RJ 10296 (Log# A0306-022) (06/25/2003)
Computer Science/Mathematics

IBM Research Report

Lecture Notes on Advanced Algorithms Spring 2003

David P. Williamson

IBM Research Division
Almaden Research Center
650 Harry Rd.
San Jose, CA 95120-6099

This page intentionally left blank.

Lecture Notes on Advanced Algorithms

David P. Williamson

Spring 2003

Contents

Preface

The contents of this book are lecture notes from a class taught in the Computer Science Department of Stanford University during the Spring 2003 quarter (CS 361B, Advanced Algorithms). The notes were created via the "scribe" system: each lecture one student was appointed as the scribe for that lecture, and was responsible for turning their notes into a LaTeX document. I then edited the notes, and made copies for the entire class. The students in the class who served as scribes were Zoë Abrams, Abhishek Bapna, Sanders Chong, Chuong Do, Mihaela Enăchescu, Vivek Farias, Charles-Henri Gros, Krishnaram Kenthapadi, Damon Mosk-Aoyama, Shubha Nabar, Sanatan Rai, Paat Rusmevichientong, Dilys Thomas, Sergei Vassilvitskii, Fang Wei, and Jiawei Zhang. Any errors which remain (or were there to begin with!) are, of course, entirely my responsibility.

The lectures were drawn from a variety of sources. Lectures 1 and 2 on linear programming, and Lectures 5-7 on minimum-cost circulations were drawn primarily from Michel Goemans' lecture notes on advanced algorithms. The max flow algorithm in Lecture 4 is from

- Satoru Fujishige, "A maximum flow algorithm using MA ordering," to appear in *Operations Research Letters.*

The lectures on generalized flow in Lectures 11 and 12 were mostly drawn from

- Éva Tardos and Kevin Wayne, "Simple generalized maximum flow algorithms," in *Integer Programming and Combinatorial Optimization*, Lecture Notes in Computer Science, vol. 1412, pp. 310–324, Springer, 1998.
- Kevin Wayne, *Generalized Maximum Flow Algorithms*, Ph.D. dissertation, Cornell University, 1999.

Lecture 13 on multicommodity flow was drawn from the two papers

- Naveen Garg and Jochen Könemann, "Faster and simpler algorithms for multicommodity flow and other fractional packing problems." In the *Proceedings of the 39th Annual IEEE Computer Society Conference on Foundations of Computer Science*, 1998.
- Lisa Fleischer, "Approximating fractional multicommodity flows independent of the number of commodities," *SIAM J. Discrete Math.* 13:505-520, 2000.

Lectures 14 and 15 on market equilibria was drawn from slides of Vijay Vazirani and

- Nikhil R. Devanur, Christos H. Papadimitriou, Amin Saberi, and Vijay V. Vazirani, "Market equilibrium via a primal-dual-type algorithm." In the *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, 389–395, 2002.

Lectures 16 and 17 on interior-point methods were drawn primarily from the book

- Stephen J. Wright, *Primal-Dual Interior-Point Methods*, SIAM, 1997.

David P. Williamson
San Jose, CA

CS 361B Advanced Algorithms April 1, 2003

Lecture 1

Lecturer: David P. Williamson *Scribe: Jiawei Zhang*

1.1 Course overview

The course will cover algorithms for problems in combinatorial optimization. Combinatorial optimization is used fundamentally in making decisions that have discrete choices: should we build a facility here or there? Which person should be assigned this task? Which facility should service this client? What paths should be used for transporting these goods?

In this class we will focus on several things:

- **Efficient algorithms in combinatorial optimization**. That is, algorithms that run in polynomial time. Many problems in combinatorial optimization are NP-hard and thus not solvable in polynomial time (at least, not to our knowledge). However, many problems are solvable in polynomial time. We will look mostly at these problems and algorithms, and a little at efficient algorithms for approximating NP-hard problems.
- **Central role of linear programming and duality.** LP is one of the fundamental tools in combinatorial algorithms and in thinking about these problems – both in modelling the problems and algorithms for them. We will focus heavily on LP – its structure, use, and a little on algorithms for its solution.
- **Problems/algorithms that are either a fundamental piece of background, a useful technique for solving other problems, or an interesting open research direction.**

1.2 An introduction to linear programming

A linear programming (LP) problem is an optimization problem that maximizes or minimizes a linear objective function of the decision variables $x_1, x_2, \cdots, x_n$, subject to some linear constraints on x_j's. The standard form of an LP problem is the following:

$$\begin{array}{lll} \min & \sum_{j=1}^{n} c_j x_j & \text{(objective function)} \\ \text{s.t.} & \sum_{j=1}^{n} a_{ij} x_j = b_i \quad i = 1, 2, \cdots, m & \text{(constraints)} \\ & x_j \geq 0 \quad j = 1, 2, \cdots, n & \text{(nonnegativity constraints)} \end{array}$$

where c_j, a_{ij}, b_i are given for $i = 1, 2, \cdots, m$ and $j = 1, 2, \cdots, n$. In the mathematical programming literature, it is often expressed using matrices:

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & Ax = b \\ & x \geq 0 \end{array}$$

where

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \in \Re^{n \times 1}, \quad b = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix} \in \Re^{m \times 1}, \quad c = \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix} \in \Re^{n \times 1},$$

and

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \vdots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix} \in \Re^{m \times n}.$$

Definition 1.1 If x satisfies constraints $Ax = b$ and $x \geq 0$, then x is feasible.

Definition 1.2 An LP is feasible if there exists a feasible x; otherwise the LP is infeasible.

Definition 1.3 x^* is an optimal solution if x^* is feasible and

$$c^T x^* = \min\{c^T x : Ax = b, x \geq 0\}.$$

Definition 1.4 An LP is unbounded if there exists $d \in \Re$, such that for any $z \leq d$, there exists a feasible x such that $c^T x \leq z$.

It is simple to transform a linear program in general form to standard form. Two forms are equivalent in the same sense that they have the same set of optimal solutions or both are unbounded or infeasible.

1. A maximization problem is equivalent to a minimization problem:

$$\max \; c^T x \Leftrightarrow \min \; -c^T x$$

2. An equality constraint can be represented by a pair of inequality constraints:

$$a_i^T x = b_i \Leftrightarrow a_i^T x \geq b_i \text{ and } a_i^T x \leq b_i$$

3. An inequality constraint can be represented by an equality constraint by adding a slack variable:

$$a_i^T x \leq b_i \Leftrightarrow a_i^T x + s_i = b_i,\ \ s_i \geq 0$$

4. If a variable x_j is unrestricted, we can replace it with $x_j^+ - x_j^-$ and add non-negativity constraints $x_j^+ \geq 0$, $x_j^- \geq 0$.

The canonical form of an LP is the following:

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & Ax \geq b \end{array}$$

Consider the following linear program in the canonical form

$$\begin{array}{ll} \min & x_2 \\ \text{s.t.} & x_1 \geq 2 \\ & 3x_1 - x_2 \geq 0 \\ & x_1 + x_2 \geq 6 \\ & -x_1 + 2x_2 \geq 0 \end{array}$$

The optimal solution is $x^* = \begin{pmatrix} 4 \\ 2 \end{pmatrix}$ and the optimal value is 2 (see Figure 1.2). Notice that the optimal solution is obtained at a 'corner' of the feasible region. This is true in general if the LP is bounded. A formal statement will be presented in the next section.

If the first constraint $x_1 \geq 2$ is replaced by $x_1 \leq 2$ and the constraint $-x_1+2x_2 \geq 0$ replaced by $-x1+2x_2 \leq 0$, the problem is *infeasible*. If the original problem is a maximization problem rather than a minimization problem, the problem is unbounded.

1.2.1 The geometry of LP

Let $P = \{x : Ax = b, x \geq 0\}$.

Definition 1.5 $x \in P$ is a vertex of P if $\forall y \neq 0$, either $x + y \notin P$ or $x - y \notin P$.

Theorem 1.1 If $\min\{c^T x : x \in P\}$ is finite, then $\forall x \in P$, there exists a vertex x' such that $c^T x' \leq c^T x$.

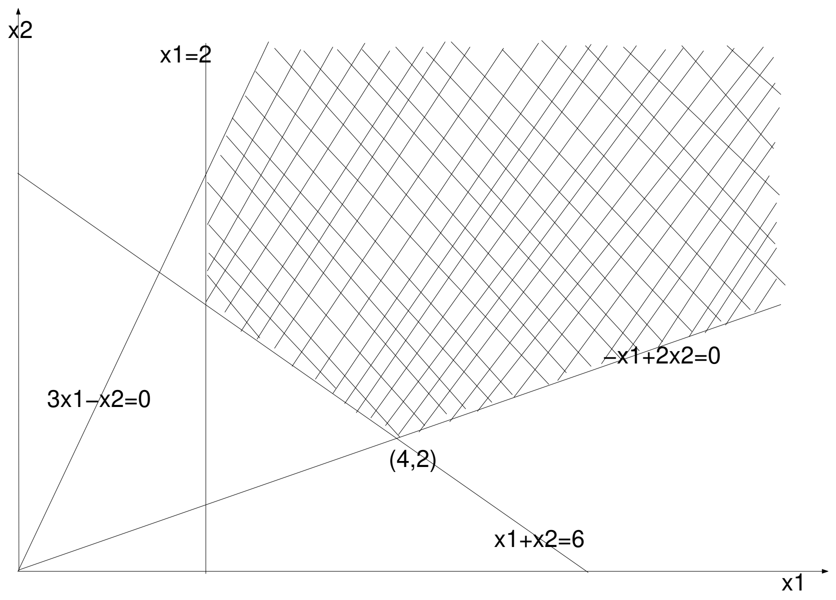


Figure 1.1: The feasible region of an linear program.

Proof: If x is a vertex, then let $x' = x$ and we are done. Otherwise, $\exists y \neq 0$ such that $x + y \in P$ and $x - y \in P$. It follows that $Ay = 0$ since $A(x+y) = b = A(x-y)$. We can assume that $c^T y \leq 0$ (WLOG). For $\lambda > 0$, consider $x + \lambda y$. By assumption,

$$c^T(x+\lambda) = c^T x + \lambda c^T y \leq c^T x.$$

Case 1. $\exists j$ such that $y_j < 0$. (This case is true, WLOG, if $c^T y = 0$.)

Choose

$$\lambda = \min_{j: y_j < 0} \frac{x_j}{-y_j} = \frac{x_k}{-y_k}.$$

It is clear that $\lambda > 0$. Furthermore, $x + \lambda y \geq 0$ and $A(x + \lambda y) = Ax + \lambda Ay = Ax = b$. This implies that $x + \lambda y \in P$ and $(x + \lambda y)_k = 0$. Replace x with $x + \lambda y$ and repeat the process. This process will terminate since $x + \lambda y$ has one more zero component than x.

Case 2. $y_j \geq 0$ for all j. This implies that $c^T y < 0$. In this case, $x + \lambda y \geq 0$ for all $\lambda > 0$ since $x \geq 0$ and $y \geq 0$. Moreover, $A(x + \lambda y) = Ax + \lambda Ay = Ax = b$. Thus $x + \lambda y \in P$. But $c^T(x + \lambda y) = c^T x + \lambda c^T y \to -\infty$ as $\lambda \to \infty$, which implies that the LP is unbounded, a contradiction. □

Corollary 1.2 If $\min\{c^T x : x \in P\}$ is finite, there is an optimal solution x^* that is a vertex.

Theorem 1.3 Let $P = \{x : Ax = b, x \geq 0\}$. For $x \in P$, let A_x be the submatrix of A with columns j such that $x_j > 0$. Then x is a vertex if and only if A_x has linearly independent columns.

For example, if

$$A = \begin{pmatrix} 2 & 1 & 3 & 0 \\ 7 & 3 & 2 & 1 \\ 0 & 0 & 0 & 5 \end{pmatrix}, \quad x = \begin{pmatrix} 2 \\ 0 \\ 1 \\ 0 \end{pmatrix},$$

then

$$A_x = \begin{pmatrix} 2 & 3 \\ 7 & 2 \\ 0 & 0 \end{pmatrix},$$

whose columns are linearly independent. Therefore, x is a vertex.

Proof:

$\Leftarrow$) If x is not a vertex, then $\exists y \neq 0$, such that $x + y \in P$ and $x - y \in P$. Again, it implies that $Ay = 0$. Thus A_y must have dependent columns since $y \neq 0$. On the other hand, $x + y \in P$ and $x - y \in P$ together imply that if $x_j = 0$ then $y_j = 0$ since $x + y \geq 0$ and $x - y \geq 0$. It follows that A_y must be a submatrix of A_x. Therefore, A_x has dependent columns.

$\Rightarrow$) If A_x has linearly dependent columns, then $\exists y' \neq 0$ such that $A_x y' = 0$. Let $y \in \Re^n$ with components $y_j = y'_j$ when $x_j > 0$, and $y_j = 0$ otherwise. Then $Ay = 0$.

Pick $\lambda > 0$ such that $|\lambda y_j| < x_j$ for any j such that $x_j > 0$. Set $y'' = \lambda y$. Then $Ay'' = \lambda Ay = 0$, $x + y'' \geq 0$ and $x - y'' \geq 0$, and thus $x + y'' \in P$, $x - y'' \in P$. By definition, x is not a vertex. This completes the proof. $\square$

Let $B \subseteq \{1, 2, \cdots, n\}$. Let A_B be the submatrix of A that has colunms corresponding to the indices in B. We assume that $A \in \Re^{m \times n}$, $m \leq n$ and rank(A)$= m$.

Definition 1.6 If $|B| = m$ and A_B has linearly independent columns, then B is a basis.

Let $N = \{1, 2, \cdots, n\} - B$. x_B and x_N denote the components of x selected by B and N respectively.

Definition 1.7 Variable x_j s.t. $j \in B$ is called a basic variable; Variable x_j s.t. $j \in N$ is called a non-basic variable.

Suppose that $x_N = 0$; then $x_B = A_B^{-1} b$ (since if B is a basis, then A_B is invertible). It is not necessarily true that $A_B^{-1} b \geq 0$. But if it is the case, i.e., $x_B \geq 0$, then $x \in P$.

Definition 1.8 x is a basic feasible solution if $x_N = 0$ and $x_B = A_B^{-1} b \geq 0$.

Theorem 1.4 There exists a basis B such that x is a basic feasible solution if and only if x is a vertex.

Proof:

$\Leftarrow$) If x is a vertex, set $B = \{j : x_j > 0\}$. By Theorem 1.3, A_B has linearly independent columns.

If $|B| = m$, then we are done. If $|B| < m$, then we can add columns of A to A_B such that A_B has linearly independent columns and $|B| = m$ (This can be done since the rank of A is m.)

$\Rightarrow$) If x is a basic feasible solution with respect to some basis B, then A_B has linearly independent columns, and $x_j = 0$ for $j \notin B$. By definition, x is a vertex. $\square$

Corollary 1.5 If $\min\{c^T x : Ax = b, x \geq 0\}$ is finite, then $\exists B$ such that $x_B = A_B^{-1}b, x_N = 0$ is optimal.

1.2.2 The Simplex algorithm

The simplex algorithm tries to find an optimal solution from basic feasible solutions or vertices. The idea is to start from a basis and check if objective value can be improved or not by moving to an adjacent basis.

Given a current basis B and the corresponding basic feasible solution x, consider

$$\begin{array}{ll} \min & c_B^T x_B + c_N^T x_N \\ \text{s.t.} & A_B x_B + A_N x_N = b \\ & x_B, x_N \geq 0 \end{array}$$

It is clear that

$$x_B = A_B^{-1}(b - A_N x_N) = A_B^{-1}b - A_B^{-1}A_N x_N$$

and the objective function can be expressed as

$$\begin{aligned} c^T x &= c_B^T x_B + c_N^T x_N \\ &= c_B^T(A_B^{-1}b - A_B^{-1}A_N x_N) + c_N^T x_N \\ &= c_B^T A_B^{-1} b - (c_B^T A_B^{-1} A_N + c_N^T) x_N \\ &= c_B^T A_B^{-1} b + (c_N^T - c_B^T A_B^{-1} A_N) x_N \end{aligned}$$

Call $\tilde{c}_N^T = c_N^T - c_B^T A_B^{-1} A_N$ the *reduced costs* of the non-basic variables.

We can then give the following algorithm for solving linear programs. This was the first algorithm given for solving LPs, and is still the most widely used in practice – indeed if there were an Algorithms Hall of Fame, the Simplex method would certainly belong there.

The central idea is quite simple: if there is a non-basic variable whose reduced cost is negative, we can improve the value of the objective function by increasing it from zero. Since x_B is dependent on x_N at some point this increase might cause a basic variable to become zero. It is then obvious that we should update the basis by swapping the non-basic variable for the basic variable that has become zero. This swap is called a *pivot*.

There are many issues about the Simplex method that we will not discuss (e.g. how to find an initial basic feasible solution). The reason is simple: in this class we are discussing efficient algorithms for combinatorial optimization problems, and there is no known polynomial-time version of the Simplex method.

Simplex Algorithm (Dantzig 1947)

```
Given an initial basis $B$ and a basic feasible solution $x$
While $\exists j \in N$, s.t. $\tilde{c}_j < 0$
    increase $x_j$ as long $x_B \geq 0$
    when there exists some $k \in B$ such that $x_k = 0$
        $B \leftarrow B - \{k\} + \{j\}$
        $N \leftarrow N - \{j\} + \{k\}$
```

CS 361B Advanced Algorithms April 3, 2003

Lecture 2

Lecturer: David P. Williamson *Scribe: Krishnaram Kenthapadi*

2.1 An introduction to linear programming (cont.)

2.1.1 The Simplex algorithm (cont.)

The Simplex Algorithm tries to find an optimal solution from basic feasible solutions or vertices. The idea is to start from a basis and check if the objective value can be improved or not by moving to an adjacent basis. Let B be the current basis and $N = \{1, 2, \ldots, n\} \setminus B$. Let x_B denote the basic variables and x_N the non-basic variables.

The central idea of the Simplex Algorithm is quite simple: if there is a non-basic variable whose reduced cost is negative, we can improve the value of the objective function by increasing it from zero. Since x_B is dependent on x_N, at some point, this increase might cause a basic variable to become zero. It is then obvious that we should update the basis by swapping the non-basic variable for the basic variable that has become zero. This swap is called a *pivot*.

Simplex Algorithm (Dantzig 1947)

```
Given an initial basis $B$ and a basic feasible solution $x$
While $\exists j \in N$, s.t. $\tilde{c}_j < 0$
    increase $x_j$ as long $x_B \geq 0$
    when there exists some $k \in B$ such that $x_k = 0$
        $B \leftarrow B - \{k\} + \{j\}$
        $N \leftarrow N - \{j\} + \{k\}$
```

Pivoting rule:
We need a pivoting rule to determine which index should enter the basis and which index should be removed from the basis. For instance, there could be many choices for j and k above (i.e., many $\tilde{c}_j$'s which are negative and once we increase some x_j, simultaneously many x_k's ($k \in B$) become zero). Hence we need to decide which x_j has to be increased amongst those for which $\tilde{c}_j < 0, j \in N$ and also which $x_k = 0, k \in B$ has to be removed from the basis.

Remark 2.1 There is no known pivoting rule for which the number of pivots in the worst case is better than exponential. Hence the Simplex Algorithm (using any of the known pivoting rules) can take exponential time in the worst case. It is an *interesting* **open** problem to find a pivoting rule that requires only polynomially many pivots. A related problem is to analyze the length of the shortest path between any two vertices of a convex polyhedron where the path is along edges.

Remark 2.2 (Kalai) There is a randomized pivoting rule that gives expected subexponential number of pivots.

Remark 2.3 There exists a pivoting rule that gives polynomially many pivots for a randomly perturbed matrix A.

Hirsch Conjecture:
For m constraints (i.e., hyperplanes) in d dimensions, the length of the shortest path between any two vertices of the arrangement is at most $m-d$.

Even if the Hirsch Conjecture were true, it doesn't say much about the number of pivots for the Simplex Algorithm. This is because, in the Simplex method, the path we take is monotonic with respect to the objective function whereas the objective function need not be non-increasing (it doesn't even come into the picture!) along the shortest path.

2.1.2 LP duality

Duality is one of the most important ideas in Linear Programming. It is extremely useful in the design of algorithms for problems in combinatorial optimization.

The concept of duality helps us to lower bound the value of the optimal solution to a linear programming (minimization) problem. For example, consider the following LP:

$$\begin{array}{lrrrl}
\text{Min} & 6x_1 + & 4x_2 + & 2x_3 & \\
\text{s.t.} & 4x_1 + & 2x_2 + & x_3 & \geq 5 \\
 & x_1 & + x_2 & & \geq 3 \\
 & & 2x_2 + & x_3 & \geq 4 \\
 & & & x_i & \geq 0 \;\; i = 1,2,3
\end{array}$$

The first inequality gives a lower bound of 5 on the value of LP, since $6x_1 + 4x_2 + 2x_3 \geq 4x_1 + 2x_2 + x_3 \geq 5$ due to the nonnegativity of the x_i's. By considering the first inequality plus twice the second inequality, we get a better bound: value(LP) $\geq 6x_1 + 4x_2 + x_3 \geq 5 + 2 * 3 = 11$. We get an even better bound by considering four times the second inequality: value(LP) $\geq 4x_1 + 4x_2 \geq 4 * 3 = 12$.

We might think about how to find the best possible lower bound by taking combinations of the rows as above. It turns out that we can express this as a linear program:

$$\begin{array}{lllll}
\text{Max} & 5y_1+ & 3y_2+ & 4y_3 & \\
\text{s.t.} & 4y_1+ & y_2 & & \leq 6 \\
 & 2y_1+ & y_2+ & y_3 & \leq 4 \\
 & y_1\ + & & y_3 & \leq 2 \\
 & & & y_j & \geq 0 \ \ j = 1,2,3
\end{array}$$

Here y_1 is the multiplier for the first constraint (inequality), y_2 for the second and y_3 for the third. y_j's have to be nonnegative so as to preserve the signs of the original inequalities. Suppose the second constraint was an equality: $x_1 + x_2 = 3$. Then y_2 which multiplies this constraint can be negative as well.

