)} - c^T x^{(2)}| \geq \frac{1}{2^{2L}}$$

where L is the size of the initial linear program.

Proof: We have that A is a matrix with integer entries and b is a vector with integer entries. Then we can express:

$$x^{(1)} = (\frac{p_1}{q}, \frac{p_2}{q}, ..., \frac{p_n}{q})^T \text{ and } x^{(2)} = (\frac{r_1}{s}, \frac{r_2}{s}, ..., \frac{r_n}{s}),$$

$$\forall i, j: \; q, s, p_i, q_j \in \mathbb{Z} \text{ and } q, s, p_i, q_j \leq 2^L$$

by the theorem we had in a previous lecture.

Assume c is a vector with integer entries as well. Then:

$$|c^T x^{(1)} - c^T x^{(2)}| = |\frac{integer}{q} - \frac{integer}{s}| = |\frac{integer}{qs}| > \frac{1}{qs} > \frac{1}{2^{2L}}.$$

$\square$

Now choose ϵ so that $n\epsilon < 2^{-2L}$. This implies $k = \Theta(\sqrt{n}L)$. As the number of steps in the ellipsoid algorithm was $\Theta(n^2 L)$ we can see that interior algorithm has much smaller number of iterations. Also, in ellipsoid algorithm we were reducing the volume of the ellipsoid by a small factor. Here there are tons of small tricks to reduce μ well and to start with μ that is not so big. One could also do

$x \leftarrow x + 2\Delta x, s \leftarrow s + 2\Delta s$ to make it more appealing in practice. Often in actual implementations the number of iterations is independent of the size of the program.

By choosing ϵ so that $n\epsilon < 2^{-2L}$, we know that every vertex below the current solution is an optimum solution of the linear program. But How do we find such a vertex? In previous lectures we had a Lemma saying that for every x in a polyhedron P, there is always a vertex of P below it of value less than it. In fact, the proof of this lemma provides an algorithm for finding such a vertex.

At this moment we stop with linear programming to go on to network flows. To recapitulate we have seen two main classes of linear programming algorithms:

- Ellipsoid: has the advantage that it only requires us to provide a separation subroutine, rather than listing all the constraints of the linear program.
- Interior Point Method: has the advantage that it is much faster than the ellipsoid algorithm. Variants of this algorithm are actually used in practice.

A major open question in linear programming is if there exists a strongly polynomial time linear programming algorithm, i.e., one whose running time does not depend on the size of numbers but only on dimensions n, m.

Network Flows

A network flow problem that we will consider in more detail is: Given network (graph) and quantity of flow allowed on each edge (capacity) and cost, our goal is to find a flow of minimum cost that satisfies some circulation properties (to be defined later). This problem can be seen as a linear programming problem. However, we will use the structure of this problem to build faster algorithms and will be able to give a strongly polynomial time algorithm that solves this problem (i.e., its running time will depend only on the number of edges and vertices in the graph, and not on the numerical values in the input).

Problems Being Considered in Network Flows

There are many problems that can be considered as Network Flow problems. Here's a list of some of them:

- Shortest path problem
- Bipartite matching problem
- Maximum flow problem
- Minimum-cost flow problem: This problem is more general than the above problems, there exist a strongly polynomial algorithm for it and we can also reduce other problems to this one. This is the very reason we will focus on this one.
- Multicommodity flow problem

Notations

We will use directed graph $G = (V, E)$ to represent a network. We will use the Goldberg-Tarjan notation. Description of our network model follows:

- We are given a bi-directed graph $G = (V, E)$, i.e., for every $v, w \in V$,
$$(v, w) \in E \Rightarrow (w, v) \in E.$$
- On every edge $(v, w) \in E$ there is a capacity $u(v, w)$. $(u : E \to \mathbb{Z})$
- On every edge $(v, w) \in E$ there is a cost $c(v, w)$ that is anti-symmetric:
$$c : E \to \mathbb{Z}, \quad \forall (v, w) \in E : \; c(v, w) = -c(w, v).$$

Definition 1 *A* flow *is a function $f : E \to \mathbb{R}$ that satisfies*

1. *anti-symmetry: $\forall (v, w) \in E : f(v, w) = -f(w, v)$, and*
2. *capacity constraint: $\forall (v, w) \in E : f(v, w) \le u(v, w)$.*

Definition 2 *A* circulation *is a flow f, for which nothing is lost in the network, i.e., net flow into every vertex is 0. More formally,*

$$\forall v \in V : \sum_{w} f(v, w) = 0.$$

Now the Minimum Cost Circulation problem is to find a circulation which minimizes:

$$c^T x = \sum_{(v,w) \in E} c(v, w) f(v, w).$$

Note that here because of anti-symmetric costs and flows we are counting cost of every edge (if we see them in an undirected way) exactly twice.