In general, let P and D denote the following pair of dual linear programs:

$$\text{(P)} \quad Z_P = \min\{c^T x | Ax = b, x \geq 0\}$$

$$\text{(D)} \quad Z_D = \max\{b^T y | A^T y \leq c\}$$

(P) is called the *primal* linear program and (D) the *dual* linear program. We can also express these LPs without using matrix notation as:

Primal LP (P):

$$\begin{array}{llr}
\min & \sum_{j=1}^{n} c_j x_j & \text{[objective function]} \\
\text{s.t.} & \sum_{j=1}^{n} a_{ij} x_j = b_i \quad i = 1,2,\cdots,m & \text{[constraints]} \\
 & x_j \geq 0 \quad j = 1,2,\cdots,n & \text{[nonnegativity constraints]}
\end{array}$$

Dual LP (D):

$$\begin{array}{llr}
\max & \sum_{i=1}^{m} b_i y_i & \text{[objective function]} \\
\text{s.t.} & \sum_{i=1}^{m} y_i a_{ij} \leq c_j \quad j = 1,2,\cdots,n & \text{[constraints]} \\
 & y_i \quad \text{[unrestricted]} &
\end{array}$$

We can formalize the notion of the dual being a lower bound as follows.

Theorem 2.1 (Weak Duality) If x is a feasible solution to (P) and y is a feasible solution to (D), then $c^T x \geq b^T y$. As a consequence, $Z_P \geq Z_D$.

Proof: $c^T x \geq (A^T y)^T x = y^T A x = y^T b = b^T y$. The first inequality follows from the dual constraint and the second equality follows from the primal constraint. □

As a corollary, we see that only some possibilities of primal/dual feasibility/unboundness are possible.

Corollary 2.2 For any primal LP, (P) and the corresponding dual LP (D), exactly one of the following is true:

1. Both (P) and (D) are feasible and are bounded.
2. (P) is infeasible and (D) is unbounded.
3. (D) is infeasible and (P) is unbounded.
4. Both (P) and (D) are infeasible.

In fact, something even more useful and interesting than weak duality is true. The proof can be either geometric or algorithmic. Due to lack of time, we will not prove this in the class.

Theorem 2.3 (Strong Duality) If (P) and (D) are both feasible, then $Z_P = Z_D$.

Complementary slackness

Definition 2.1 Given a feasible x for (P) and (y, s) for (D), the **duality gap** is defined as $c^T x - b^T y$.

As a consequence of the weak duality theorem, the duality gap is nonnegative. $c^T x - b^T y = c^T x - (Ax)^T y = x^T c - x^T A^T y = x^T (c - A^T y) = x^T s \geq 0$ since $x, s \geq 0$. From the strong duality theorem, it follows that the duality gap is zero for the optimal solutions x^* for (P) and (y^*, s^*) for (D), i.e., $c^T x^* - b^T y^* = Z_P - Z_D = 0$. Thus the duality gap provides a measure of how close a feasible x and y are to the optimal solutions for (P) and (D).

Corollary 2.4 (Complementary Slackness) Let x^* be a feasible solution for (P) and (y^*, s^*) be a feasible solution for (D). Then the following are equivalent:

1. x^* is optimal for (P) and (y^*, s^*) is optimal for (D).
2. $(x^*)^T s^* = 0$.

3. $x_j^* s_j^* = 0 \quad \forall j = 1, 2, \ldots, n.$

4. For any j, if $s_j^* > 0$, then $x_j^* = 0$

5. For any j, if $x_j^* > 0$, then $s_j^* = 0$

Proof:

$\underline{(1) \Rightarrow (2)}$: As a result of the strong duality theorem, the duality gap, $(x^*)^T s^*$ is zero for the optimal solutions x^* for (P) and (y^*, s^*) for (D).

$\underline{(2) \Rightarrow (1)}$: $(x^*)^T s^* = 0 \Leftarrow c^T x^* = b^T y^*$ Since the primal and dual LP values are equal, the corresponding solutions are optimal.

$\underline{(2) \Leftrightarrow (3) \Leftrightarrow (4) \Leftrightarrow (5)}$: Follows from the feasibility and hence non-negativity of x^* and s^*.

$\square$

In algorithms for combinatorial problems, we manipulate the primal and the dual solutions. We know that we have reached the optimal solutions when the complementary slackness conditions are met (typically one of (4) or (5) above). Many algorithms fix two of the following conditions to start with and then try to satisfy the third condition:

1. x is feasible for (P).

2. y is feasible for (D).

3. Complementary slackness conditions hold.

Rules for transforming primal LP to dual LP

Suppose we are given a primal LP with the following structure.

<u>Primal LP Framework</u>:

$$\begin{array}{ll} \min & c_1^T x_1 + c_2^T x_2 + c_3^T x_3 \\ \text{s.t.} & A_{11}x_1 + A_{12}x_2 + A_{13}x_3 = b_1 \ [(1)] \\ & A_{21}x_1 + A_{22}x_2 + A_{23}x_3 \geq b_2 \ [(2)] \\ & A_{31}x_1 + A_{32}x_2 + A_{33}x_3 \leq b_3 \ [(3)] \\ & x_1 \geq 0; x_2 \leq 0; x_3 \text{ unrestricted} \end{array}$$

Here, the variables which are nonnegative are grouped together as x_1, those nonpositive as x_2 and the rest as x_3. $A_{11}, A_{12}, \ldots, A_{33}$ are the corresponding parts of the matrix A. We obtain the corresponding dual below.

Corresponding Dual LP Framework:

$$\begin{array}{rl}
\max & b_1^T y_1 + b_2^T y_2 + b_3^T y_3 \\
\text{s.t.} & A_{11}^T y_1 + A_{21}^T y_2 + A_{31}^T y_3 \le c_1 \text{ [since } x_1 \ge 0] \\
& A_{12}^T y_1 + A_{22}^T y_2 + A_{33}^T y_3 \ge c_2 \text{ [since } x_2 \le 0] \\
& A_{13}^T y_1 + A_{23}^T y_2 + A_{33}^T y_3 = c_3 \text{ [equality since } x_3 \text{ is unrestricted]} \\
& y_1 \text{ [unrestricted, due to equality in (1)]} \\
& y_2 \ge 0 \text{ [due to `` } \ge \text{ '' in (2)]} \\
& y_3 \le 0 \text{ [due to `` } \le \text{ '' in (3)]}
\end{array}$$

2.2 The minimum cut problem

Now that we've reviewed linear programming, we will start our exploration of algorithms for combinatorial problems with one of the most basic problems in combinatorial optimization.

Definition 2.2 Given an undirected graph, $G = (V, E)$ and a nontrivial $S \subset V$, the **cut** induced by S is the set of edges with exactly one endpoint in S.

We let $\delta(S) = \{(i,j) \in E | i \in S, j \in V \setminus S\}$ denote the set of edges in the cut. Given *capacities* $u_e \ge 0$ for all edges $e \in E$, the *capacity of the cut* is the sum of the capacities of edges in the cut: $u(\delta(S)) = \sum_{e \in \delta(S)} u_e$.

Mincut

- **Input:**
  - Undirected graph, $G = (V, E)$
  - Edge capacities, $u_e \ge 0 \ \forall e \in E$
- **Goal:** Find $S \subset V, S \ne \emptyset$ that minimizes the capacity of the cut, $u(\delta(S))$

A slight variant is the following problem:

Min s–t cut

- **Input:**
  - Undirected graph, $G = (V, E)$
  - Edge capacities, $u_e \geq 0 \ \forall e \in E$
  - Two distinguished vertices $s, t \in V, s \neq t$
- **Goal:** Find $S \subset V, S \neq \emptyset$ such that $s \in S, t \notin S$ that minimizes $u(\delta(S))$

The mincut problem can be reduced to the min s–t cut problem as follows. Choose $s = 1$ (first vertex) and obtain the optimum s–t cut for $t = 2, 3, \ldots, n$. The cut with the minimum capacity will correspond to the global mincut.

CS 361B Advanced Algorithms April 8, 2003

Lecture 3

Lecturer: David P. Williamson *Scribe: Abhishek Bapna*

3.1 Minimum cuts and maximum flows

In this section, we discuss how the *min s-t cut* problem defined for directed graphs relates to the *max s-t flow* problem. Recall that for undirected graphs, we had defined the *min s-t cut* problem as follows.

The Min s-t Cut Problem

- **Input:**
  - Undirected Graph $G = (V, E)$.
  - Capacities $u_e \geq 0 \ \forall e \in E$.
  - $s, t \in V, s \neq t$.
- **Goal:** Find a set $S \in V, S \neq \emptyset$, $s \in S, t \notin S$, that minimizes $u(\delta(S))$, where $\delta(S)$ is the the set of edges with exactly on endpoint in S, and $u(\delta(S)) = \sum_{e \in \delta(S)} u_e$.

For a directed graph, we define the cut to be $\delta^+(S) = \{(i,j) \in A, i \in S, j \notin S\}$, where A is the set of arcs. The goal now is to minimize $u(\delta^+(S))$.

When in a given network, the links have a limited capacity, the problem of finding the maximum amount of flow on the network sometimes becomes an important one.This is known as the max-flow problem. Some sample applications use the max-flow problem in answering questions of the form:

- What is the maximum amount of traffic that can be routed on a given network of roads?
- Given a network of pipelines, what is the maximum amount of water that can be pumped?
- A recent development in solving for market equilibrium prices in a market of goods and buyers asks the question: What is the maximum amount of a particular good that can be supplied to consumers getting maximum benefit out of it?

Definition 3.1 Given a directed graph $G = (V, A)$, with integer capacities $u_{ij} \geq 0$ for all $(i, j) \in A$ we define a *s-t flow* $f : A \rightarrow \Re^{\geq 0}$ such that:

(i) $f_{ij} \leq u_{ij}$ (Capacity Constraints).

(ii) $\sum_{k:(i,k)\in A} f(i,k) = \sum_{k:(i,k)\in A} f(k,i)\ \forall i \in V, i \neq s,t$ (Flow Conservation)

Then the *max s-t flow* problem in a directed graph is:

The Max s-t Flow Problem

- **Input:**
  - Directed Graph $G = (V, A)$.
  - Integer capacities $u_{ij} \geq 0\ \forall (i,j) \in A$.
  - $s, t \in V$, $s \neq t$
- **Goal:** Find an s-t flow that maximizes $\sum_{k:(s,k)\in A} f(s,k) - \sum_{k:(k,s)\in A} f(k,s)$.

Letting $x_{ij} = f(i,j)$, the linear program corresponding to the above $max\ s{-}t\ flow$ problem is

$$\begin{array}{lll} \text{Max} & \displaystyle\sum_{j:(s,j)\in A} x_{sj} - \sum_{j:(j,s)\in A} x_{js} & \\ \text{s.t.} & & \\ & \displaystyle\sum_{i:(i,k)\in A} x_{ik} - \sum_{i:(k,i)\in A} x_{ki} = 0 & \forall k \neq s,t \\ & 0 \leq x_{ij} \leq u_{ij} & \forall (i,j) \in A. \end{array}$$

Introducing dual variables z_{ij} corresponding to the nonnegativity constraints, and y_i corresponding to the flow balance constraints, we can write the dual (call it (MFD)) as

$$\begin{array}{llll} & \text{Min} & \displaystyle\sum_{(i,j)\in A} u_{ij} z_{ij} & \\ & \text{s.t.} & & \\ & & z_{ij} + y_j - y_i \geq 0 & \forall (i,j) \in A, i \neq s,t; j \neq s,t \\ (MFD) & & z_{sj} + y_j \geq 1 & \forall (s,j) \in A \\ & & z_{js} + y_j \geq -1 & \forall (j,s) \in A \\ & & z_{it} - y_i \geq 0 & \forall (i,t) \in A \\ & & z_{ij} \geq 0. & \end{array}$$

Claim 3.1 Let z_D be the optimal value of MFD and let MC refer to the value of the min s-t cut. Then $z_D = MC$.

If the above claim is true, then the following theorem follows from the strong duality theorem of linear programs.

Theorem 3.2 The max s-t flow in any directed graph is equal to the value of the min s-t cut of the graph.

Proof of Claim 3.1:

(i) $z_D \leq MC$.
Let S be the min cut of the graph. Then create a feasible solution (y, z) to (MFD) of value MC as follows.

$$z_{ij} = \begin{cases} 1 & \text{if } i \in S, j \notin S \\ 0 & \text{otherwise} \end{cases} \qquad y_i = \begin{cases} 1 & \text{if } i \in S \\ 0 & \text{otherwise} \end{cases}$$

A quick verification shows that the above is a feasible solution to (MFD) that has an objective function value MC. Therefore $z_D \leq MC$ since z_D is the optimal solution by definition.

(ii) $MC \leq z_D$.
Let (y^*, z^*) be an optimal solution to MFD. Set $y_s^* = 1,\ y_t^* = 0$. Now create a cut S as follows:

- generate a uniform random number, $U \in (0, 1]$.
- if $y_i^* \geq U$, set $i \in S$.

Now the probability of a particular arc (i, j) to be a part of the cut

$$\begin{aligned} \Pr[(i,j) \text{ in cut}] = \Pr[i \in S, j \notin S] &= \begin{cases} y_i^* - y_j^* & \text{if } y_i^* \geq y_j^* \\ 0 & \text{otherwise} \end{cases} \\ &= \max(0, y_i^* - y_j^*) \\ &\leq z_{ij}^* \\ \text{and so, E[value of cut]} &= \textstyle\sum_{(i,j)\in A} u_{ij} \Pr[(i,j) \text{ in cut}] \\ &\leq \textstyle\sum_{(i,j)\in A} u_{ij} z_{ij}^* \\ &= z_D \end{aligned}$$

So there exists some cut with value $\leq z_D \Rightarrow MC \leq z_D$

Thus from (i) and (ii), $z_D = MC$ and the proof is complete. □

3.1.1 Minimum cut via MA orderings

We now return to undirected graphs. Given an undirected graph $G = (V, E)$, let $\delta(A, B) = \{(i, j) \in E, i \in A, j \in B\}$.

Now, suppose we order the nodes of the graph through the following computation:

- Choose an arbitrary vertex, v_1 and set $S \leftarrow \{v_1\}$
- For $i \leftarrow 2$ to n

```
        Choose $v_i$ that maximizes $u(\delta(S, v_i))$      $\forall v_i \in V - S$
        $S \leftarrow S \cup \{v_i\}$
```

Then, the sequence of vertices $v_1, v_2, ..., v_n$ is called an *MA Ordering* of the graph. (MA stands for "maximum adjacency").

We now make a claim that induces an algorithm to compute the min cut in a graph.

Claim 3.3 For MA Ordering $v_1, v_2, ..., v_n$, the cut around the node $\{v_n\}$ is a min $v_{n-1} - v_n$ cut.

The following algorithm shows how this property can be used to compute the min cut of a undirected graph. The proof of the claim will follow the algorithm.

Finding MinCut using MA ordering

```
$MC \leftarrow \infty, S \leftarrow \emptyset$
While $|V| > 1$
    Compute MA ordering $v_1, v_2, ..., v_n$
    If $u(\delta(v_n)) < MC$
        $MC \leftarrow u(\delta(v_n)), S \leftarrow \{v_n\}$
        Contract $v_{n-1}$ and $v_n$ into a single node
Return $S$.
```

An example of the above algorithm is presented at the end.

To prove the claim made above, we need the following lemma.

Lemma 3.4 Let $\lambda(G, s, t)$ denote value of the min s-t cut in G. Then for any three vertices $p, q, r \in V$, $\lambda(G, p, q) \geq \min(\lambda(G, r, q), \lambda(G, p, r))$.

Proof: Let S be the min p-q cut of the graph and $p \in S$. Now suppose $r \in S$. Then, $\lambda(G, p, q) \geq \lambda(G, r, q)$. If $r \notin S$, then $\lambda(G, p, q) \geq \lambda(G, p, r)$. In either case, the result holds. $\square$

Proof of Claim 3.3: We know that by the definition of the min cut $\lambda(G, v_{n-1}, v_n) \leq u(\delta(v_n))$. We need to show that $\lambda(G, v_{n-1}, v_n) \geq u(\delta(v_n))$. We do this through an induction on the number of nodes and edges, $|E| + |V|$.

- The base case, i.e. when either $|E| = 0$ or $|V| = 2$, holds trivially.

- For the inductive case, there are two possibilities

  (i) $(v_{n-1}, v_n) \in E$:
  Let $(v_{n-1}, v_n) = e$, $G' \leftarrow G - e$, $\delta' \leftarrow \delta$. Now, $v_1, v_2, ..., v_n$ is still an MA ordering of G', and

$$\begin{aligned} u(\delta(v_n)) &= u(\delta'(v_n)) + u_e \\ &= \lambda(G', v_{n-1}, v_n) + u_e \\ &= \lambda(G, v_{n-1}, v_n). \end{aligned}$$

  (ii) $(v_{n-1}, v_n) \notin E$:
  In this case, we need to apply the inductive hypothesis twice. First, let $G' \leftarrow G - v_{n-1}$. Note that $v_1, v_2, \ldots, v_{n-2}, v_n$ is an MA ordering in G', and by the inductive hypothesis,

$$\begin{aligned} u(\delta(v_n)) &= u(\delta'(v_n)) \\ &= \lambda(G', v_{n-2}, v_n) \\ &\leq \lambda(G, v_{n-2}, v_n). \end{aligned}$$

  The last inequality follows since the cut in G separating v_{n-2} and v_n has no greater value in G'.

  Now, let $G' \leftarrow G - v_n$. Again, $v_1, v_2, \ldots, v_{n-1}$ is an MA ordering in G', and by the construction of the ordering, and the inductive hypothesis,

$$\begin{aligned} u(\delta(v_n)) &\leq u(\delta(v_{n-1})) \\ &= u(\delta'(v_{n-1})) \\ &= \lambda(G', v_{n-2}, v_{n-1}) \\ &\leq \lambda(G, v_{n-2}, v_{n-1}). \end{aligned}$$

  Again, the last inequality follows since the cut in G separating v_{n-2} and v_{n-1} has no greater value in G'.

  Now using Lemma 3.4,

$$\lambda(G, v_{n-1}, v_n) \geq \min(\lambda(G, v_{n-2}, v_{n-1}), \lambda(G, v_{n-2}, v_n)) \geq u(\delta(v_n)).$$

  Therefore by the principle of mathematical induction, $\lambda(G, v_{n-1}, v_n) \geq u(\delta(v_n))$ holds for any number of vertices and edges. This proves the claim.

$\square$

A note: Using Fibonacci heaps, an MA ordering of a graph can be computed in $O(m + n \log n)$ time. Thus, the algorithm to compute the MinCut using MA orderings presented above has a time complexity of $O(n(m + n \log n))$. The fastest known algorithm runs in $O(m \log^3 n)$ randomized time.

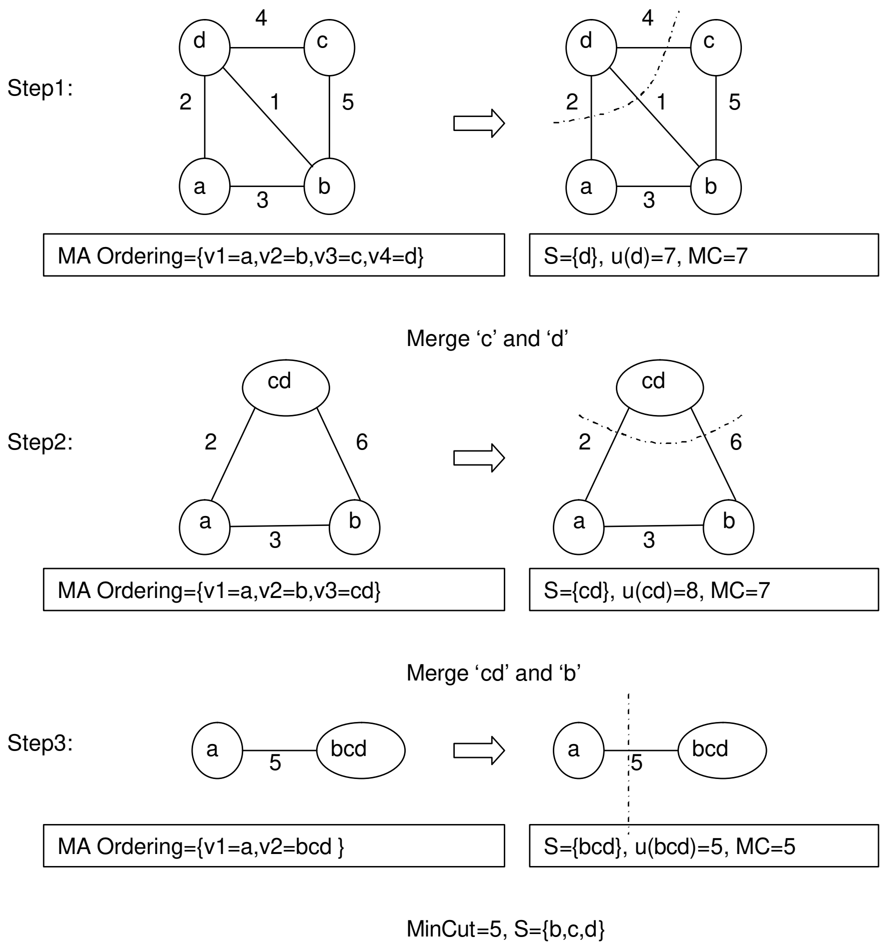


Figure 3.1: An Example of the MinCut Algorithm via MA orderings.

CS 361B Advanced Algorithms	April 10, 2003

Lecture 4

Lecturer: David P. Williamson — *Scribe: Fang Wei*

4.1 Minimum cuts and maximum flows (cont.)

Recall from the previous lecture we showed that the maximum s-t flow problem could be formulated as the following linear program:

$$
\begin{aligned}
\text{Max} \quad & \sum_{j:(s,j)\in A} x_{sj} - \sum_{j:(j,s)\in A} x_{js} \\
\text{subject to:} \quad & \\
& \sum_{k:(i,k)\in A} x_{ik} - \sum_{k:(k,i)\in A} x_{ki} = 0 & \forall i \neq s,t \\
& x_{ij} \leq u_{ij} & (i,j) \in A,
\end{aligned}
$$

where u_{ij} is the capacity of arc (i,j). We assume that the capacities are non-negative and integral.

We showed that the dual of the linear program is as follows:

$$
\begin{aligned}
\text{Min} \quad & \sum_{(i,j)\in A} u_{ij} z_{ij} \\
\text{subject to:} \quad & \\
& z_{ij} + y_j - y_i \geq 0 & \forall (i,j) \in A, i,j \neq s,t \\
& z_{sj} + y_j \geq 1 & \forall (s,j) \in A \\
& z_{it} - y_i \geq 0 & \forall (i,t) \in A \\
& z_{ij} \geq 0
\end{aligned}
$$

We also showed that the dual gives the minimum s-t cut problem.

4.1.1 MA orderings

Recall that we defined the concept of an MA ordering of an undirected graph, and showed that this could be used to find the minimum global cut. Here we extend the concept of an MA ordering to a directed graph. First, let

$$\delta(C,D) = \{(i,j) \in A : i \in C, j \in D\}.$$

We say that for $B \subseteq A$, $u(B) = \sum_{(i,j)\in A} u_{ij}$. Then we can define a directed MA ordering of a capacitated directed graph as follows.

MA-Ordering

```
$S \leftarrow \{v_1\}$
for $i \leftarrow 2$ to $n$
    Choose $v_i$ to maximize $u(\delta(S, \{v\}))$ $\forall v \in V - S$
    $S \leftarrow S \cup \{v_i\}$.
```

4.1.2 Residual graphs

To give an algorithm computing a maximum s-t flow, we first need the concept of a residual graph.

Definition 4.1 Given a flow f, the *residual graph* $G_f = (V, A_f, u^f)$ where

$$A_f = \{(i,j) \in A : f(i,j) < u_{ij}\} \cup \{(j,i) : (i,j) \in A, f(i,j) > 0\}$$
$$u_{ij}^f = \begin{cases} u_{ij} - f(i,j) & \text{if } (i,j) \in A, (i,j) \in A_f \\ f(i,j) & \text{if } (j,i) \in A, (i,j) \in A_f \end{cases}$$

Note that if an arc $(i,j) \in A_f$ then it has *residual capacity* $u_{ij}^f > 0$.

We now show one of the oldest and most celebrated theorems in combinatorial optimization

Theorem 4.1 (Ford, Fulkerson 1955) f is a maximum $s-t$ flow $\Longleftrightarrow$ there is no augmenting path in G_f.

Proof: Suppose there exists an augmenting path $s = i_0, ..., i_k = t$, Let

$$\delta \leftarrow \min_{j=0,...k-1} u_{i_j, i_{j+1}}^f,$$
$$f'(i_j, i_{j+1}) \leftarrow \begin{cases} f(i_j, i_{j+1}) + \delta & \text{if } (i_j, i_{j+1}) \in A \\ f(i_j, i_{j+1}) - \delta & \text{o.w.} \end{cases}$$

f' is a valid flow by construction. Furthermore, the flow on f' out of source is greater by δ.

To prove that if no augmenting path exists that we must have a maximum flow, we look at the dual and use complementary slackness. Let S be the set of vertices reachable from s in the residual graph G_f. Note that since there is no augmenting path $t \notin S$. Construct the following dual solution:

$$y_i \leftarrow \begin{cases} 1 & \text{if } i \in S \\ 0 & \text{o.w.} \end{cases}$$
$$z_{ij} \leftarrow \begin{cases} 1 & \text{if } i \in S, j \notin S \\ 0 & \text{o.w.} \end{cases}$$

By arguments from the previous lecture, one can show that this dual solution is feasible. We now need to show that complementary slackness holds. To do this, we need to show two sets of conditions. First, we want to show that if $z_{ij} > 0$ then $x_{ij} = u_{ij}$. Second, we want to show that if $x_{ij} > 0$ then the corresponding dual constraint is met with equality. To see the first case, note that if $z_{ij} = 1$, then $(i, j) \notin A_f$, which implies that $f(i, j) = u_{ij}$.

To show the second type of complementary slackness condition, we consider (as an example) just the case in which $i, j \neq s, t$. Thus we need to show that if $x_{ij} > 0$ then $z_{ij} + y_j - y_i = 0$. We observe that if both $i, j \in S$, we have $0 + 1 - 1 = 0$. If both $i, j \notin S$, we have $0 + 0 - 0 = 0$. If $i \in S, j \notin S$, we have $1 + 0 - 1 = 0$. Finally, if $j \in S, i \notin S$, $x_{ij} > 0$ implies that (j, i) is in the residual graph, which contradicts the definition of S.

Therefore we have a primal feasible solution $x = f$, and dual feasible solution (y, z) and complementary slackness holds, so that the flow must in fact be optimum.

□

4.1.3 A maximum s-t flow algorithm via MA orderings

The algorithm

There are many, many known algorithms for maximum s-t flows. Here we give a very recent one due to Fujishige, which uses the concept of an MA ordering.

Max Flows via MA Ordering (Fujishige 2002)

```
$f(i,j) \leftarrow 0 \ \ \forall (i,j) \in A$
while $\exists$ an augmenting path in $G_f$
    Compute MA-ordering in $G_f$ starting at $v_1 = s$.
    Let $k$ be such that $t = v_k, V_j = \{v_1, ..., v_j\}$.
    $\alpha \leftarrow \min_{j=2,\dots,k} u^f(\delta(V_{j-1}, v_j))$.
    $\beta(t) \leftarrow \alpha, \beta(v) \leftarrow 0 \ \ \forall v \neq t$
    For $j \leftarrow t$ down to 2
        For each $(v_i, v_j) \in \delta(V_{j-1}, v_j)$  $(Note : i < j)$
        $\gamma \leftarrow \min(\beta(v_j), u^f_{v_i,v_j})$.
          $f'(v_i, v_j) \leftarrow \begin{cases} f(v_i, v_j) + \delta & \text{if } (v_i, v_j) \in A. \\ f(v_i, v_j) - \delta & \text{if } (v_j, v_i) \in A. \end{cases}$
        $\beta(v_i) \leftarrow \beta(v_i) + \gamma$.
        $\beta(v_j) \leftarrow \beta(v_j) - \gamma$.
    $f \leftarrow f'$.
```

We need to argue that this algorithm produces a feasible flow. Consider what happens in the "down to" loop when we consider vertex v_j. By definition of α we know that at least α units of residual capacity enter v_j. We have used $\beta(v_j)$ capacity going out so far. We maintain inductively that $\sum_l \beta(v_l) = \alpha$, so we know that $\beta(v_j) \leq \alpha$; that is, we can ensure that the total flow entering v_j is equal to the total flow leaving it. Hence f' is a feasible flow.

Running time analysis

We now want to show that the algorithm runs in polynomial time. Let n be the number of vertices in the graph, $v(f)$ be the value of the flow f, and f^* be a max flow.

Theorem 4.2 $v(f^*) - v(f) \leq (1 - \frac{1}{n})[v(f^*) - v(f)]$.

We now argue that this theorem is enough to show that the algorithm runs in polynomial time. Let $f^{(n)}$ = flow after n iterations of the main loop. Thus the Theorem implies that

$$\begin{aligned} v(f^*) - v(f^{(n)}) &\leq \left(1 - \frac{1}{n}\right)^n [v(f^*) - v(f)] \\ &\leq \frac{1}{e}[v(f^*) - v(f)], \end{aligned}$$

using the fact that $\left(1 - \frac{1}{n}\right)^n < e^{-1}$.

Let $U = \max_{i,j} u_{ij}$. Then $v(f^*) \leq mU$, where m is the number of arcs in the graph. Hence after at most $n \log(mU)$ iterations,

$$v(f^*) - v(f) < 1.$$

This is enough since capacities are integers and thus the amount by which the value of a flow increases is also an integer. So if $v(f^*) - v(f) < 1$ it must be the case that $v(f^*) - v(f) = 0$, and f is max flow.

Because we can compute an MA ordering in $O(m + n \log n)$ time, the overall runtime is $O((m + n \log n) n \log(mU))$. The current fastest max flow algorithm is due to Goldberg and Rao, and has running time

$$O(min[n^{\frac{2}{3}}, \sqrt{m}] m \log(\frac{n^2}{m}) \log U).$$

We now prove the theorem.

Proof of Theorem 4.2: We need to show

$$v(f^*) - v(f^{(n)}) \leq \left(1 - \frac{1}{n}\right)^n [v(f^*) - v(f)]$$

.

Suppose $\alpha = u^f(\delta(V_{i-1}, v_i))$. Then

$$\sum_{j \in V_{i-1}, k \in V - V_{i-1}} u_{kj}^f = \sum_{k \geq i} u^f(\delta(V_{k-1}, v_k)) \leq n\alpha.$$

The value α is the value of a cut in the residual graph, which must be at least the value of the minimum cut in the residual graph. By an extension of the theorem we proved above, the value of the minimum cut in the residual graph is at least $v(f^*) - v(f)$. Since we increase the value of the flow by α, we know $v(f') - v(f) \geq \alpha$. Thus $v(f^*) - v(f) \leq n[v(f') - v(f)]$

$$\Rightarrow nv(f^*) - nv(f') \leq (n-1)v(f^*) - (n-1)v(f)$$

$$\Rightarrow v(f^*) - v(f') \leq \left(1 - \frac{1}{n}\right)[v(f^*) - v(f)].$$

$\square$

CS 361B Advanced Algorithms April 15, 2003

Lecture 5

Lecturer: David P. Williamson *Scribe: Sanatan Rai*

5.1 Minimum-cost circulations

Last time we considered the problem of finding a maximum flow in a directed graph. For the next few lectures we will consider the next more complicated variant of this problem, in which there are costs c_{ij} associated with each arc (i, j), and each unit of flow sent on (i, j) incurs the cost c_{ij}.

Minimum-cost circulation problem

- **Input:**
  - A directed graph $G = (V, A)$.
  - Integer costs $c_{ij} \geq 0$, $\forall (i, j) \in A$.
  - Integer capacities $u_{ij} \geq 0$, $\forall (i, j) \in A$.
  - Integer demands $0 \leq l_{ij} \leq u_{ij}$, $\forall (i, j) \in A$.
- **Goal:** Find a minimum-cost circulation.

The goal is to find a flow $f : A \to \Re^{\geq 0}$ that minimizes $\sum_{(i,j)\in A} c_{ij} f_{ij}$ such that

$$\begin{array}{rl} l_{ij} \leq f_{ij} \leq u_{ij}, & \forall (i,j) \in A \\ \sum_{k:(i,k)\in A} f_{ik} - \sum_{k:(k,i)\in A} f_{ki} = 0, & \forall i \in V \end{array}$$

This is related to the min-cost flow problem, where the input is a directed graph $G = (V, A)$ with integer costs $c_{ij} \geq 0$ and integer capacities $u_{ij} \geq 0$ for each edge $(i, j) \in A$. The difference is that there are demands b_i on the vertices $i \in V$, such that the sum of demands over all the vertices is zero: $\sum_{i\in V} b_i = 0$. The goal is to find the flow that minimizes $\sum_{(i,j)\in A} c_{ij} f_{ij}$ such that

$$\begin{array}{rl} 0 \leq f_{ij} \leq u_{ij}, & \forall (i,j) \in A, \\ \sum_{k:(i,k)\in A} f_{ik} - \sum_{k:(k,i)\in A} f_{ki} = b_i, & \forall k \in V. \end{array}$$

Theorem 5.1 These two problems are equivalent.

Proof: We sketch the transformations that convert one problem to the other.

Given a min cost flow, add a node s to the graph. For $i \in V$ such that $b_i > 0$ then attach an arc (i, s) with cost 0, and $l_{is} = u_{is} = b_i$. For $i \in V$ such that $b_i < 0$ we attach an arc (s, i) of cost 0 such that $l_{si} = u_{si} = |b_i|$. Note that given a feasible min-cost flow in the original problem we have a min-cost circulation in the modified problem since the flow coming into each node is equal to the flow going out of each node (including the node s, since $\sum_{i:b_i>0} b_i = \sum_{i:b_i<0} |b_i|$). The reverse is also true – given a min-cost circulation in the modified problem, the flow on the arcs of the original problem is a feasible min-cost flow of the same value.

For the converse, we change variables. Set $f'_{ij} = f_{ij} - l_{ij}$, with $f'_{ij} \geq 0$ and $0 \leq f'_{ij} \leq u_{ij} - l_{ij}$. Set $b_i = \sum_{k:(i,k)\in A} l_{ik} - \sum_{k:(k,i)\in A} l_{ki}$. This provides a direct transformation between the two problems. Given a feasible min-cost circulation f in the original problem, we have a feasible min-cost flow f' in the modified problem of the same cost, and vice versa. □

5.1.1 Finding an initial solution

We use the correspondence to the minimum-cost flow problem to find a feasible circulation. Set $f_{ij} = l_{ij}$, $b_i = \sum_{k:(k,i)\in A} f_{ki} - \sum_{k:(i,k)\in A} f_{ik}$, and $u'_{ij} = u_{ij} - l_{ij}$. We claim that we can find an initial circulation (if one exists) by determining a max flow for the above data. To see this, add a source node s and a sink node t to G to obtain a new graph $G' = (V, A')$ as follows. Add an arc $s \to i$ if $b_i > 0$ with capacity b_i. Add an arc $i \to t$ if $b_i < 0$ with capacity $|b_i|$. Now compute the max flow f' on G'. Then we claim the following.

Lemma 5.2 The s-t flow value of f' in G' is $\sum_{i:b_i>0} b_i$ iff $f' + f$ is a feasible circulation in G.

Proof: We prove only the $\Rightarrow$ part. Since $f' \geq 0$ and $f = l$, $l_{ij} \leq f'_{ij} + f_{ij}$. Moreover, $f'_{ij} \leq u'_{ij}$, so that $f'_{ij} + l_{ij} \leq u'_{ij} + l_{ij} = u_{ij}$. Therefore, $l_{ij} \leq f'_{ij} + f_{ij} \leq u_{ij}$ for each edge $(i, j) \in A$. Finally, by flow conservation we know that

$$\forall i \in V, i \neq s, t : \sum_{k:(k,i)\in A'} f'_{ki} - \sum_{k:(i,k)\in A'} f'_{ik} = 0.$$

We also know that

$$\sum_{k:(k,i)\in A} f_{ki} - \sum_{k:(i,k)\in A} f_{ik} = b_i.$$

Suppose $b_i > 0$. Since the flow out of the source equals $\sum_{i:b_i>0} b_i$, and this means that all the arcs out of s are to capacity, this forces $f'_{si} = b_i$. Thus adding the two equations together and subtracting $f'_{si} = b_i$ from both sides gives

$$\sum_{k:(k,i)\in A} (f'_{ki} + f_{ki}) - \sum_{k:(i,k)\in A} (f'_{ik} + f_{ik}) = 0.$$

The case if $b_i < 0$ is similar. $\square$

Now we make a slight change in notation. Replace each arc by two arcs of opposite orientations. If f_{ij} is the flow in (i, j), then force $f_{ji} = -f_{ij}$. This is called antisymmetry. Also set $u_{ji} = -l_{ij}$. This removes the lower bound constraints, since $f_{ji} = -f_{ij} \leq -l_{ij} = u_{ji}$. We make the costs antisymmetric, too: $c_{ji} = -c_{ij}$. Thus the total cost for the two edges with flow f is $c_{ji}f_{ji} + c_{ij}f_{ij} = 2c_{ij}f_{ij}$. Hence optimising for the total cost for this new graph is the same as optimising for the total cost for the original graph.

5.1.2 Residual graph

Given a flow f on G, define the *residual graph* $G_f = (V, A_f)$ where the new arc set

$$A_f := \{(i, j) \in A : f_{ij} < u_{ij}\}.$$

Note that we are using the new notation here. Impose the upper bounds $u^f_{ij} = u_{ij} - f_{ij}$. Then clearly $u^f_{ij} > 0$ for all $(i, j) \in A_f$.

5.1.3 Potentials

A *potential* is a function $p : V \rightarrow \Re$. Given a potential p, define the *reduced cost* $c^p_{ij} := c_{ij} + p_i - p_j$. Then $c^p_{ji} = -c^p_{ij}$. The potential plays the role of the dual variable. We shall show this formally in an moment. Observe that if Γ is a cycle and $c(\Gamma) := \sum_{(i,j)\in\Gamma} c_{ij}$, and $c^p(\Gamma)$ is defined similarly, then $c^p(\Gamma) = c(\Gamma)$. Define $c \cdot f := \sum_{(i,j)\in A} c_{ij} f_{ij}$. We can then prove the following.

Theorem 5.3 $c \cdot f = c^p \cdot f$.

Proof:

$$\begin{aligned}
c^p \cdot f &= c \cdot f + \sum_{(i,j)\in A} (p_i - p_j) f_{ij} \\
&= c \cdot f + \sum_{i\in V} p_i \big(\sum_{k:(i,k)\in A} f_{ik} - \sum_{k:(k,i)\in A} f_{ki} \big) \\
&= c \cdot f.
\end{aligned}$$

This follows since the term in parentheses is zero because of flow conservation. $\square$

5.1.4 Optimality conditions

We now characterise the minimum-cost circulation.

Theorem 5.4 The following are equivalent:

1. f is a minimal cost flow,
2. there are no negative cost cycles in G_f, and,
3. there exists a potential p such that $c^p_{ij} \geq 0$ for all $(i,j) \in A_f$.

Proof:

$[\neg(2) \Rightarrow \neg(1)]$ Let Γ be a negative cost cycle in A_f. Define

$$\delta = \min_{(i,j)\in\Gamma} u^f_{ij},$$

then $\delta > 0$ and let

$$f'_{ij} = \begin{cases} f_{ij} + \delta, & (i,j) \in \Gamma, \\ f_{ij} - \delta, & (j,i) \in \Gamma, \\ f_{ij}, & \text{otherwise.} \end{cases}$$

Thus, $f'_{ij} = -f'_{ji}$ and f' is a feasible circulation if f is. Also, $f'_{ij} \leq u_{ij}$. Furthermore,

$$c \cdot f' = c \cdot f + 2\delta c(\Gamma) < c \cdot f,$$

since Γ is a negative cost cycle. Therefore, f is not of minimum cost.

Note: In $G_{f'}$, Γ does not exist. This is so because, $f'_{ij} = u_{ij}$ for some $(i,j) \in \Gamma$, and this means that $(i,j) \notin A_{f'}$, and so $\Gamma \not\subseteq A_{f'}$. We say that Γ has been *cancelled.*

$[(2) \Rightarrow (3)]$ Add a node s to G_f, and add arcs of cost 0 from s to each $i \in V$. Then let p_i be the length of the shortest path from s to i using costs c_{ij} as the edge lengths. These paths are well defined since there are no negative-cost cycles. Moreover, by definition, $p_j \leq p_i + c_{ij}$, so that $c^p_{ij} \geq 0$ on A_f.

$[(3) \Rightarrow (1)]$ Let f' be any other circulation. We shall show that $c \cdot f' \geq c \cdot f$, so that f is minimal.

Suppose that (i,j) and (j,i) are both present in A_f. Then by assumption, $c^p_{ij} \geq 0$, and $c^p_{ji} \geq 0$. This implies that in fact, $c^p_{ij} = c^p_{ji} = 0$, since $c^p_{ji} = -c^p_{ij}$.

On the other hand if neither (i,j) nor (j,i) is in A_f, then $f_{ij} = u_{ij}$ and $f_{ji} = u_{ji} = -l_{ij}$. So that, $f_{ij} = u_{ij} = l_{ij}$ and this means that $f'_{ij} = f_{ij}$.

Lastly suppose that $(i,j) \notin A_f$ but $(j,i) \in A_f$. Then $f_{ij} = u_{ij}$ so that $f'_{ij} \leq f_{ij} = u_{ij}$. Therefore, $c^p_{ij} \leq 0$ since $c^p_{ji} \geq 0$.

Using the fact that $c \cdot f = c^p \cdot f$, we have that

$$\sum_{(i,j)\in A} c_{ij}(f'_{ij} - f_{ij}) = \sum_{(i,j)\in A} c^p_{ij}(f'_{ij} - f_{ij}) \geq 0,$$

since for any pair of arcs (i,j) and (j,i) the terms in the sum will be non-negative. Thus $c \cdot f' \geq c \cdot f$, and f must be a circulation of minimum cost. This completes the proof. □

CS 361B Advanced Algorithms April 17, 2003

Lecture 6

Lecturer: David P. Williamson *Scribe: Dilys Thomas*

6.1 Minimum-cost circulations (cont.)

6.1.1 Some applications

In the previous lecture the application of min-cost flows to real world problems was asked for. So here we provide examples from the book by Ahuja, Magnanti, and Orlin.

Production-distribution

A car manufacturer has several manufacturing plants and produces several car models at each plant that it then shipped to geographically dispersed retail centers throughout the country. Each retail center requests a specific number of cars of each model. The firm must determine the production plan of each model at each plant and a shipping pattern that satisfies the demands of each retail center and minimizes the overall cost of production and transportation. There are 4 kinds of nodes *plant nodes*, *plant-model nodes*, *retailer-model nodes*, and *retailer nodes*. *Production arcs* connect plant nodes to plant-model nodes with cost equal to manufacturing cost of the model at the plant. Upper and lower bounds indicate minimum and maximum production of each particular car model at the plants. *Transportation arcs* connect plant-model nodes to retailer-model nodes with shipping costs and capacity bounds imposed by contractual agreements with the shippers. *Demand arcs* connect retailer-model nodes to retailer nodes, have zero cost and lower bounds that equal the demand of the model at that retail center. Refer to Figure 6.1.