First as exercise convince yourself that this is a linear program. Therefore we can apply linear programming algorithms we have seen so far. Now let's discuss reductions from other network flow problems to this problem.

Maximum Flow Problem

Given a directed graph $G = (V, E)$, two vertices $s, t \in V$, capacity $u : E \to \mathbb{N}$, find a flow $f : E \mapsto \mathbb{Z}$ that satisfies

$$\forall (v, w) \in E : 0 \le f(v, w) \le u(v, w) \tag{1}$$

$$\forall v \ne s, t : \sum_{w:(w,v) \in E} f(w, v) - \sum_{w:(v,w) \in E} f(v, w) = 0 \tag{2}$$

and maximizes $\sum_{v:(s,v) \in E} f(s, v)$. One can prove that maximum flow is equal to the capacity of the smallest cut (This is known as the Max-Flow-Min-Cut theorem).

The maximum flow problem is a special case of the minimum cost circulation problem. Indeed, given an instance of the maximum flow problem, add an edge between s and t and define $u(t,s) = \infty$, $u(s,t) = 0$, $c(t,s) = -1 = -c(s,t)$ and $c(v,w) = 0$ for all $(v,w) \neq (s,t)$.

The capacities on the bidirected edge (s,t) is such that $f(t,s) \geq 0$, implying that the flow goes from t to s. There is a one-to-one correspondence between circulations in this extended graph and flows in the original graph satisfying all flow conservation constraints in $V \setminus \{s,t\}$. Moreover, the cost of any circulation in this extended graph is exactly equal to minus the net flow out of s (or into t) in the original graph. As a result, the maximum flow problem in G is equivalent to the minimum cost circulation problem in the extended graph.

The following theorem is known as the integrality theorem.

Theorem 5 *If all capacities are integers ($u : E \mapsto \mathbb{Z}$), then there is an optimum flow f whose values are all integers ($f : E \mapsto \mathbb{Z}$).*

To prove this theorem we can use algorithmic or linear programming approaches. We will see the first proof later on and just to note that main idea for the second one is that one could argue that vertices of linear program induced by this problem have only integer components.

Bipartite Matching problem

Given an undirected bipartite graph $G = (V,E)$ with bipartition $V = A \cup B$, $(A \cap B = \emptyset, E \subseteq A \times B)$, find a maximum cardinality matching, i.e., a set of non-adjacent edges of largest cardinality.

A matching is called perfect iff it covers every vertex exactly once. This problem is a special case of the maximum flow problem and therefore the minimum cost circulation problem. To transform the maximum cardinality bipartite matching problem into a maximum flow problem, we

1. direct all the edges from A to B,
2. add a source vertex s, a sink vertex t,
3. add the edges (s,a) for all vertices $a \in A$ and the edges (b,t) for all vertices $b \in B$ and
4. define the capacity of all existing edges to be 1 and the capacity of their reverse edges to be 0 (in other words, the flow on the existing edges have a lower bound of 0).

By the integrality theorem, we know that the flow on any existing edge can be assumed to be either 0 or 1. Therefore, to any flow f, there corresponds a matching $M = \{(v,w) \in E : f(v,w) = 1\}$ whose cardinality is precisely equal to the net amount of flow out of vertex s.

It is also easy to construct from a matching M a flow of value $|M|$. As a result, any integral flow of maximum value will correspond to a matching of maximum cardinality.

In fact, the minimum weighted bipartite matching problem is also a special case of the minimum cost circulation problem. We can modify the above transformation in the following way. Define the cost of any edge of the original graph to be its original cost and the cost of any new edge to be 0. Now, we can model three versions of the minimum weighted bipartite matching problem by appropriately defining the capacities on the edges (t,s) and (s,t):

1. If $u(t,s) = n$ and $u(s,t) = -n$ where $n = |A| = |B|$, we get the minimum weighted perfect (a *perfect* matching is a matching that covers all the vertices) matching.

2. If $u(t, s) = n$ and $u(s, t) = 0$, we obtain the minimum weighted matching.
3. If $u(t, s) = k$ and $u(s, t) = -k$, we obtain the minimum weighted matching of size k.