The minimum cost flow yields a optimal production and shipping schedule.

Racial balancing of schools

Suppose there are two ethic communities 1 and 2 which must be balanced in schools. Each location i is modeled as two nodes, $l_i^{'}$ and $l_i^{''}$ and each school j is modeled as nodes $s_j^{'}$ and $s_j^{''}$. An arc from $l_i^{'}$ to $s_j^{'}$ denotes the number of students from race 1 assigned at location i to school j, similarly between $l_i^{''}$ and $s_j^{''}$ for race 2. These arcs are uncapacitated and have cost per unit flow the distance between location i and school j. For each j, connect $s_j^{'}$ and $s_j^{''}$ to school node s_j with upper and lower bounds

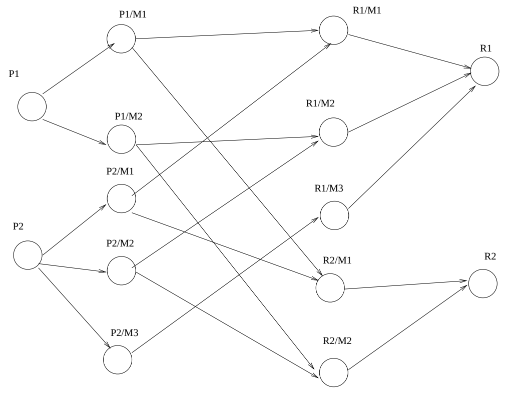


Figure 6.1: Min cost flow for production distribution

on the arcs denoting the bounds on the races needed for school j. Finally add a sink node t, and join each s_j to t with arc of capacity u_j the capacity of school j, so that schools are not overfilled. To ensure everyone goes to school $l_i^{'}$ and $l_i^{''}$ are given supply equal to number of children of ethnic communities 1,2 respectively at location i and the sink t is has demand equal to the sum of all supplies.

6.1.2 Optimality conditions (another proof)

Last time it was proved that if there exists a potential function p such that $c_{ij}^p \geq 0$ for all $(i,j) \in A_f$ then f is a minimum cost flow. We will now reprove this using complementary slackness. First we give the primal and dual formulations.

Primal LP (min-cost circulation):

$$\text{Min} \quad \sum_{(i,j)\in A} c_{ij}x_{ij}$$

subject to:

$$\begin{aligned} &\sum_{k:(k,i)\in A} x_{ki} - \sum_{k:(i,k)\in A} x_{ik} = 0 && \forall i \in V \\ &x_{ij} + x_{ji} = 0 && \forall (i,j) \in A \\ &x_{ij} \leq u_{ij} \end{aligned}$$

Dual:

$$\text{Max} \quad \sum_{(i,j)\in A} u_{ij}z_{ij}$$

subject to:

$$\begin{aligned} &p_j - p_i + w_{ij} + z_{ij} = c_{ij} && \forall (i,j) \in A \\ &z_{ij} \leq 0. \end{aligned}$$

For all $(i,j) \in A$, we can set $w_{ij} = \max(c^p_{ij}, 0)$ and $z_{ij} = \min(c^p_{ij}, 0)$. This solution obeys complementary slackness as $z_{ij} < 0$ implies $c^p_{ij} < 0$ which in turn implies $(i,j) \notin A_f$ which means $x_{ij} = u_{ij}$; that is, the primal constraint is tight. This proves the optimality of the flow.

6.1.3 Klein's cycle cancelling algorithm

The obvious way to improve a circulation is to push flow around a cycle when the residual graph has a negative cycle, so as to decrease the circulation cost.

Cycle cancelling algorithm (Klein 1967)

```
Let f be any circulation
While $A_f$ contains negative cycle $\Gamma$
    Cancel $\Gamma$, update f
```

The problem with the above algorithm is that choosing arbitrary negative cycle Γ does not give a polynomial time algorithm. Also finding the most negative cost cycle is NP-hard. However always selecting certain cycles can ensure the algorithm to run in time polynomial in the input.

Definition 6.1 Let the *mean cost* of a cycle Γ be $\frac{c(\Gamma)}{|\Gamma|}$, where $|\Gamma|$ is the number of arcs in Γ.

Definition 6.2 Let $\mu(f)$ be the minimum mean cost cycle in A_f; that is,

$$\mu(f) = min_{cycles\ \Gamma in A_f} \frac{c(\Gamma)}{|\Gamma|}.$$

6.1.4 Goldberg-Tarjan min-mean cycle cancelling

We can now give the following algorithm.

Min-mean cycle cancelling algorithm (Goldberg & Tarjan 1989)

```
Let f be any circulation
While $\mu(f) < 0$
    Cancel min-mean cycle $\Gamma$, update f
```

Definition 6.3 A circulation f is ϵ-optimal if there exist potentials p s.t. $c^p_{ij} \geq -\epsilon$ for all $(i,j) \in A_f$

Clearly f is 0-optimal iff f is a min cost circulation.

Definition 6.4 Define $\epsilon(f)$ to be the minimum ϵ such that f is ϵ-optimal.

Interestingly, the two values of $\epsilon(f)$ and $\mu(f)$ are closely related.

Theorem 6.1 For a circulation f, $\mu(f) = -\epsilon(f)$.

Proof:
We first show that $\mu(f) \geq -\epsilon(f)$. Since $c^p_{ij} \geq -\epsilon(f)$ for all $(i,j) \in A_f$, $c^p(\Gamma) \geq -\epsilon(f)|\Gamma|$ for all $\Gamma \in A_f$. Thus

$$\mu(f) = \frac{c(\Gamma)}{|\Gamma|} = \frac{c^p(\Gamma)}{|\Gamma|} \geq -\epsilon(f).$$

We now show that $\mu(f) \leq -\epsilon(f)$. Set $\overline{c}_{ij}$= c_{ij} - $\mu(f)$. Then for any cycle Γ in A_f, $\overline{c}(\Gamma) = c(\Gamma) - |\Gamma|\mu(f)$. As $\mu(f) \leq \frac{c(\Gamma)}{|\Gamma|}$, we have $\overline{c}(\Gamma) \geq 0$. We introduce a source vertex s, connected to all vertices with edges of cost $\overline{c} = 0$, and define the potential p_i of node i to length of shortest path from s to i using costs $\overline{c}_{ij}$. By the definition of shortest path, for all $(i,j) \in A_f$, $p_j \leq p_i + \overline{c}_{ij} = p_i + c_{ij} - \mu(f)$ which implies $c^p_{ij} = c_{ij} + p_i - p_j \geq \mu(f)$ for all $(i,j) \in A_f$, which implies the result. $\square$

Given circulation f, let $f^{(i)}$ denote the circulation i iterations later. The following theorems, which we will prove later, will show that the Goldberg-Tarjan algorithm runs in polynomial time.

Theorem 6.2 $\epsilon(f^{(1)}) \leq \epsilon(f)$

Theorem 6.3 $\epsilon(f^{(m)}) \leq (1 - 1/n)\epsilon(f)$

Theorem 6.4 When $\epsilon(f) < 1/n$ then circulation f is optimal.

Proof: Since $\epsilon(f) < 1/n$, this implies that there exist potentials p such that $c^p_{ij} > -1/n$ for all $(i,j) \in A_f$. Thus for all cycles $\Gamma \in A_f$, $c^p(\Gamma) > -1$, which by integrality of costs gives $c(\Gamma) \geq 0$. $\square$

We now prove use the previous three results to prove that the Goldberg-Tarjan algorithm terminates in time bounded by a polynomial in the input size.

Theorem 6.5 The Goldberg-Tarjan minimum mean-cost cycle cancelling algorithm requires at most $O(mn\log(nC))$ iterations, where $C = \max_{(i,j)\in A}|c_{ij}|$.

Proof:

Any initial circulation is C-optimal by the assignment $p_i = 0$ for all $i \in V$. After $k = mn\log(nC)$ iterations, we have that

$$\epsilon(f^{(k)}) \leq (1 - 1/n)^{n\log(nC)}C < e^{-\log(nC)}C = 1/n,$$

using the fact that $(1 - 1/n)^n < e^{-1}$. This proves the optimality of $f^{(k)}$ by Theorem 6.4. $\square$

The running of the Goldberg-Tarjan algorithm is $O(m^2n^2 \log(nC))$ time as min-mean cycle computations take $O(mn)$ time. Note that this algorithm is not strongly polynomial. A strongly polynomial algorithm will be presented in the next lecture along with the proofs of Theorem 6.2 and Theorem 6.3.

CS 361B Advanced Algorithms April 24, 2003

Lecture 7

Lecturer: David P. Williamson *Scribe: Sergei Vassilvitskii*

7.1 Minimum-cost circulations (cont.)

7.1.1 Goldberg-Tarjan min-mean cycle cancelling (cont.)

Remember that we first got rid of lower bounds on the edges by introducing opposite edges:

$$\begin{tikzcd} i \arrow[r, bend left, "l_{ij}\le f_{ij}\le u_{ij}", "c_{ij}"'] & j & \Longrightarrow & i \arrow[r, bend left, "{f_{ij}\le u_{ij},\ c_{ij}}"] & j \arrow[l, dotted, "{f_{ji}=-f_{ij}\le -l_{ij}=u_{ji},\ c_{ji}=-c_{ij}}"] \end{tikzcd}$$

With this change in mind, we can rewrite the conservation of flow condition:

$$\sum_{j:(i,j)\in A} f_{ij} - \sum_{j:(j,i)\in A} f_{ji} = 0 \Longrightarrow \sum_{j:(i,j)\in A} f_{ij} = 0$$

We have also shown last time the following:

Theorem 7.1 For a circulation f the following are equivalent:

1. f is of minimum cost
2. There are no negative cost cycles in G_f
3. There exist potentials p such that $c^p_{ij} \geq 0 \ \forall (i,j) \in A_f$.

Recall that we defined the minimum-mean cost cycle of the residual graph as

Definition 7.1 $\mu(f) = \min\limits_{\text{cycles } \Gamma\in G_f} \dfrac{c(\Gamma)}{|\Gamma|}$

We can now present an algorithm for finding a minimum cost circulation.

Min-mean cycle cancelling algorithm (Goldberg & Tarjan 1989)

```
Find initial circulation $f$
While $\mu(f) < 0$
    Cancel mean-min cycle $\Gamma$, update $f$
```

Definition 7.2 A circulation f is ϵ-optimal if $\exists$ potentials p such that $c^p_{ij} \geq -\epsilon \; \forall (i,j) \in A_f$. Further, we define $\epsilon(f)$ as the minimum ϵ such that f is an ϵ-optimum circulation.

If given a flow f we denote by $f^{(i)}$ the circulation after i cancellations. Last time we stated the following theorems:

Theorem 7.2 $\epsilon(f^{(1)}) \leq \epsilon(f)$

Theorem 7.3 $\epsilon(f^{(m)}) \leq \left(1 - \frac{1}{n}\right) \epsilon(f)$

Theorem 7.4 If $\epsilon(f) < \frac{1}{n}$ then f is a minimum cost circulation.

Theorem 7.5 Let $C = \max_{i,j} |c_{ij}|$. Then the above algorithm terminates after at most $\mathcal{O}(mn \log nC)$ iterations. This gives the overall running time of $\mathcal{O}(m^2 n^2 \log nC)$.

Last time we showed that Theorem 7.5 follows from Theorems 7.2, 7.3, and 7.4, and gave a proof for Theorem 7.4. We now complete the proof of Theorem 7.5 by proving the other two theorems. Recall that last time we also proved the following.

Theorem 7.6

$$\mu(f) = -\epsilon(f).$$

Proof of Theorem 7.2: We know there exist potentials p such that $c^p_{ij} \geq -\epsilon(f)$ $\forall (i,j) \in A_f$. For a cancelled cycle Γ, $\mu(f) = -\epsilon(f)$. Since $\mu(f) = c^p(\Gamma)/|\Gamma|$, it follows that for all$(i,j) \in \Gamma$ $c^p_{ij} = -\epsilon(f)$. We now claim that $c^p_{ij} \geq -\epsilon(f)$ for all $(i,j) \in A_{f^{(1)}}$.

$$\forall (i,j) \in A_f \cap A_{f^{(1)}} \text{ this is true, nothing changed}$$
$$\forall (i,j) \in A_{f^{(1)}} - A_f \Rightarrow (j,i) \in \Gamma \Rightarrow c^p_{ji} = -c^p_{ij} = \epsilon(f) \geq 0.$$

□

Proof of Theorem 7.3: Again we know there exist potentials p such that $c^p_{ij} \geq -\epsilon(f)$ for all $(i,j) \in A_f$. Suppose that in some iteration k we cancel cycle Γ and $\exists (i,j) \in \Gamma$ such that $c^p_{ij} \geq 0$ Then:

$$\begin{aligned} -\epsilon(f^{(k)}) = \mu(f^{(k)}) &= \frac{c^p(\Gamma)}{|\Gamma|} \\ &\geq \frac{|\Gamma| - 1}{|\Gamma|}(-\epsilon(f)) \\ &\geq \left(1 - \frac{1}{n}\right)(-\epsilon(f)). \end{aligned}$$

Thus

$$\epsilon(f^{(k)}) \leq \left(1 - \frac{1}{n}\right) \epsilon(f).$$

How many consecutive iterations can it be the case that cycle Γ that is cancelled has $c^p_{ij} < 0$ for all $(i,j) \in \Gamma$? Cancelling the cycle removes one edge with $c^p_{ij} < 0$ from the residual graph and creates only edges with $c^p_{ij} \geq 0$. So we need no more than m iterations before we cancel such a cycle Γ. □

7.1.2 A strongly polynomial time analysis

Definition 7.3 An algorithm runs in strongly polynomial time if the number of basic operations (e.g. additions, subtractions, multiplications, comparisons, etc.) can be bounded by a polynomial in the number of data items that were input and is not dependent on the size of data inputs (e.g. bits to encode cost, lower bounds, etc).

If an algorithm is strongly polynomial for minimum cost circulations, its running time depends only on m and n. The first such algorithm is due to Éva Tardos in 1985.

Definition 7.4 An arc $(i, j) \in A$ is ϵ-fixed if the flow on it is the same for all ϵ-optimal circulations f.

Theorem 7.7 Let $\epsilon > 0$, let f be a circulation, and let p be potentials such that f is ϵ-optimal with respect to p. If $|c^p_{ij}| \geq 2n\epsilon$ then (i, j) is ϵ-fixed.

Proof: Suppose that f' is an ϵ-optimal circulation such that $f'_{ij} \neq f_{ij}$. Assume that $c^p_{ij} \leq 2n\epsilon$; this is without loss of generality since costs are antisymmetric.

Claim 7.8 There exists a cycle Γ in $A_{f'}$ such that $(i, j) \in \Gamma$.

We look at flow conditions across a particular cut. For any set S we know:

$$\sum_{i:(i,j)\in A} f_{ij} = 0 \Longrightarrow \sum_{j\in S} \sum_{i:(i,j)\in A} f_{ij} = 0$$

We also have the antisymmetry conditions:

$$f_{ij} + f_{ji} = 0, \forall (i, j) \in S$$

Combining the two, we conclude:

$$\sum_{\substack{i\in S\\ j\notin S\\ (i,j)\in A}} f_{ij} = 0$$

Since $c^p_{ij} \leq -2n\epsilon$ we know that $(i, j) \notin A_f$ because of ϵ-optimality of f. Therefore $f_{ij} = u_{ij}$. Thus we must have $f'_{ij} < u_{ij}$.

Let $E_< = \{(k, l) \in A : f'_{kl} < f_{kl}\}$. Observe that $E_< \subseteq A_{f'}$. Let S be the set of nodes reachable from j in $E_<$. We will show that $i \in S$ therefore a cycle Γ exists as claimed.

Suppose by contradiction that $i \notin S$.

$$\sum_{\substack{k\in S\\ l\notin S}} f_{kl} = 0 \wedge \sum_{\substack{k\in S\\ l\notin S}} f'_{kl} = 0 \Longrightarrow \sum_{\substack{k\in S\\ l\notin S}} (f_{kl} - f'_{kl}) = 0$$

But $f'_{ij} < f_{ij} \Rightarrow f'_{ji} > f_{ji}$. Therefore there is a term in the sum that is negative. Then there must be a term that is positive. So $\exists (k,l), k \in S, l \notin S$ such that $f'_{kl} < f_{kl}$. By then $(k,l) \in E_<$ and since $k \in S$, it must be that $l \in S$, which is a contradiction. $\diamond$

Therefore we know that if $|c^p_{ij}| \geq 2n\epsilon$ then (i,j) is part of a cycle Γ in the set of edges (k,l) for which $f'_{kl} < f_{kl}$. Note that this implies that the reverse cycle $\Lambda = \{(l,k) : (k,l) \in \Gamma\}$ exists in the set of arcs (l,k) for which $f'_{lk} > f_{lk}$, which implies that Λ exists in A_f since flow on the edges in this cycle cannot be at their upper bounds. Since f is ϵ-optimal we know that for $(l,k) \in A_f, c^p_{lk} \geq -\epsilon$. Therefore for any $(k,l) \in \Gamma$ we know that $c^p_{kl} \leq \epsilon$.

We know that $\mu_(f') = -\epsilon(f') \geq -\epsilon$. Thus:

$$\begin{aligned}
\frac{c(\Gamma)}{|\Gamma|} &= \frac{c^p(\Gamma)}{|\Gamma|} \\
&= \frac{1}{|\Gamma|}\left(c^p_{ij} + \sum_{(k,l)\in\Gamma:(k,l)\neq(i,j)} c^p_{kl}\right) \\
&\leq \frac{1}{|\Gamma|}(-2n\epsilon + (|\Gamma|-1)\epsilon) \\
&< \frac{1}{|\Gamma|}(-|\Gamma|\epsilon) \\
&= -\epsilon
\end{aligned}$$

Therefore there exists a cycle in $A_{f'}$ whose mean cost is less than $-\epsilon$, which is a contradiction. Therefore the flow on the arc (i,j) must be fixed. $\square$

We can now show that this analysis gives a strongly polynomial time algorithm.

Theorem 7.9 The algorithm terminates after $\mathcal{O}(m^2 n \log n)$ iterations.

Proof: Once an arc is fixed, it will always remain fixed since $\epsilon(f)$ is non-increasing. We now claim that a new arc will be fixed after at most $k = mn\log(2n)$ iterations. Let f be the current circulation and Γ be the cycle cancelled in this iteration. Then

$$\begin{aligned}
\epsilon(f^{(k)}) &\leq \left(1 - \frac{1}{n}\right)^{n\log 2n} \epsilon(f) \\
&< e^{-\log 2n}\epsilon(f) \\
&= \frac{\epsilon(f)}{2n}
\end{aligned}$$

Let p^k be the potentials associated with the flow $f^{(k)}$ such that the flow is $\epsilon(f^{(k)})$-optimal. Then

$$-\epsilon(f) = \frac{c^{p^k}(\Gamma)}{|\Gamma|} < -2n\epsilon(f^{(k)})$$

Therefore, $\exists (i,j) \in \Gamma$ such that $c_{ij}^{p^k} < -2n\epsilon(f^{(k)})$. Therefore (i,j) is fixed.

Further, note that (i,j) was not $\epsilon(f)$-fixed since $(i,j) \in \Gamma$ and the flow on it changed when we cancelled Γ. But if it was $\epsilon(f)$-fixed, the flow on it would not have changed. Therefore we fixed a new edge. $\square$

CS 361B Advanced Algorithms April 29, 2003

Lecture 8

Lecturer: David P. Williamson *Scribe: Zoë Abrams*

8.1 The primal-dual method

8.1.1 Introduction

So far, the algorithms we have studied have been primal algorithms. The algorithms start with some feasible primal solution and move towards optimality. One could also consider a dual algorithm, which maintains a dual feasible solution, and moves towards optimality. Today we will start discussions of a special case of dual algorithms known (in combinorial optimization) as primal-dual algorithms. They start with some dual feasible solution and a primal infeasible solution. The algorithm moves to reduce the infeasibility of the primal and increase the value of the dual while maintaining complimentary slackness. We will introduce the primal-dual method with an algorithm for the min-cost circulation problem. The fastest min-cost circulation algorithm is a dual algorithm due to Orlin, and has running time $O(m \log n(m + n \log n))$. We will then use the primal-dual method in an approximation algorithm for a problem called the hitting set problem.

8.1.2 A minimum-cost circulation algorithm

The min-cost circulation problem is defined at the begining of Lecture 5. To begin, we will look at the Primal LP for this problem.

$$\text{Min} \quad \sum_{(i,j)\in A} c_{ij} x_{ij}$$

subject to:

$$\sum_{k:(k,i)\in A} x_{ki} - \sum_{k:(i,k)\in A} x_{ik} = 0 \qquad \forall i \in V$$

$$l_{ij} \le x_{ij} \le u_{ij}.$$

We then take the dual of this LP to obtain the following:

$$\text{Max} \quad \sum_{(i,j)\in A} l_{ij}w_{ij} - \sum_{(i,j)\in A} u_{ij}z_{ij}$$

subject to:

$$\begin{aligned} &p_j - p_i + w_{ij} - z_{ij} = c_{ij} && \forall (ij) \in A \\ &w_{ij} \geq 0 \\ &z_{ij} \geq 0. \end{aligned}$$

Now, suppose that the node potentials p are given. The reduced cost $c^p_{ij} = c_{ij} + p_i - p_j$, and $c_{ij} + p_i - p_j = w_{ij} - z_{ij}$ in the dual LP. If we know the potentials, then we can compute all dual variables by setting $w_{ij} = \max(C^p_{ij}, 0) \equiv (c^p_{ij})^+$ and $-z_{ij} = min(c^p_{ij}, 0) \equiv (c^p_{ij})^-$. Therefore, finding potentials p yields a solution to the dual and the following LP is equivalent to the dual LP:

$$\text{Max} \quad \sum_{(i,j)\in A} l_{ij}(c^p_{ij})^+ - \sum_{(i,j)\in A} u_{ij}(c^p_{ij})^-$$

subject to:

$$c_{ij} + p_i - p_j = c^p_{ij} \qquad \forall (ij) \in A$$

Complementary slackness conditions are then:

$$\begin{aligned} c^p_{ij} > 0 \Leftrightarrow w_{ij} > 0 \Rightarrow x_{ij} = l_{ij} \\ c^p_{ij} < 0 \Leftrightarrow z_{ij} > 0 \Rightarrow x_{ij} = u_{ij} \end{aligned}$$

Our primal dual algorithm will start with a dual feasible solution by setting all potentials equal to 0. We will then determine whether there exists a primal feasible solution that obeys complimentary slackness by defining a new circulation problem with modified upper and lower bounds $\tilde{u}$ and $\tilde{l}$.

$$\begin{aligned} C^p_{ij} > 0 \Rightarrow \tilde{l}_{ij} = \tilde{u}_{ij} = l_{ij} \\ C^p_{ij} < 0 \Rightarrow \tilde{l}_{ij} = \tilde{u}_{ij} = u_{ij} \\ C^p_{ij} = 0 \Rightarrow \tilde{l}_{ij} = l_{ij}, \tilde{u}_{ij} = u_{ij} \end{aligned}$$

As with most primal dual approaches, we have reduced a problem with cost to a problem without cost where we only need to check for feasibility. If we can find a feasible circulation in the problem with bounds $\tilde{l}_{ij}$ and $\tilde{u}_{ij}$, we are finished, since then we will have a primal feasible solution and a dual feasible solution that obey the complementary slackness conditions, and thus are optimal.

If not we can find a cut S such that $\tilde{l}(\delta^+(S)) > \tilde{u}(\delta^-(S))$. Here $\delta^+(S) = \{(i,j) \in A : i \in S, j \notin S\}$ is the set of arcs in the graph such that the tail is in S but the head is not. Similarly, $\delta^-(S) = \{(i,j) \in A : i \notin S, j \in S\}$ is the set of arcs in the graph such that the head is in S but the tail is not. We will modify the dual to increase the dual objective function. To do this, we will increase the potentials of nodes in the cut S by a value β, chosen so that after modification all reduced costs stay on the same side of zero.

$$\beta = \min(\min(|c^p_{ij}| : (i,j) \in \delta^+(S) \wedge c^p_{ij} < 0), \min(c^p_{ij} : (i,j) \in \delta^-(S) \wedge c^p_{ij} > 0)).$$

Lemma 8.1 Increasing p_i by β for all $i \in S$ strictly increases the dual objective function.

Proof: How is the dual changed? To answer this, we define the reduced costs after the addition of β:

$$\hat{C}^p_{ij} = \begin{cases} C^p_{ij} + \beta & (i,j) \in \delta^+(S) \\ C^p_{ij} - \beta & (i,j) \in \delta^-(S) \\ C^p_{ij} & \text{Otherwise} \end{cases}$$

The dual changes by

$$\beta \left(\sum_{(i,j)\in\delta^+(S):c^p_{ij}\geq 0} l_{ij} - \sum_{(i,j)\in\delta^-(S):c^p_{ij}>0} l_{ij} + \sum_{(i,j)\in\delta^+(S):c^p_{ij}<0} u_{ij} - \sum_{(i,j)\in\delta^-(S):c^p_{ij}\leq 0} u_{ij} \right).$$

Note that

$$\sum_{(i,j)\in\delta^+(S):c^p_{ij}\geq 0} l_{ij} + \sum_{(i,j)\in\delta^+(S):c^p_{ij}<0} u_{ij} = \tilde{l}(\delta^+(S))$$

and

$$\sum_{(i,j)\in\delta^-(S):c^p_{ij}>0} l_{ij} + \sum_{(i,j)\in\delta^-(S):c^p_{ij}\leq 0} u_{ij} = \tilde{u}(\delta^-(S)).$$

Thus the dual changes by

$$\beta \left(\tilde{l}(\delta^+(S)) - \tilde{u}(\delta^-(S)) \right) > 0,$$

since by hypothesis $\tilde{l}(\delta^+(S)) > \tilde{u}(\delta^-(S))$. □

In summary:

Primal-Dual Algorithm

```
$p \leftarrow 0$
While the solution to the new circulation problem (using $\tilde{l}, \tilde{u}$) is infeasible
    Find a cut $S$
    Increase potentials of nodes in $S$ by $\beta$
Return circulation in the new circulation problem.
```

The running time of this algorithm is $O(mCU * \text{max flow compuations})$ where m is the number of arcs, C is the value of the largest edge cost, and U is the value of the largest capacity. This is since the dual solution can be at worst $-mCU$ and at most mCU and we increase the dual by at least 1 each iteration by the integrality of the costs and capacities. The algorithm is not polynomial time because of its dependence on C and U. However, it is *pseudopolynomial*; that is, it is polynomial time if the input is encoded in unary.

8.1.3 The hitting set problem

There are many problems, unlike the problems we have dealt with until now, that are not known to be solvable in polynomial time. However, we can still design polynomial algorithms with solutions that are provably close to the value of an optimal solution.

Definition 8.1 An algorithm is an *α-approximation algorithm* for an optimization problem if

1. The algorithm runs in polynomial time
2. The algorithm always produces a solution which is within a factor of α of the value of the optimal solution

For example, the TSP problem has a $\frac{3}{2}$-approximation. Notice that this definition does not require the problem to be NP-Hard. We will show how the primal dual method can be used in an approximation algorithm for the Hitting Set problem.

Hitting Set

- **Input:**
  - ground set $E = \{e_1, e_2, \ldots, e_n\}$
  - subsets $T_1, T_2, \ldots, T_p \subseteq E$
  - costs $c_e \geq 0 \quad e \in E$
- **Goal:** Find min-cost $A \subseteq E$ s.t. $A \cap T_i \neq \emptyset \quad \forall i$

The integer program below models the hitting set problem.

$$\text{Min} \sum_{e \in E} c_e x_e$$

subject to:

$$\sum_{e \in T_i} x_e \geq 1 \qquad i = 1 \ldots p$$

$$x_e \in \{0, 1\}$$

The integer program can be relaxed to a linear program in the usual way:

$$x_e \in \{0, 1\} \rightarrow x_e \geq 0.$$

Once we have an LP relaxation, we can take the dual of this LP, and obtain the following:

$$\begin{aligned} \text{Max} \quad & \sum_i y_i \\ \text{subject to:} \quad & \\ & \sum_{i: e \in T_i} y_i \leq c_e && \forall e \in E \\ & y_i \geq 0 && \forall i \end{aligned}$$

We want to show for our primal solution $A \subseteq E$ that

$$\sum_{e \in A} c_e \leq \alpha \sum_{i=1}^{p} y_i \leq \alpha OPT,$$

for some value of α, where OPT is the optimal value of the integer primal solution. Note that for any feasible dual solution y, the dual objective function $\sum_i y_i$ is a lower bound on the cost an optimal solution to the linear programming relaxation, which is a lower bound on the cost of an optimal integer solution, since the integer optimal solution is feasible for the linear programming relaxation. Thus $\sum_{i=1}^{p} y_i \leq OPT$.

Our algorithm starts with the dual feasible solution $y_i = 0 \ \forall i$. Complimentary slackness conditions are:

$$y_i > 0 \Rightarrow \sum_{e \in T_i} x_e = 1$$
$$x_e > 0 \Rightarrow \sum_{i: e \in T_i} y_i = c_e$$

We cannot obtain an optimal integral solution in polynomial time (unless $P = NP$), so we will have to modify the primal-dual schema in some fashion. Here we will drop the first condition and maintain only the second. Given a feasible dual solution y, the least infeasible solution that obeys the second complementary slackness condition is obtained by setting $A = \{e \in E : \sum_{i: e \in T_i} y_i = c_e\}$. A is feasible iff $A \cap T_i \neq \emptyset \ \forall i$. If A is feasible, we are done. Otherwise, $\exists k$ s.t. $A \cap T_k = \emptyset \Rightarrow \forall e \in T_k \ \sum_{i: e \in T_i} y_i < c_e$. We call such a set T_k a *violated* set.

Therefore, we can increase the dual objective function by raising the value of y_k. We will increase y_k by $\arg\min_{e \in T_k} \{c_e - \sum_{i: e \in T_i} y_i\}$.

This leads to the following algorithm:

Primal-Dual Algorithm

```
$y \leftarrow 0$
$A \leftarrow \emptyset$
While $A$ is not feasible
    Choose some violated $T_k$ (i.e. $T_k$ s.t. $A \cap T_k = \emptyset$)
    Increase $y_k$ until $\exists\ e \in T_k$ such that $\sum_{i:e \in T_i} y_i = c_e$
    $A \leftarrow A \cup \{e\}$
Return $A$.
```

Once an element e is in A, none of the sets T_i that contain it can be violated, therefore none of the associated dual variables y_i will be increased later in the algorithm. Therefore,

$$\sum_{e \in A} c_e = \sum_{e \in A} \sum_{i:e \in T_i} y_i$$

because the second complimentary slackness condition was maintained throughout the algorithm. Further,

$$\sum_{e \in A} \sum_{i:e \in T_i} y_i = \sum_{i=1}^{p} y_i |A \cap T_i|$$

We now obtain an approximation algorithm by showing that $\forall i, |A \cap T_i| \leq \alpha$ for some reasonable α. As an example, we apply this algorithm to the vertex cover problem. We transform vertex cover into a hitting set problem where V is the ground set of elements, the costs c_e of the elements are the weights of the vertices, and we must hit the sets $T_i = \{u, v\}$ for each $(u, v) \in E$. Since $|T_i| = 2$ for each set, it follows that $|A \cap T_i| \leq 2$ for all i, and by the reasoning above we have a 2-approximation algorithm for vertex cover.

CS 361B Advanced Algorithms May 1, 2003

Lecture 9

Lecturer: David P. Williamson *Scribe: Sanders Chong*

9.1 The primal-dual method (cont.)

9.1.1 The feedback vertex set problem

In previous lectures we discussed methods to "solve" the Hitting Set and Vertex Cover within a factor α of optimal. Furthermore, we introduced the primal-dual formulation to generate an algorithm and prove α-optimality. In this lecture, we begin by presenting yet another example: the Feedback Vertex Set Problem (FVS).

Feedback Vertex Set in Undirected Graphs

- **Input:**
  - Undirected graph $G = (V, E)$
  - Weights $w_i \geq 0 \quad \forall i \in V$
- **Goal:** Find $S \subseteq V$ minimizing $\sum_{i \in S} w_i$ such that for every cycle C in G, $C \cap S \neq \emptyset$. (Equivalently, find a min-weight set of vertices S such that removing S from the graph causes the remaining graph to be acyclic).

We claim that the feedback vertex set problem is just a hitting set problem with:

- Ground set V
- Cost w_i
- Sets to hit: $T_i = C_i$ for each cycle C_i in graph

We now have a hitting set problem with potentially an exponential number of sets to hit. How do we deal with this problem? The answer is that we do not need to enumerate or find all cycles: the algorithm only needs to find a violated set when one exists.

Before we formulate the FVS using the primal-dual method, let us first recap what is needed: To apply the primal-dual method, we first need a primal and a dual. The integer program below models the feedback vertex set problem.

$$\begin{aligned} &\text{Min} \quad \sum_{i \in V} w_i x_i \\ &\text{subject to:} \\ &\sum_{e \in C_i} x_i \geq 1 \qquad \forall \text{ cycles} C_i \\ &x_e \in \{0,1\}. \end{aligned}$$

The integer program can be relaxed to a linear program in the usual way:

$$x_e \in \{0,1\} \rightarrow x_e \geq 0.$$

Once we have an LP relaxation, we can take the dual of this LP, and obtain the following:

$$\begin{aligned} &\text{Max} \quad \sum_{C} y_C \\ &\text{subject to:} \\ &\sum_{C: i \in C} y_C \leq w_i \qquad \forall i \in V \\ &y_C \geq 0 \qquad \forall \text{ cycles } C \end{aligned}$$

The basic goal in the primal dual method is to obtain a solution $A \subseteq E$ and a dual feasible solution y such that

$$\sum_{i \in A} w_i \leq \alpha \sum_{C} y_C \leq \alpha OPT,$$

for some value of α.

With this in mind, let us now formulate the primal-dual method here and get a good performance guarantee. This follows if we can show that $|A \cap C_i| \leq \alpha$ for all cycles C_i on which $y_i > 0$. This would be true if $|C_i| \leq \alpha$ for all cycles i, but α can be as large as n in this case. To be able to bound $|A \cap C_i|$ in this case, we consider only "interesting" vertices. An interesting vertex will be defined as a vertex that we will consider to be in solution subset A.

First of all, any vertex not in any cycle is not interesting, since it can't participate in the solution set. Additionally, on any path of nodes of degree 2, only one vertex is interesting; namely, the vertex of least cost, since any cycle which goes through the vertices on this path also goes through this vertex of least cost. Thus we may as well only consider the vertex of least cost.

To get our algorithm, we need the following lemma:

Lemma 9.1 (Erdös, Posa) In every non-empty graph which is not a forest, there is a cycle of at most $4 \log_2 n$ interesting vertices.

Proof: From G we create a graph G' consisting of only interesting vertices by deleting vertices not in any cycle, and shortcutting all remaining uninteresting vertices out of G. Note that there is still a one-to-one correspondance of cycles in G and G'. Now G' has no degree 1 vertices, and each vertex of degree 2 has two neighbors of degree more than 2.

We now find the requisite cycle. We do a breadth-first search of G'. By the properties of the graph, if we do not close a cycle by revisiting a previously explored node, then at least in every other level the number of explored nodes increases by a factor of 2. Thus at depth i, we will have explored $2^{i/2}$ nodes. By depth $2\log_2 n$, we will have found a cycle. $\square$

We can now state our primal-dual algorithm.

Primal-DualFVS (Bar-Yehuda, Geiger, Naor, Roth 1994)

```
$y \leftarrow 0$
$A \leftarrow \emptyset$
$I \leftarrow$ interesting vertices of $G$
While $A$ is not feasible
    Find cycle $C$ s.t. $|C \cap I| \leq 4\log_2 n$
    Increase $y_C$ until $\exists\, i \in I : \sum_{C: i \in C} y_C = w_i$
    $A \leftarrow A \cup \{i\}$
    Remove $i$ from graph, remove all vertices no longer in cycles from $I$,
        remove new uninteresting verts (w.r.t. to prices
        $\tilde{w_i} = w_i - \sum_{C: i \in C} y_i$).
Return $A$.
```

We should observe that no uninteresting vertex ever becomes interesting again. Certainly any vertex not in any cycle will be part of a cycle again. Similarly, any uninteresting vertex on a path of degree two nodes will always be in the same cycles as the interesting vertex on that path, and thus choosing the interesting vertices in the way specified above guarantees that an interesting vertex will be one for which the dual inequality becomes tight first. This also implies that the dual solution we construct is feasible. We leave a formal proof of this to the reader.

Theorem 9.2 (Bar-Yehuda, Geiger, Naor, Roth '94) PrimalDualFVS is a $(4\log_2 n)$-approximation algorithm for the feedback vertex set problem in undirected graphs.

Proof: By using the standard primal-dual analysis, we have that

$$\begin{aligned}\sum_{i\in A} w_i &= \sum_{i\in A}\sum_{C:i\in C} y_C \\ &= \sum_C y_C|A\cap C| \\ &\le \sum_C y_C|I\cap C| \\ &\le (4\log_2 n)\sum_C y_C \\ &\le (4\log_2 n)OPT.\end{aligned}$$

□

In fact, one can get a 2-approximation algorithm for this problem using the primal-dual method; however, one has to use a substantially different integer programming formulation of the problem, and we will not get into it here. For further discussion, see papers by Chudak, Goemans, Hochbaum, and Williamson; and the paper by Fujito.

9.1.2 Shortest s-t path

Here we consider the problem of finding the shortest s-t path in an undirected graph. This problem can be seen as a hitting set problem as follows:

Ground Set : the set of edges E
Costs : $c_e \ge 0,\ \forall e \in E$
Sets to Hit : $T_i = \delta(S_i),\ s \in S_i,\ t \notin S_i$

where $\delta(S) = \{(u,v) \in E :\ u \in S \text{ and } v \notin S\}$. That is, the sets S_i are the s-t cuts and the sets $T_i = \delta(S_i)$ are the edges crossing the s-t cuts.

To see that this hitting set problem captures the shortest s-t path problem, we need that a set of edges contains an s-t path if and only if it hits every s-t cut[1]. First, if a set of edges A does not cross some s-t cut S_i then A must consist exclusively of edges joining two vertices of S_i or joining two vertices of the complement of S_i. Thus any path starting from $s \in S_i$ consisting of such edges can only bring us to vertices that are also in S_i, but $t \notin S_i$. Conversely, if a set of edges does not contain an s-t path then let S_i be the largest connected component (corresponding to those edges) containing s. By assumption $t \notin S_i$ and the set of edges could not contain any edge from $\delta(S_i)$ or else we could have found a larger connected component containing s by including the other vertex incident on that edge. Thus the absence of an s-t path implies that some s-t cut was not hit. We find then that a set of edges contains an s-t path if and only if it hits every s-t cut.

[1]This follows directly from the max-flow/min-cut theorem, but for completeness we prove it here.

We now wish to apply the primal-dual method to this problem. We set up our primal and dual:

$$\text{Min} \quad \sum_{e \in E} c_e x_e$$

subject to:

$$\sum_{e \in \delta(S)} x_e \geq 1 \qquad \forall S : s \in S, t \notin S$$
$$x_e \geq 0$$

$$\text{Max} \quad \sum_{S} y_S$$

subject to:

$$\sum_{S: e \in \delta(S)} y_S \leq c_e \qquad \forall e \in E$$
$$y_S \geq 0 \qquad \forall S : s \in S, t \notin S$$

To use our primal-dual algorithm, we have to say which violated s-t cut we will choose in each iteration of the algorithm. We will choose the cut S which is the connected component containing S. By the reasoning above, S will not contain t unless we have already selected an s-t path.

The problem with the standard primal-dual method is that it will include too many edges not on the path from s to t. Hence we will need to add some steps at the end to get rid of all edges not on the s-t path. We do this in a particular fashion below (in the reverse of the order in which edges are added) not so much because it is useful for this particular problem, but because it is useful for several other problems.

Primal-DualSP

```
$y \leftarrow 0$
$A \leftarrow \emptyset$
$l \leftarrow 0$ ($l$ is a counter)
While $A$ is not feasible
    $l \leftarrow l + 1$
    Let $S$ be the connected component containing $s$
    Increase $y_S$ until $\exists\ e_l \in \delta(S) : \sum_{S: e_l \in \delta(S)} y_S = c_{e_l}$
    $A \leftarrow A \cup \{e_l\}$
For $j \leftarrow l$ down to 1
    If $A - \{e_j\}$ is still feasible
        $A \leftarrow A - \{e_j\}$
Return $A' \leftarrow A$.
```

Once again, we have that

$$\sum_{e \in A'} c_e = \sum_{e \in A'} \sum_{S: e \in \delta(S)} y_S = \sum_S y_S |A' \cap \delta(S)|.$$

In the following theorem, we will see that for all S such that $y_S > 0$, $|A' \cap \delta(S)| = 1$. Thus this algorithm is optimal; in fact, it is just Dijkstra's algorithm cleverly disguised.

Theorem 9.3 For all S such that $y_S > 0$, $|A' \cap \delta(S)| = 1$.

Proof: Let A_S be the edges in A when we increased the variable y_S. Let $B = A' - A_S$. Let k be the iteration in which we increase y_S. Observe that $A_S \cup B$ contains an s-t path, and furthermore, that if we remove any edge $e \in B$, $A_S \cup B - e$ no longer contains an s-t path. This follows because when we go through the edge deletion step at the end of the algorithm, when we reach the consideration of edge e_k, the remaining edges in A are exactly $A_S \cup B$, and all edges in B were necessary for feasibility.

Now let $s, v_1, v_2, \ldots, v_l, t$ be an s-t path in $(V, A_S \cup B)$. Choose i such that $v_i \in S$, $v_{i+1} \notin S$ where i is as large as possible. Since S is a connected component there must be a s-v_i path exclusively in S of the form $s, w_1, w_2, \ldots, w_j, v_i$, where $w_\ell \in S$. Thus $s, w_1, w_2, \ldots, w_j, v_i, v_{i+1}, \ldots, v_l, t$ is an s-t path. Since all the edges $(v_i, v_{i+1}), \ldots, (v_l, t)$ are not in A_S, they must be in B. Therefore

$$|A' \cap \delta(S)| = |B \cap \delta(S)| = |\{(v_i, v_{i+1})\}| = 1,$$

since the first edge is the only one to have an endpoint in S. □

CS 361B Advanced Algorithms May 6, 2003

Lecture 10

Lecturer: David P. Williamson *Scribe: Shubha Nabar*

10.1 The primal-dual method (cont.)

10.1.1 Generalized Steiner trees

We now consider another problem for which the primal-dual method gives a good approximation algorithm, the Generalized Steiner Tree problem.

Generalized Steiner Tree Problem

- **Input:**
  - An undirected graph $G = (V, E)$
  - l pairs of vertices $(s_i, t_i), i = 1 \dots l$
  - Costs $c_e \geq 0$ for each edge $e \in E$
- **Goal:** Find a minimum-cost set of edges F such that for all i, s_i and t_i are in the same connected component of (V, F).

This can be modelled as a hitting set problem:

Ground Set : the set of edges E
Costs : $c_e \geq 0, \forall e \in E$
Sets to Hit : $T_i = \delta(S_i)$ iff $|S_i \cap \{s_j, t_j\}| = 1$ for some j (the s_j-t_j cuts).

Note that by the logic we used for the shortest s-t path problem that a set of edges will be feasible for this hitting set problem if and only if it contains a path between s_i and t_i for each i.

Let us consider how to apply the primal-dual method to this problem. Suppose we do more or less the same thing here we did for the shortest s-t path problem. We know that if A is not feasible then there must be some connected component S containing s_j but not t_j for some j. Suppose the algorithm picks $\delta(S)$ as the violated set and increases its dual. The difficulty is that the reasoning used above in the s-t path problem will not yield a good bound here since a solution A' may cross the cut many times. Consider the problem for which $s = s_1 = s_2 = \cdots = s_l$ and for which there are edges $(s, t_j), \forall j$ and say that $A' = \{(s, t_j)\}, j = 1 \dots n$ is a solution in which all edges are necessary, and that crosses the cut $\delta(\{s\})$ l times, which implies an l-approximation algorithm (since $|A' \cap \delta(S)| = l$). This is not very good.

Perhaps we picked the wrong infeasible solution to augment; that is, maybe our previous algorithm will work if we change the way that we choose the violated set. It turns out that this approach does not work either because it still leads to a l-approximation in the worst case scenario (where all the s's are in one group, unconnected to any of the t's).

Suppose we consider all the reasonable choices of violated sets (e.g. all connected components S containing one of s_j or t_j for some j). If we look at the example above, the average of $|A' \cap \delta(S)|$ over these S is $\frac{l+l}{l+1} < 2$. Therefore it might be wise to pick several violated sets and increase the associated dual variables all at the same time. We do this in the algorithm below:

Primal-DualGST

```
$y \leftarrow 0$
$A \leftarrow \emptyset$
$l \leftarrow 0$ ($l$ is a counter)
While $A$ is not feasible
    $l \leftarrow l + 1$
    $\mathcal{C}_l \leftarrow \{S : S$ a connected component of $(V, A) : |S \cap \{s_j, t_j\} = 1\}$
    Increase $y_S$ for all $S \in \mathcal{C}_i$ uniformly until $\exists\ e_l \notin A : \sum_{S: e_l \in \delta(S)} y_S = c_{e_l}$
    $A \leftarrow A \cup \{e_l\}$
For $j \leftarrow l$ down to 1
    If $A - \{e_j\}$ is still feasible
        $A \leftarrow A - \{e_j\}$
Return $A' \leftarrow A$.
```

We claim that our statement about the average over this choice of violated sets holds in general.

Claim 10.1

$$\sum_{C \in \mathcal{C}_i} |A' \cap \delta(C)| \leq 2|\mathcal{C}_i|.$$

From the claim we can prove the following lemma.

Lemma 10.2 Claim 10.1 implies that

$$\sum_{e \in A'} c_e \leq 2 \sum_S y_S.$$

Proof: From prior analysis we know that

$$\sum_{e \in A'} c_e = \sum_{e \in A'} \sum_{S: e \in \delta(S)} y_S = \sum_S y_S |A' \cap \delta(S)|.$$

Hence we want to show that

$$\sum_S y_S |A' \cap \delta(S)| \leq 2 \sum_S y_S.$$

We prove this by induction on the construction of y by the algorithm. In the base case, $y = 0$, so the inequality is true. Suppose the inequality holds in the ith iteration. In iteration $i+1$, suppose we increase all dual variables y_S for $S \in \mathcal{C}_i$ by ϵ. Then the right-hand side of the inequality increases by $2\epsilon|\mathcal{C}_{i+1}|$, and the left-hand side of the inequality increases by $\epsilon \sum_{C \in \mathcal{C}_i} |A' \cap \delta(C)|$. But given Claim 10.1, this means that the increase of the left-hand side is no more than the increase of the right-hand side, and thus the inequality continues to hold. □

We now prove the claim.

Theorem 10.3

$$\sum_{C \in \mathcal{C}_i} |A' \cap \delta(C)| \leq 2|\mathcal{C}_i|.$$

Proof: Consider A_i in iteration i. Let $B = A' - A_i$. As we argued in the case of the shortest s-t path problem, $A_i \cup B$ is a feasible solution to the problem, but for any $e \in B$, $A_i \cup B - e$ is not feasible.

Suppose we contract every connected component of (V, A_i). In this contracted graph, call the nodes corresponding to the connected components in $\mathcal{C}_i$ red and the rest blue. Now consider the graph $G' = (V', B)$ where V' is the vertex set. We note that G' must be a forest, since if it had a cycle we could remove an edge of the cycle and maintain feasibility, contradicting the fact that every edge in B is necessary.

How does the inequality we wish to prove translate to the graph G'? Note that $|A' \cap \delta(C)|$ in G for a connected component C is equal to $deg(v)$ in G' for the vertex v corresponding to S. Similarly, $|\mathcal{C}_i|$ in G is simply the number of red vertices in G'. We let *Red* and *Blue* represent the sets of red and blue vertices in G', so that we can rewrite the above inequality as

$$\sum_{v \in Red} deg(v) \leq 2|Red|.$$

We will need the following claim.

Claim 10.4 If $v \in Blue$ then $deg(v) \neq 1$.

Proof: If $deg(v) = 1$ then we claim $A_i \cup B - e$ is feasible for $e \in B$ and adjacent to v; this is a contradiction. Let S be the connected component in G that corresponds to the vertex v in G'. If $A_i \cup B - e$ is not feasible, then there must be some s_j-t_j pair that is connected in $(V, A_i \cup B)$ but not in $(V, A_i \cup B - e)$. Thus either s_j or t_j is in

S, and the other vertex is in $V - S$. But then it must have been the case that $S \in \mathcal{C}_i$ and $v \in Red$, which is a contradiction. $\square$

To complete the proof, we first discard all blue nodes with $deg(v) = 0$. Then

$$\begin{aligned}
\sum_{v \in Red} deg(v) &= \sum_{v \in Red \cup Blue} deg(v) - \sum_{v \in Blue} deg(v) \\
&\leq 2(|Red| + |Blue|) - 2|Blue| \\
&= 2|Red|
\end{aligned}$$

The inequality follows since the sum of the degrees of nodes in a forest is at most twice the number of nodes, and since every blue node has degree at least two. $\square$

This 2-approximation algorithm for the generalized Steiner tree is just an example of the kind of graph problem for which the primal-dual method can obtain a good approximation algorithm. A generalization of the proof above gives 2-approximation algorithms for many other graph problems. The result above is due to Agrawal, Klein, Ravi '95 and Goemans, W '95.

CS 361B Advanced Algorithms May 9, 2003

Lecture 11

Lecturer: David P. Williamson *Scribe: Mihaela Enăchescu*

11.1 Generalized flows

In this lecture we come back to discuss algorithms on (generalized) flows. We already introduced a generalization of flows, when we considered adding costs to the edges. Today we will consider a model in which the edges are also "lossy" so the flow is no longer conserved, but transformed along edges. This models leaks, theft, taxes, etc.

$$\begin{tikzcd} 80\bullet \arrow[r, "\gamma=3/4"] & \bullet \arrow[r, "\gamma=1/2"] & \bullet 30 \end{tikzcd}$$

In the above graph, if we start with 80 units of flow, we obtain 60 units after following the first arc and 30 units after the second arc. We call the parameter γ the "gain" of the edge.

Another application for this model would be converting currency. Consider, for instance, the graph below in which we want to convert, say, \$1000 into Hungarian forints. Besides the "gain" factor we can also add, as before, capacity constraints to (some) edges, for example we can convert at most \$800 directly into forints. Note that some paths lead better rates than others; for example, the $\$ \rightarrow euro \rightarrow forint$ path gives an exchange rate of 6 $forints/\$$ as opposed to the direct path for which the rate is just 5).

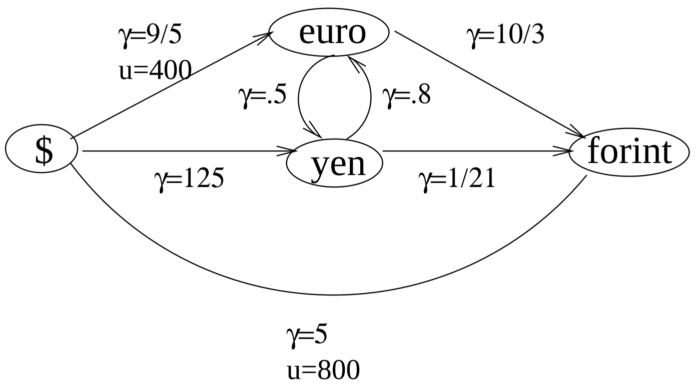


Figure 11.1: Currency conversion

11.1.1 Definitions

In this section we will define the generalized circulation problem. We will state the problem first, then give additional definition to clarify the notation/meaning of our goal.

Generalized Circulation Problem

- **Input:**
  - A *symmetric* directed graph $G = (V, A)$, i.e. $(i, j) \in A \Rightarrow (j, i) \in A$
  - Source s and sink t, $s, t \in V$
  - Integer capacities u_{ij} $\forall (i, j) \in A$
  - Gains $\gamma_{ij} : \gamma_{ji} = 1/\gamma_{ij}$ for all $(i, j) \in A$
  - All γ's are ratios of integers
  - All input integers are bounded by B.
- **Goal:** Find a circulation g that maximizes the excess e_t^g, denoted by $|g|$, and also called the value of the flow.

The following definitions will help us clarify what we mean by excess of a flow in the context of the generalized circulation problem.

Definition 11.1 A flow $g : A \to \Re$ is a *generalized pseudo-flow* if:

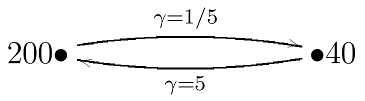
- $g_{ij} \leq u_{ij}$ $\forall (i, j) \in A$ (capacity constraints)
- $g_{ij} = -\gamma_{ij} g_{ji}$ (anti-symmetry condition)

Definition 11.2 The *residual excess* of a flow g at a node i is given by

$$e_i^g = -\sum_{j:(i,j)\in A)} g_{ij}.$$

If $e_i^g > 0$ we say we have an *excess* at node i. If $e_i^g < 0$ we say we have a *deficit* at node i.

For example if the flow on the upper edge of the figure below is 200 units, then the flow on the lower reverse edge is -40 by antisymmetry. Note that the definition of excess, although somewhat unintuitive, is capturing the notion of the total amount of flow entering a node minus that leaving the node.



Definition 11.3 A *flow* g is a pseudo-flow such that $e_i^g \geq 0 \ \forall i \in V$.

Definition 11.4 A *circulation* is a flow such that $e_i^g = 0 \ \forall i \in V, i \neq t$.

Definition 11.5 Given a pseudo-flow g in a graph $G = (V, A, u)$, we define the *residual graph* $G_g = (V, A_g, u^g)$ (where the u's denote the capacities) as follows:

$$A_g = \{h(i,j) \in A : g_{ij} < u_{ij}\}$$
$$u_{ij}^g = u_{ij} - g_{ij}$$

Definition 11.6 A *labeling function* $\mu : V \to \Re^{\geq 0} \cup \{\infty\}$ such that $\mu_t = 1$, represents the change in units of measurement of a node. Namely

$$\mu_i = \frac{\text{old units}}{\text{new units}}$$

For example if we wanted to perform the currency conversion (from Figure 1) in cents instead of dollars, we would need $\mu_\$ = 1/100$. The conversion rates involving the relabeled node would be affected (5 forints/\$ becomes .05 forints/cent), and also the capacity of the edges incident to the node (800 would become 80000 on the lowest edge, for instance).

In general we would have to perform the following changes for the gains, capacities, and excess at each relabeled node:

$$u_{ij}^\mu = u_{ij}/\mu_i$$
$$\gamma_{ij}^\mu = \gamma_{ij} \times \mu_i/\mu_j$$
$$e_i^\mu = e_i/\mu_i$$

Note that relabelling does not change the value of $|g|$ since $\mu_t = 1$ by definition.

Definition 11.7 For a path P, we define the *gain of the path* as follows:

$$\gamma(P) = \prod_{(i,j)\in P} \gamma_{ij}$$

Similarly for a cycle C, the gain of the cycle is:

$$\gamma(C) = \prod_{(i,j)\in C} \gamma_{ij}.$$

We use the following terminology for a cycle C. If $\gamma(C) > 1$, then C is a *flow-generating cycle.* If $\gamma(C) < 1$, then C is a *flow-absorbing cycle.* If $\gamma(C) = 1$, then C is a *unit-gain cycle.*

Definition 11.8 We call μ a *canonical labeling* if

$$\mu_i = \max_{\text{path P from i to t}} \gamma(P)$$

We can find the maximum $\gamma(P)$ by setting $c_{ij} = -\log(\gamma_{ij})$, and finding the shortest path in G using lengths c_{ij}. This is true because

$$\sum_{(i,j)\in P} c_{ij} = -\sum_{(i,j)\in P} -\log(\gamma_{ij}) = -\log \prod_{(i,j)\in P} \gamma_{ij} = -\log(\gamma(P)).$$

So finding the shortest path using lengths c_{ij} is equivalent to maximizing the gain from i to t. However, shortest paths are not well-defined if we have any negative-cost cycles. Here negative-cost cycles are equivalent to having flow generating cycles, since

$$\sum_{(i,j)\in C} c_{ij} < 0 \Leftrightarrow \log(\gamma(C)) > 0 \Leftrightarrow \gamma(C) > 1.$$

We will use the convention that if we cannot reach t from i then $\mu_i = \infty$ (and also $c/\infty = 0$, $\infty/c = \infty$, $\infty c/\infty = 1$ for any constant c).

Finally we want to define what we would like to detect if we have not yet discovered the optimal solution (our circulation does not yet produce the maximal excess).

Definition 11.9 A *generalized augmenting path* (GAP) is a flow generating cycle in the *residual graph* G_g with a (possibly trivial) path from a node on cycle to the sink t.

11.1.2 Optimality conditions

We are now ready to state the optimality conditions for the generalized circulation problem.

Theorem 11.1 The following are equivalent for a generalized circulation g:

1. g is optimal
2. G_g has no generalized augmenting paths (no GAPs).
3. There exist labeling μ such that the relabeled gains satisfy

$$\gamma^{\mu}_{ij} \leq 1, \forall (i,j) \in A_g$$

Proof:

- ($\neg 2 \Rightarrow \neg 1$) Assume that a GAP exists in G_g. Let C be the flow generating cycle, and P be the path from a node i on the cycle to the sink t. Now consider a flow of δ coming into i (ignore for now the source of this flow). If we push this flow around the cycle C we end up back at i with a flow of $\delta\gamma(C)$. Since $\gamma(C) > 1$ we can pay back the original δ flow, and still remain with $\delta(\gamma(C) - 1) > 0$ amount of flow at i. Pushing forward this flow from i to t on the path P, we add an extra $\delta(\gamma(C) - 1)\gamma(P)$ flow at t. Set δ such that residual capacities (along C and P) are obeyed, and we get a circulation g' such that $|g'| > |g|$. Thus g was not optimal.

- ($2 \Rightarrow 3$) Let S be the set of nodes that can reach t in G_g. We have no GAPs (by assumption) in S, thus there are no negative cost cycles in S for costs $c_{ij} = -\log \gamma_{ij}$. Set c_i to be the shortest path from i to t with costs c_{ij}, and $\mu_i = e^{c_i}$. If $(i,j) \in A_g$ then, by definition of the c_i's we have that $c_i \le c_j + c_{ij}$. This implies that
$$\mu_i \le e^{c_{ij}}\mu_j = (1/\gamma_{ij})\mu_i.$$
Thus $\gamma_{ij}\mu_i/\mu_j \le 1$. By setting $\mu_i = \infty$ for all $i \in V - S$ we ensure that our labeling satisfies the conditions of (3).

- ($3 \Rightarrow 1$) Consider any other circulation $\tilde{g}$.
  - If $(i,j),(j,i) \in A_g$, $\gamma^\mu_{ij} \le 1$ and also $\gamma^\mu_{ji} \le 1$, by (3). However, $\gamma^\mu_{ji} = 1/\gamma^\mu_{ij}$ Thus, we conclude that $\gamma^\mu_{ji} = \gamma^\mu_{ij} = 1$.
  - If $(i,j),(j,i) \notin A_g$, then it must be the case that the flow through (i,j) is completely determined by the capacity constraints. Thus $g_{ij} = \tilde{g}_{ij}$ and $g_{ji} = \tilde{g}_{ji}$.
  - If $(i,j) \in A_g$ and $(j,i) \notin A_g$ we have $\gamma^\mu_{ij} \le 1 \Rightarrow \gamma^\mu_{ji} \ge 1$. Thus $g_{ji} = u_{ji} \ge \tilde{g}_{ji}$ Also, since $g_{ij} = -\gamma_{ji}g_{ji}$ and $\tilde{g}_{ij} = -\gamma_{ji}\tilde{g}_{ji}$ (by antisymmetry), it follows that $g_{ji} \ge \tilde{g}_{ji}$ implies that $g_{ij} \le \tilde{g}_{ij}$.

  In all the above three cases we have that for any arc (i,j)
$$(\gamma^\mu_{ij} - 1)(g_{ij} - \tilde{g}_{ij}) \ge 0,$$
with equality, actually, in the first two cases. Summing over all arcs, we obtain
$$\sum_{(i,j)\in A} (\gamma^\mu_{ij} - 1)(g_{ij} - \tilde{g}_{ij}) \ge 0.$$
We can rewrite this as
$$\sum_{(i,j)\in A} \gamma^\mu_{ij}(g_{ij} - \tilde{g}_{ij}) - \sum_{(i,j)\in A} (g_{ij} - \tilde{g}_{ij}) \ge 0.$$
By antisymmetry $-\gamma^\mu_{ij}g_{ij} = g_{ji}$, so we again rewrite the above as
$$\sum_{(i,j)\in A} (\tilde{g}_{ji} - g_{ji}) - \sum_{(i,j)\in A} (g_{ij} - \tilde{g}_{ij}) \ge 0.$$

Since in a circulation $e_g^i = -\sum_{j:(i,j)\in A} g_{ij} = 0$, for all $i \neq t$, we can reduce in the previous expression, for both g and $\tilde{g}$, all arcs that are not leaving t. We obtain, finally, that

$$\sum_{i:(t,i)\in A} \tilde{g}_{ti} - \sum_{i:(t,i)\in A} g_{ti} \geq 0 \Leftrightarrow -\sum_{i:(t,i)\in A} g_{ti} \geq -\sum_{i:(t,i)\in A} \tilde{g}_{ti}.$$

The last expression is, by definition, $|g| \geq |\tilde{g}|$, and it is true for any arbitrary circulation $\tilde{g}$. Thus we can conclude that g is optimal, so (1) holds.

$\square$

CS 361B Advanced Algorithms May 13, 2003

Lecture 12

Lecturer: David P. Williamson *Scribe: Chuong Do*

12.1 Generalized flows (cont.)

12.1.1 Truemper's algorithm for generalized flow

Last class, we considered a generalized circulation problem in which arcs (i, j) are associated with gains $\gamma_{ij} > 0$ which serve as multiplicative transformations of flows along those edges. We defined *pseudoflows* $g : A \to \Re$ which met capacity ($g_{ij} \leq u_{ij}$) and antisymmetry ($g_{ji} = -\gamma_{ij} g_{ij}$) constraints, and defined the residual excess of a node i in a pseudoflow g as $e_i^g = -\sum_{j:(i,j)\in A} g_{ij}$. The generalized circulation problem then was to find a pseudoflow that maximized the residual excess at some sink node t, $e_t^g \equiv |g|$, subject to constraints that $e_i^g = 0, \forall i \in V, i \neq t$.

To do this, we defined the residual graph $G_g = (V, A_g, u^g)$ consisting of edges $(i, j) \in A$ with $g_{ij} < u_{ij}$ such that $u_{ij}^g = u_{ij} - g_{ij}$. We also developed a notion analogous to the reduced costs of the min-cost circulation problem. Given a labeling function $\mu : V \to \Re^{\geq 0} \cup \{\infty\}$ subject to the scaling constraint that $\mu_t = 1$, we defined the relabelled quantities $u_{ij}^\mu = u_{ij}/\mu_i$, $\gamma_{ij}^\mu = \gamma_{ij}\mu_i/\mu_j$, and $e_i^{g,\mu} = e_i^g/\mu_i$. We called μ a canonical labeling if $\mu_i = \max_{\text{paths } P \text{ in } G_g \text{ from } i \text{ to } t} \gamma(P) = \prod_{(i,j)\in P} \gamma_{ij}$ for all $i \in V$. We noted that we could compute these canonical labels using the Bellman-Ford shortest path algorithm with costs $c_{ij} = -\log \gamma_{ij}$ as long as there were no negative cost cycles in the graph, or equivalently, no flow-generating cycles in G_g ($\gamma(C) > 1$). Finally, we defined a generalized augmenting path (GAP) as consisting of a flow-generating cycle in G_g with a connecting path to the sink t (possibly trivial) and proved the following theorem:

Theorem 12.1 The following are equivalent for a circulation g:

1. g is an optimal circulation.
2. G_g has no GAPs.
3. There exists a labeling μ such that $\gamma_{ij}^\mu = \gamma_{ij}\mu_i/\mu_j \leq 1, \forall (i, j) \in A_g$.

How can we solve this problem? We'll look at a primal-dual style algorithm which decouples GAPs by (1) pushing flows along flow-generating cycles to create excesses at nodes, and (2) pushing these excesses to the sink. To accomplish the first aim, we use a min mean-cost cycle canceling algorithm on costs $c_{ij} = -\log \gamma_{ij}$ as in the min-cost circulation problem. In this case, however, the costs are no longer integral, so new (weaker) termination criteria are needed (see Problem Set 3). For now, we state without proof:

Claim 12.2 We may cancel all flow-generating cycles in time $O(m^2 n^3 \log(nB))$.

Once all flow-generating cycles have been cancelled, we may compute canonical labels using the Bellman-Ford shortest path algorithm as described earlier. Let c_i be the length of the shortest path from i to the sink t using edge costs c_{ij}. Then setting $\mu_i = e^{-c_i}$ gives

$$\begin{aligned} c_j &\leq c_i + c_{ij} \\ -\log \mu_j &\leq -\log \mu_i - \log \gamma_{ij} \\ \log \gamma_{ij} + \log \mu_i - \log \mu_j &\leq 0 \\ \gamma^{\mu}_{ij} = \frac{\gamma_{ij}\mu_i}{\mu_j} &\leq 1. \end{aligned}$$

Thus for all $(i,j) \in A_g$ we have that $\gamma^{\mu}_{ij} \leq 1$. Thus, if we can move the excesses to the sink node t, then we will have an optimal circulation satisfying the third criterion by the theorem above. The following algorithm as described .

Truemper's algorithm (1977)

```
Cancel flow-generating cycles
While $\exists e^g_i > 0$ that can reach $t$ in $G_g$
    Compute canonical labels $\mu$
    Compute max flow $f$ that pushes flow from $\{i \in e^g_i > 0\}$ in graph
        $(V, \{(i,j) \in A_g : \gamma^{\mu}_{ij} = 1\}, u^g_{ij})$
    $g^{\mu}_{ij} \leftarrow g^{\mu}_{ij} + f_{ij}$
```

Observe that pushing flow along paths of unit gain cannot create new flow generating cycles. To see this, observe that if we push flow along arc (i,j) with $\gamma^{\mu}_{ij} = 1$, we may create a new arc (j,i) in the residual graph; however, its gain will be $\gamma^{\mu}_{ji} = 1/\gamma^{\mu}_{ij} = 1$. Thus all arcs (i,j) in the residual graph will continue to have relabeled gain of value at most 1, and hence there can be no flow-generating cycle.

There is one remaining issue that we will not address; namely, the algorithm needs to be able to handle situations where excesses cannot reach the sink. We will assume that we can "undo" the creation of such excesses.

We thus assume the correctness of the algorithm and seek to prove bounds on its running time. The following lemma about Truemper's algorithm helps us accomplish this:

Lemma 12.3 The number of iterations of the while loop is no more than the number of different gains of paths.

Proof: After augmentation, there are no remaining augmenting paths on edges with $\gamma^{\mu}_{ij} = 1$. Any augmenting path must only use arcs (i, j) such that $\gamma^{\mu}_{ij} < 1$. Therefore, the gains for paths from excess nodes to the sink must decrease and thus the new canonical labels are necessarily smaller than the old canonical labels. This proves the lemma. □

Given this lemma, we just need to make sure that the number of different possible gains is polynomially bounded. In general, though, this is not the case.

12.1.2 Gain scaling

However, we can force the desired condition by modifying the gains so that there are only a polynomial number of different gains of paths by rounding the reduced gains as follows. Let $b = (1 + \epsilon)^{1/n}$. Then define

$$\begin{aligned}\overline{\gamma}_{ij} &= b^{\lfloor \log_b \gamma^{\mu}_{ij} \rfloor} \\ \overline{\gamma}_{ji} &= 1/\overline{\gamma}_{ij}, \forall (i,j) \in A_g.\end{aligned}$$

Note that rounding down is consistent for both $\overline{\gamma}_{ij}$ and $\overline{\gamma}_{ji}$ since both can only be present in the residual graph if both are equal to 1 (given that we have cancelled all flow-generating cycles).

How many different gain values of paths are now possible? We can bound the gain of a path P by

$$B^{-n} \leq \overline{\gamma}(P) \leq B^n,$$

and given that all gains are powers of b, then only

$$O(\log_b B^{2n}) = O(n \log_b B) = O(n^2 \log_{(1+\epsilon)} B)$$

paths with different gains are possible. Thus, if we let H denote a network with gains $\overline{\gamma}$, then we may use the Truemper algorithm to find an optimal flow h in H is polynomial time. To obtain an approximate solution to the original generalized flow problem, we may interpret h in G as

$$g_{ij} = \begin{cases} h_{ij} & \text{if } h_{ij} \geq 0 \\ -\gamma_{ji} h_{ji} & \text{if } h_{ij} < 0. \end{cases}$$

Finally, we may obtain the bounds on the approximation found.

Definition 12.1 A flow g is ϵ-*optimal* if for an optimal flow g^*, $|g| \geq (1 - \epsilon)|g^*|$.

Theorem 12.4 For an optimal flow h in H, its interpretation in G is ϵ-optimal.

Proof: Let g^* be the optimal flow in G. What is its value in H? For each path P pushing δ units of excess to the sink t gives $\gamma(P)\delta$ units at the sink. In H, the same path gives

$$\begin{aligned}
\overline{\gamma}(P) &\geq \frac{\gamma(P)\delta}{b^{|P|}} \\
&\geq \frac{\gamma(P)\delta}{b^n} \\
&\geq \frac{\gamma(P)\delta}{1+\epsilon} \\
&\geq \gamma(P)(1-\epsilon)\delta
\end{aligned}$$

units of flow at the sink. Thus, the total flow pushed to the sink in H by g is

$$\begin{aligned}
\sum_P \overline{\gamma}(P)\delta_P &\geq \sum_P \gamma(P)\delta_P(1-\epsilon) \\
&= (1-\epsilon)|g^*|
\end{aligned}$$

so the optimal flow h must have value greater than $(1-\epsilon)|g^*|$ in the network H. Since the gains in G are only larger than those in H, the interpretation of h in G will only have larger value, and thus is at least $(1-\epsilon)|g^*|$. □

This gives a polynomial time ϵ-optimal approximation algorithm for the generalized flow problem.

12.1.3 Error scaling

In the context of the approximation algorithm discussed in the previous section, we give the following theorem without proof:

Theorem 12.5 Given a B^{-4m} optimal flow, we can compute an optimal circulation in $O(CC + MF)$ time where $O(CC)$ is the time required for cycle cancelling and $O(MF)$ is the time required for a maximum flow computation.

Setting $\epsilon = B^{-4m}$, we can obtain a B^{-4m}-approximation using the Truemper algorithm with gain scaling. Unfortunately, this method gives an $O(n^2 \log_{1+\epsilon} B)$ running time which is polynomial in B, since $\log_{1+\epsilon} B$ for $\epsilon =^{-4m}$ is $O(B^{4m} \log B)$. This is expoenential in the size of the input. It is possible, however, to modify the Truemper gain scaling approach to derive an actual polynomial time algorithm for computing exact generalized flows. To do this, we introduce the following algorithm.

Iterated Rounded Truemper (Tardos & Wayne 1998)

```
$g \leftarrow 0$
For $i \leftarrow 1$ to $\log_2 B^{4m}$
    $g \leftarrow$Cancel cycles in $G_g$
    $g \leftarrow$Rounded Truemper $(G_g, \frac{1}{2})$
```

Rounded Truemper(G, ϵ)

```
Round down gains to $(1+\epsilon)^{1/n}$ to get graph $H$
$h \leftarrow$Truemper$(H)$
Return interpretation of $h$ in $G$
```

Theorem 12.6 For $\epsilon = \frac{1}{2}$, Rounded Truemper runs in $O(CC + (n^2 \log B)MF)$ time.

Proof: Trivial. $\square$

Theorem 12.7 Iterated Rounded Truemper computes a B^{-4m}-optimal flow in $O((m \log B)(CC + (n^2 \log B)MF))$ time.

Proof: The initial flow is 1-optimal. Each iteration finds a $\frac{1}{2}$-optimal flow in G_g, so the ith iteration is 2^{-i}-optimal. In $\log_2 B^{4m}$ iterations, the flow is B^{-4m}-optimal. $\square$

Thus, Iterated Rounded Truemper solves the generalized flow problem in polynomial time. The best known combinatorial algorithm for this problem runs in $\widetilde{O}(m^3 \log B)$ time, ignoring logarithmic factors in n (Goldfarb, Jin, and Orlin, 1997). No strongly polynomial has been discovered for this problem yet.

CS 361B Advanced Algorithms May 20, 2003

Lecture 13

Lecturer: David P. Williamson *Scribe: Charles-Henri Gros*

13.1 Multicommodity flow

After studying max-flow, min-cost cirulation and generalized flows, we now move on to an even more complex type of network problem: multicommodity flow.

This idea is once again to maximize flow, but this time we have several pairs of sources and sinks (each representing a different "commodity"), but the "pipes" (edges with capacities) are common.

Multicommodity flow

- **Input:**
  - A directed graph $G = \{V, A\}$
  - A set of k source-sink pairs: $\{(s^a, t^a), a = 1, \ldots, k\} \subseteq V^2$
  - a capacity function $u : A \to \mathbb{N}$
  - (optional) A set of k demands $d^a : 1..k \to \mathbb{N}$
- **Goal:** Find a set of functions $\{f^a : A \to \Re, a = 1, \ldots, k\}$ such that:
  - $\forall a = 1, \ldots, k$, f^a is a valid flow from s^a to t^a i.e. $\forall a \in 1, \ldots, k, \forall i \in V - \{s^a, t^a\}, \sum_{(i,j)\in A} f^a_{i,j} = \sum_{(j,i)\in A} f^a_{j,i}$
  - The total flow respects capacities i.e. $\forall (i,j) \in A, \sum_{a=1,\ldots,k} f^a_{i,j} \leq u_{i,j}$
  - The total flow is maximized. This flow is equal to $\sum_{a=1,\ldots,k} |f^a|$ where $|f^a| = \sum_{(s^a,j)\in A} f^a_{s^a,j} - \sum_{(j,s^a)\in A} f^a_{j,s^a}$

A "fair" version of the problem includes a set of k demands $d^a, a = 1, \ldots, k$, and the goal is to maximize $\lambda = \min_{a=1,\ldots,k} \frac{|f^a|}{d^a}$

13.1.1 A linear programming formulation

First we'll define these quantities:

- X_P the total flow along path P.

- $\mathcal{P}^a = \{\text{paths } P \text{ from } s^a \text{ to } t^a\}$
- $\mathcal{P} = \bigcup_{a=1..k} \mathcal{P}^a$

The LP formulation of the max commodity flow problem is the following:

$$\max \sum_{P \in \mathcal{P}} X_P$$

$$\forall (i,j) \in A, \sum_{P \in \mathcal{P}, (i,j) \in P} X_P \leq u_{i,j}$$

$$\forall P \in \mathcal{P}, X_P \geq 0.$$

The dual version of this LP is:

$$\min \sum_{(i,j) \in A} u_{i,j} l_{i,j}$$

$$\forall P \in \mathcal{P}, \sum_{(i,j) \in P} l_{i,j} \geq 1$$

$$\forall (i,j) \in A, l_{i,j} \geq 0.$$

In this program l might be viewed as an arc length function, in which case $\sum_{(i,j)\in P} l_{i,j}$ is the length of path P. Checking that all path lengths are at least 1 is equivalent to checking that the minimum length path from s^a to t^a is at least 1, and we know how to do that (since $l \geq 0$ there are no negative cost cycles).

13.1.2 An approximation algorithm

We give the following algorithm for the maximum multicommodity flow problem.

ε-approximate maximum multicommodity flow (Garg & Konemann 1998)

```
$X_P \leftarrow 0 \ \forall P \in \mathcal{P}$
$l_{i,j} \leftarrow \delta \ \forall (i,j) \in A$
while $\exists P \in \mathcal{P}$ s.t. $l(P) = \sum_{(i,j)\in P} l_{i,j} < 1$
    Pick $P$ such that $l(P) < 1$
    $u \leftarrow \min_{(i,j)\in P} u_{i,j}$
    $X_P \leftarrow X_P + u$
    $\forall (i,j) \in P, l_{i,j} \leftarrow l_{i,j}(1 + \varepsilon \frac{u}{u_{i,j}})$
Pick $M$ such that $\frac{X}{M}$ is feasible
return $\frac{X}{M}$
```

Note that this algorithm is quite different from previous flow algorithms that we have considered. We are not using the notion of a residual graph. Our solution X

while running the main loop is not even necessarily feasible; it is quite possible that the flow on an edge exceeds its capacity. Thus we scale down the flow at the end of the algorithm to ensure that the solution we return is a feasible flow.

First we show that the algorithm will terminate quickly.

Lemma 13.1 The algorithm terminates after at most $m \log_{1+\varepsilon} \frac{1+\varepsilon}{\delta}$ iterations.

Proof: Initially, $\forall (i,j) \in A, l_{i,j} = \delta$.

At no point in the algorithm is $l_{i,j} \geq 1+\varepsilon$. Indeed, $l_{i,j}$ only changes if it is in a path of length $l < 1$. Since all edges have positive length, this means that $l_{i,j} < 1$. Furthermore, $l_{i,j}$ is increased by a factor that is not above $1+\varepsilon$ (since by definition $u \leq u_{i,j}$) so it can't become greater than $1+\varepsilon$.

Also, at each iteration at least one edge has its length augmented by a factor of $1+\varepsilon$.

So after $m+1$ iterations, at least one edge has augmented by a factor of at least $(1+\varepsilon)^2$. After k iterations, at least one edge has augmented by a factor of at least $(1+\varepsilon)^{\lceil \frac{k}{m} \rceil}$ so has value at least $\delta(1+\varepsilon)^{\lceil \frac{k}{m} \rceil}$. Since this value is necessarily less than $1+\varepsilon$, if we set $i = \lceil \frac{k}{m} \rceil$, we have :

$$\delta(1+\varepsilon)^i < 1+\varepsilon$$

$$i < \log_{1+\varepsilon} \frac{1+\varepsilon}{\delta}$$

So :

$$k \leq mi < m \log_{1+\varepsilon} \frac{1+\varepsilon}{\delta}$$

□

We now show that if we scale the flow by a fixed quantity, the flow becomes feasible.

Lemma 13.2 If we scale flows f^a by $M = \log_{1+\varepsilon} \frac{1+\varepsilon}{\delta}$ then the total flow becomes feasible.

Proof: Fix an edge (i,j). At each iteration k, if $(i,j) \in P_k$ where P_k is the selected path, the flow on this edge (i,j) is increased by u_k. If we set $a_k = \frac{u_k}{u_{i,j}} \leq 1$, the length $l_{i,j}$ is increased by a factor of $1 + a_k\varepsilon$. At the end, $l_{i,j}$ is increased by a factor of $\prod_{k:(i,j)\in P_k}(1 + a_k\varepsilon)$. The flow on these edges, on the other hand, is increased by $\sum_{k:(i,j)\in P_k} u_k = u_{i,j} \sum_{k:(i,j)\in P_k} a_k$, starting from 0. Since initially $l_{i,j} = \delta$, and at the end $l_{i,j} < 1+\varepsilon$, we have

$$\delta \prod_{k:(i,j)\in P_k} (1 + a_k\varepsilon) < 1+\varepsilon.$$

Since $a_k \leq 1, 1 + a_k\varepsilon \geq (1+\varepsilon)^{a_k}$ so that

$$\delta(1+\varepsilon)^{\sum_{k:(i,j)\in P_k} a_k} < 1 + \varepsilon$$

$$\sum_{k,(i,j)\in P_k} a_k < \log_{1+\varepsilon} \frac{1+\varepsilon}{\delta} = M.$$

Thus since the total amount of flow on edge (i,j) is $u_{i,j}\sum_{k:(i,j)\in P_k} a_k$, if we divide the flows by M, the total amount of flow on edge (i,j) will be no more than $u_{i,j}$, and the flow will be feasible. $\square$

Theorem 13.3 The algorithm computes a $1-2\varepsilon$ approximate flow.

Proof: A few definitions:

- For length function l, we'll set $D(l) = \sum_{(i,j)\in A} u_{i,j} l_{i,j}$ (dual objective function) and $\alpha(l) = \min_{P\in\mathcal{P}} l(P)$.
- we'll note l^k the length function at the end of iteration k.
- we'll also note $D(k) = D(l^k)$ and $\alpha(k) = \alpha(l^k)$.
- we'll set $\beta = \min_{l \text{ feasible}} D(l) = \min_{l\geq 0, \alpha(l)\neq 0} \frac{D(l)}{\alpha(l)}$. This equality comes from the fact that if you divide a positive length function by its corresponding shortest path, the new shortest path becomes 1 so the length function becomes feasible.
- we'll set $X^k = \sum_{P\in\mathcal{P}} X_P^k$, primal value at the end of iteration k.
- t is the index of the last iteration.

By definition of t, $1 \leq \alpha(t)$. We'll assume for now (we'll prove it later) that

$$\alpha(t) \leq \delta n e^{\varepsilon \frac{X^t}{\beta}}$$

We then have:

$$\frac{X^t}{\beta} \geq \frac{\ln(\frac{1}{\delta n})}{\varepsilon}$$

We now want to show that:

$$\frac{X^t}{M} \geq (1-2\varepsilon)\beta$$

We will set $\delta = (1+\varepsilon)((1+\varepsilon)n)^{-\frac{1}{\varepsilon}}$. This value is chosen so that $\frac{\ln(\frac{1}{\delta n})}{M} = (1-\varepsilon)\ln(1+\varepsilon)$.

$$\begin{aligned}\frac{X^t}{M\beta} &\geq \frac{\ln(\frac{1}{\delta n})}{M\varepsilon} \\ &\geq \frac{(1-\varepsilon)\ln(1+\varepsilon)}{\varepsilon} \\ &\geq \frac{(1-\varepsilon)(\varepsilon - \varepsilon^2/2)}{\varepsilon} \\ &\geq (1-2\varepsilon).\end{aligned}$$

Now we want to show that:

$$\alpha(t) \leq \delta n e^{\varepsilon \frac{X^t}{\beta}}$$

To do this, we consider how the dual objective function changes from iteration to iteration. For an arbitrary iteration k,

$$\begin{aligned} D(k) &= \sum u_{i,j} l_{i,j}^k \\ &= D(k-1) + \sum_{(i,j)\in P_k} u_{i,j} l_{i,j}^{k-1}(1 + \varepsilon \frac{u}{u_{i,j}} - 1) \\ &= D(k-1) + \varepsilon u \sum_{(i,j)\in P_k} l_{i,j}^{k-1} \\ &= D(k-1) + \varepsilon (X^k - X^{k-1})\alpha(k-1). \end{aligned}$$

Thus we have in the final iteration t that

$$D(t) = D(0) + \varepsilon \sum_{k=1}^{t} (X^k - X^{k-1})\alpha(k-1).$$

We are looking for a bound on β. If we consider length function $l^t - l^0$, since β is minimum we have

$$\beta \leq \frac{D(l^t - l^0)}{\alpha(l^t - l^0)}$$

D is linear so $D(l^t - l^0) = D(t) - D(0)$. Now, $\alpha(l^t - l^0)$ is the length of some path P. $\alpha(l^t - l^0) = l^t(P) - l^0(P) \geq \alpha(t) - \delta n$ since l^0 is constant equal to δ on every edge, and P has less than n edges. Thus we have that

$$\begin{aligned} \beta &\leq \frac{D(t) - D(0)}{\alpha(t) - \delta n} \\ &\leq \frac{\varepsilon \sum_{k=1}^{t} (X^k - X^{k-1})\alpha(k-1)}{\alpha(t) - \delta n}. \end{aligned}$$

Rearranging terms, we have that

$$\beta(\alpha(t) - \delta n) \leq \varepsilon \sum_{k=1}^{t} (X^k - X^{k-1})\alpha(k-1),$$

or that

$$\alpha(t) \leq \delta n + \frac{\varepsilon}{\beta \sum_{k=1}^{t} (X^k - X^{k-1})\alpha(k-1)}.$$

Let $\alpha'(k)$ be the maximum possible value of $\alpha(k)$ given the above equation, for $1 \leq k \leq t$, and $\alpha'(0) = \delta n$. Then we have that

$$\begin{aligned}
\alpha'(0) &= \delta n \\
\alpha'(1) &= \delta n + \frac{\varepsilon}{\beta}(X^1 - X^0)\alpha'(0) = \delta n(1 + \frac{\varepsilon}{\beta}(X^1 - X^0)) \\
\alpha'(2) &= \delta n + \frac{\varepsilon}{\beta}((X^2 - X^1)\alpha'(1) + (X^1 - X^0)\alpha'(0)) \\
&= \alpha'(0)(1 + \frac{\varepsilon}{\beta}(X^1 - X^0)) + \frac{\varepsilon}{\beta}(X^2 - X^1)\alpha'(1) \\
&= \alpha'(1)(1 + \frac{\varepsilon}{\beta}(X^2 - X^1)).
\end{aligned}$$

In general we obtain that

$$\begin{aligned}
\alpha'(k) &= \alpha'(k-1)(1 + \frac{\varepsilon}{\beta}(X^k - X^{k-1})) \\
&\leq \alpha'(k-1)e^{\frac{\varepsilon}{\beta}(X^k - X^{k-1})}.
\end{aligned}$$

Then applying the bound repeatedly, we get that

$$\alpha'(k) \leq \alpha'(0)e^{\frac{\varepsilon}{\beta}(X^k - X^0)}.$$

So then

$$\alpha(t) \leq \alpha'(t) \leq \alpha'(0)e^{\frac{\varepsilon}{\beta}(X^t - X^0)}$$

Since $X^0 = 0$ and $\alpha'(0) = \delta n$,

$$\alpha(t) \leq \delta n e^{\frac{\varepsilon}{\beta}X^t}.$$

□

CS 361B Advanced Algorithms May 22, 2003

Lecture 14

Lecturer: David P. Williamson *Scribe: Damon Mosk-Aoyama*

14.1 Market equilibria

In this lecture, we consider a problem from economics: that of finding prices that will cause a market to clear. We refer to this problem as the Market-Clearing Pricing Problem.

Market-Clearing Pricing Problem

- **Input:**
  - Set B of buyers
  - Set A of unit amounts of divisible goods ($|A| = n$)
  - Integer amount of money m_i $\forall i \in B$
  - Integer utilities u_{ij} $\forall i \in B$, $\forall j \in A$
    (utility u_{ij} specifies the happiness buyer i derives from one unit of good j)
- **Goal:** Find prices p_j $\forall j \in A$ such that the *market clears*:
  - All buyers buy only goods that maximize happiness
  - All money is spent
  - No goods remain unpurchased

It has been long known that prices exist that clear the market. A result of Arrow and Debreu from 1954 implies the existence of market-clearing prices, though this may not be earliest work that establishes the existence of such prices. The previous proofs that market-clearing prices exist, however, were non-constructive.

The Market-Clearing Pricing Problem was defined in 1891 by Fisher, who invented a hydraulic machine to solve it (in the case of three goods). Recently, in 2002, a polynomial-time algorithm was given for the problem, demonstrating that there still exist nice problems, which are solvable in polynomial time, for which no polynomial-time algorithm was previously known. We present this algorithm for computing market-clearing prices, which was developed by Devanur, Papadimitriou, Saberi, and Vazirani.

14.1.1 Characterizing market clearance using flow

First, we formalize the notion that all the buyers must buy only goods that maximize their happiness in order for the market to clear. Given prices p_j, the "bang per buck" that a buyer i derives from a good j is the ratio of the utility u_{ij} to the price p_j. Figure 14.1(a) depicts sample data and the corresponding bang per buck ratios. Buyers try to maximize the bang per buck they get for the goods that they buy, and so we define α_i as follows to represent the best bang per buck that a buyer i can obtain.

$$\alpha_i = \max_{j \in A} \frac{u_{ij}}{p_j}$$

A buyer i will only buy goods j such that $\frac{u_{ij}}{p_j} = \alpha_i$. We define a graph that represents the goods that each buyer may purchase.

Definition 14.1 The *equality subgraph* $G = (A, B, E)$ is a bipartite graph (with vertex sets A and B) where $(i, j) \in E$ if and only if $\alpha_i = \frac{u_{ij}}{p_j}$.

Given a particular set of prices p_j $\forall j \in A$, we can determine whether the prices clear the market by performing a maximum flow computation. We add a source vertex s and a sink vertex t to the equality subgraph. For each good $j \in A$, we add an arc (s, j) with capacity p_j. For each buyer $i \in B$, we add an arc (i, t) with capacity m_i. We orient each edge (i, j) corresponding to a buyer $i \in B$ and a good $j \in A$ in the equality subgraph as a directed arc (j, i) with capacity ∞. Figure 14.1(b) shows an example of this graph for a particular collection of buyers and goods.

In this graph, flow from the source to the sink represents the transfer of money in the market. A unit of flow on an arc (j, i) from a good j to a buyer i represents a dollar spent by buyer i on good j. The total amount of flow from the source to the sink is the total amount of money spent by the buyers on goods. Therefore, the market clears (the buyers spend all their money) if and only if the maximum flow value is $\sum_{i \in B} m_i$.

14.1.2 An algorithm

The idea behind this algorithm for the Market-Clearing Pricing Problem is to start with small prices, and to raise the prices over the course of the execution of the algorithm. We will keep the prices sufficiently low to ensure that all the goods are sold, but the buyers have left-over money (a surplus). We will maintain the invariant that the singleton set $\{s\}$ is a minimum s-t cut. The goal will be to find prices such that $V - \{t\}$ is also a minimum s-t cut, because the capacity of the arcs crossing this cut is the total amount of money the buyers have. When this cut becomes a minimum s-t cut, the value of the maximum flow is $\sum_{i \in B} m_i$, and the market clears. We raise the prices gradually, decreasing the surplus of the buyers until it reaches zero.

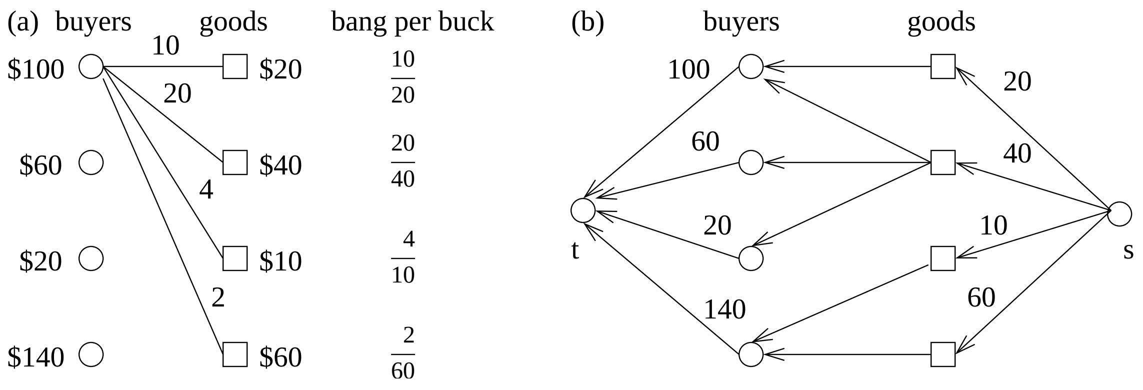


Figure 14.1: (a) An example of the computation of the bang per buck that a buyer obtains from different goods. The amounts of money the buyers have are shown on the left, the prices of the goods are shown on the right, and the label for an edge (i, j) indicates the utility u_{ij}. (b) A graph in which we can compute a maximum flow to determine whether a set of prices clears a market. The arcs from goods to buyers have infinite capacity.

Initialization of Prices

We want to assign small initial values to the prices to ensure that $\{s\}$ is a minimum s-t cut. To initialize the prices, we set $p_j = \frac{1}{n}$ $\forall j \in A$. Under these prices, $\{s\}$ is a minimum s-t cut with value 1. We also need at least one buyer for each good. If there are no buyers for good j, we compute $\alpha_i = \max_{j \in A} \frac{u_{ij}}{p_j}$ for all buyers i. Then, we reduce the price p_j to the value $\max_{i \in B} \frac{u_{ij}}{\alpha_i}$.

Raising Prices

When we raise the prices to decrease the surplus of the buyers, we would like to ensure that all edges remain in the equality subgraph. Consider a buyer i for which the edges (i, j) and (i, k) are both in the equality subgraph. By the definition of the equality subgraph, we have $\frac{u_{ij}}{p_j} = \frac{u_{ik}}{p_k}$, which implies that $\frac{p_k}{p_j} = \frac{u_{ik}}{u_{ij}}$. Multiplying both p_j and p_k by the same factor will leave this ratio unchanged. As such, we increase the prices from p_j to p'_j by setting $p'_j = p_j x$ $\forall j \in A$ for some factor x.

To determine the factor x that we will use to raise the prices, we consider the different ways in which the equality subgraph may change when we raise the prices.

- **Event type (1)**: By increasing x, the invariant that $\{s\}$ is a minimum s-t cut becomes violated.

  In the previous example, multiplying the prices by the factor $x = 2$ causes another minimum s-t cut to emerge, as shown in Figure 14.2(a). If we multiply the prices by a factor $x > 2$, then we violate the invariant, because $\{s\}$ is no longer a minimum s-t cut.

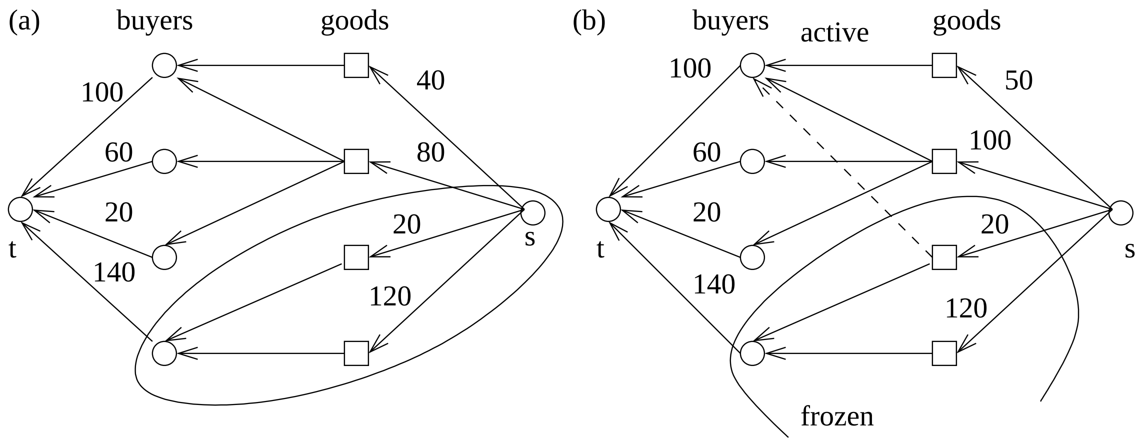


Figure 14.2: (a) An example of event type (1). If the prices are multiplied by a factor $x > 2$, then the cut shown becomes the minimum s-t cut. (b) An example of event type (2). Multiplying the prices of the active goods in (a) by a factor $x = 1.25$ causes the dashed edge shown between an active buyer and a frozen good to enter the equality subgraph.

Note that in the example, the emergence of the new minimum s-t cut when the prices are raised creates a desirable scenario for the last buyer, because all the money available to that buyer can be spent on goods. In general, the market clears in the subgraph involved in the new minimum s-t cut. As a result, we can "freeze" the subgraph involved in the cut, and consider only the remaining graph when we raise the prices again. At any point in the algorithm, we refer to the subgraph in which we are increasing the prices as *active*, and to the rest of the graph as *frozen*.

- **Event type (2)**: A new edge from an active buyer to a frozen good enters the equality subgraph.

  Continuing the example from above, if we take the prices that caused the event of type (1) to occur and multiply the prices for the active goods by $x = 1.25$, then an edge between an active buyer and a frozen good is created in the equality subgraph, as shown in Figure 14.2(b). To address this type of event, we unfreeze the good incident on the new edge, and the connected component containing the good.

Analysis and Description of Algorithm

Definition 14.2 For $S \subseteq A$, $\Gamma(S) = \{i \in B \mid \exists j \in S : (i,j) \in E\}$.

For a subset $A' \subseteq A$ of goods, let $p(A') = \sum_{j \in A'} p_j$. Similarly, for a subset $B' \subseteq B$ of buyers, let $m(B') = \sum_{i \in B'} m_i$.

Lemma 14.1 The invariant that $\{s\}$ is a minimum s-t cut holds if and only if $\forall S \subseteq A$, $p(S) \leq m(\Gamma(S))$.

Proof: ($\Rightarrow$) If the invariant holds, then the maximum-flow minimum-cut theorem implies that the maximum flow value is $p(A)$. The maximum flow can have this value only if all the arcs incident on s are at capacity, and so, for any $S \subseteq A$, the maximum flow ships $p(S)$ units of flow from the source to $\Gamma(S)$. At most $m(\Gamma(S))$ units of flow can be shipped from $\Gamma(S)$ to the sink, and thus flow conservation implies that $p(S) \leq m(\Gamma(S))$.

($\Leftarrow$) Consider any s-t cut $\{s\} \cup A_1 \cup B_1$ other than $\{s\}$, where $A_1 \subseteq A$ and $B_1 \subseteq B$. Let $A_2 = A - A_1$ and $B_2 = B - B_1$. Since the arcs in the cut are those from s to vertices in A_2 and those from vertices in B_1 to t, the value of the cut is $p(A_2) + m(B_1)$. We must have $\Gamma(A_1) \subseteq B_1$, because if this were not true, the cut would contain an arc from a vertex in A_1 to a vertex in B_2, and all such arcs have infinite capacity. This implies that $m(\Gamma(A_1)) \leq m(B_1)$. By assumption, $p(A_1) \leq m(\Gamma(A_1))$, and so we can combine these inequalities to obtain $p(A_1) \leq m(B_1)$. Now, the s-t cut $\{s\}$ has value $p(A_1) + p(A_2) \leq m(B_1) + p(A_2)$. This shows that the value of the cut $\{s\}$ is at most the value of any other cut, and therefore $\{s\}$ is a minimum s-t cut. $\square$

Suppose that we multiply the prices by a factor x^* that causes a minimum s-t cut other than $\{s\}$ to emerge. For factors $x > x^*$, multiplying the prices by x will cause the invariant to become violated. Lemma 14.1 implies that for some $S \subseteq A$, $x^* \cdot p(S) = m(\Gamma(S))$. We call a set $S \subseteq A$ that satisfies this equality *tight*.

We now state the algorithm for the Market-Clearing Pricing Problem.

Market-Clearing Prices (Devanur, Papadimitrou, Saberi, Vazirani 2002)

```
$p_j \leftarrow \frac{1}{n}$  $\forall j \in A$
Compute $\alpha_i = \max_{j \in A} \frac{u_{ij}}{p_j}$  $\forall i \in B$
For each $j \in A$ such that $\nexists i \in B : (i,j) \in E$, $p_j \leftarrow \max_{i \in B} \frac{u_{ij}}{\alpha_i}$
$(F, F') \leftarrow (\emptyset, \emptyset)$ (frozen graph)
$(H, H') \leftarrow (A, B)$ (active graph)
While $H \neq \emptyset$
    Raise prices $p_j \leftarrow p_j x$  $\forall j \in H$ until either:
    (1) $S \subseteq H$ becomes tight
        Move $(S, \Gamma(S))$ from $(H, H')$ to $(F, F')$
        Remove edges from $F'$ to $H$
    (2) For $i \in H'$, $j \in F$   $\alpha_i = \frac{u_{ij}}{p_j}$
        Add $(i, j)$ to $E$
        Move connected component containing $j$ from $(F, F')$ to $(H, H')$
Return $p_j$  $\forall j \in A$.
```

There are several outstanding issues that we must address. First, can we implement the steps of the algorithm? Second, how long does the algorithm take?

We begin to answer the first question here. The main challenge in implementing this algorithm as stated lies in determining the minimum factor x for the prices that will cause an event of one of the two types to occur. To compute this quantity, we compute the minimum value of x that will cause an event of type (1) to occur, and the minimum value of x that will cause an event of type (2) to occur, and we take the smaller of these two values. We claim that events of type (2) can be detected easily. In the next lecture, we will continue the analysis of this algorithm for the Market-Clearing Pricing Problem by proving the following lemma.

Lemma 14.2 A value of x such that an event of type (1) happens can be determined with n maximum-flow computations.

CS 361B Advanced Algorithms May 27, 2003

Lecture 15

Lecturer: David P. Williamson *Scribe: Paat Rusmevichientong*

15.1 Market equilibria (cont.)

Recall the market-clearing pricing problem we introduced last time:

Market-Clearing Pricing Problem

- **Input:**
  - Set B of buyers
  - Set A of unit amounts of divisible goods ($|A| = n$)
  - Integer amount of money m_i, $\forall i \in B$
  - Integer utilities u_{ij}, $\forall i \in B$, $\forall j \in A$
    (u_{ij} = happiness for buyer i from one unit of good j)
- **Goal:** Find prices p_j, $\forall j \in A$ such that the market clears:
  - All buyers buy only goods that maximize happiness
  - All money is spent
  - No good remains unpurchased

Let α_i denote the maximum "bang-per-buck" that the buyer i can receive, i.e.

$$\alpha_i = \max_{j \in A} \frac{u_{ij}}{p_j}.$$

The buyer i will purchase only goods j such that $\alpha_i = u_{ij}/p_j$. Given the prices of the goods, we can define an equality subgraph that represents the goods that each buyer may purchase.

Definition 15.1 The equality subgraph $G = (A, B, E)$ is a bipartite graph (with vertex set A and B) where $(i, j) \in E$ if and only if $\alpha_i = u_{ij}/p_j$.

Given a particular set of prices $p_j, j \in A$, we can determine whether the prices clear the market by performing a maximum flow computation. We add a source vertex s and a sink vertex t to the equality subgraph. For each good $j \in A$, we add an arc (s, j) with capacity p_j. For each buyer $i \in B$, we add an arc (i, t) with capacity m_i. We orient each edge (i, j) corresponding to a buyer $i \in B$ and a good $j \in A$ in the equality subgraph as a directed arc (j, i) with capacity ∞. In the previous lecture, we showed that the market clears if and only if the maximum flow value is $m(B) \equiv \sum_{i \in B} m_i$.

Our algorithm maintains the invariant that $\{s\}$ is a minimum s-t cut. We showed that the following.

Lemma 15.1 The invariant that $\{s\}$ is a minimum s-t cut holds if and only for all $S \subseteq A$, $p(S) \leq m(\Gamma(S))$.

The algorithm divides the graph into "frozen" and "active" subgraphs. We then increase the prices p_j in the "active" subgraph by a factor x. When there exists a set $S \subseteq A$ such $x \cdot p(S) = m(\Gamma(S))$, we call such set S *tight*.

15.1.1 Running time analysis

Here's the algorithm we introduced last time for the Market-Clearing Pricing Problem.

Market-Clearing Pricing (Devanur, Papadimitrou, Saberi, Vazirani 2002)

```
Price Initialization
$(F, F') \leftarrow (\emptyset, \emptyset)$ ("frozen")
$(H, H') \leftarrow (A, B)$ ("active")
While $H \neq \emptyset$
    Raise prices $p_j \leftarrow p_j \cdot x$, $\forall j \in H$ until either:
    (1) $S \subseteq H$ becomes tight
        Move $(S, \Gamma(S))$ from $(H, H')$ to $(F, F')$
        Remove edges from $F'$ to $H$
    (2) For $i \in H'$, $j \in F$, $\alpha_i = u_{ij}/p_j$
        Add $(i, j)$ to $E$
        Move the connected component of $j$ from $(F, F')$ to $(H, H')$
Return $p_j$, $\forall j \in A$
```

To execute the main loop of the algorithm, we find a value of the value of x such that (1) happens and such that (2) happens, then take the minimum x of the two. One can easily find the value of x such that (2) happens; we now show how to find the value of x such that (1) happens.

Lemma 15.2 At each iteration, we can determine x such that (1) happens using n max-flow computations.

Proof: Without loss of generality, we may assume that (A, B) is active. The same argument applies to arbitrary active subgraph. To determine such x, we need to determine

$$x^* \equiv \min_{\emptyset \neq S \subseteq A} \frac{m(\Gamma(S))}{p(S)}.$$

Let S^* denote the set that minimizes the above ratio.

We will start with $x \equiv m(B)/p(A) \geq x^*$, and compute max-flow for prices $x \cdot p_j$. If $\{s\}$ turns out to be a min s-t cut, then by Lemma 15.1 we know that $x = x^*$ and we're done. Furthermore, we can determine S^* by taking the maximum minimum cut (i.e., the largest set S such that S is an s-t min cut). The maximum minimum cut can easily be determined from the residual graph produced by maximum s-t flow algorithms.

If $x > x^*$ and $\{s\}$ is not a min s-t cut, let $\{s\} \cup A_1 \cup B_1$ be the min s-t cut. If we can show that $S^* \subseteq A_1 \subset A$, then the lemma is proven because we can recurse on $(A_1, \Gamma(A_1))$.

Claim 1: $A_1 \subset A$. If $A_1 = A$, then we must have $B_1 = B$ because the edges between A and B have infinite capacity. But, the cut $\{s\} \cup A \cup B$ has value $m(B)$ while the cut $\{s\}$ has value $x \cdot p(A)$, and we have $x \cdot p(A) \leq m(B)$. This implies that $\{s\}$ is a min s-t cut, contradicting our assumption. Therefore, $A_1 \subset A$.

Claim 2: $S^* \subseteq A_1$. Let $S_1 = S^* \cap A_1$ and $S_2 = S^* \cap A_2$. Note that we must have $\Gamma(S_1) \subseteq B_1$ since otherwise the cut will have infinite capacity. Note that the value of the cut $\{s\} \cup A_1 \cup B_1$ is $x \cdot p(A_2) + m(B_1)$.

First observe that it cannot be the case that $m(\Gamma(S_2) \cap B_2) < x \cdot p(S_2)$. Otherwise consider the cut $\{s\} \cup A_1 \cup S_2 \cup B_1 \cup (\Gamma(S_2) \cap B_2)$. It has value $x(p(A_2) - p(S_2)) + m(B_1) + m(\Gamma(S_2) \cap B_2) < x \cdot p(A_2) + m(B_1)$, which contradicts the fact that $\{s\} \cup A_1 \cup B_1$ is a minimum cut.

Note that this observation implies that it cannot be the case that $S^* = S_2$ since then $x^* < x$ implies that $m(\Gamma(S^*) \cap B_2) \leq m(\Gamma(S^*)) < x \cdot p(S^*)$.

Thus $S_1 \neq \emptyset$. Furthermore, we have that

$$m(\Gamma(S_2) \cap B_2) \geq x \cdot p(S_2) > x^* \cdot p(S_2).$$

By the definition of x^*,

$$m(\Gamma(S_2) \cap B_2) + m(\Gamma(S_1)) \leq m(S^*) = x^*(p(S_1) + p(S_2)).$$

Subtracting the first inequality from the second we obtain that

$$m(\Gamma(S_1)) < x^* \cdot p(S_1),$$

which contradicts the definition of x^*. Thus it must be the case that $S_2 = \emptyset$. □

Now, to determine the computational complexity of our algorithm, we need the following lemma.

Lemma 15.3 For any item $j \in S$ where S is a tight set, then p_j has denominator less than or equal to $\Delta \equiv nU^n$, where $U \equiv \max_{i,j} u_{ij}$.

Proof: It is not hard to show that for any $k \in S$, $p_j = p_k \cdot \frac{a_k}{b_k}$, where a_k and b_k are products of the utilities. Now, for a tight set S, we have

$$m(\Gamma(S)) = p(S) = \sum_{k \in S} p_k = p_j \sum_{k \in S} \frac{a_k}{b_k},$$

which implies that

$$p_j = \frac{m(\Gamma(S))}{\sum_{k \in S} \frac{a_k}{b_k}}.$$

Thus, the denominator of p_j is the sum of at most n products of n utilities, which is bounded above by $\Delta = nU^n$. $\square$

Before we proceed to the main result of this section, we note that if the price p_j in iteration $i+k$ is *strictly greater* than the price p_j in iteration i, then the difference must be at least $1/\Delta^2$. This result follows from the fact that for any positive integers a, b, c, d, if $a/b > c/d$ and $b, d \leq \Delta$, then $(a/b) - (c/d) \geq 1/\Delta^2$.

Theorem 15.4 The algorithm for the Market-Clearing Pricing problem runs in time $O(m(B) \cdot n^2 \cdot \Delta^2 \cdot MF)$, where MF denotes the running time for computing max-flow.

Proof: By the lemma and observation above, each time good j is frozen because of event (1), its price p_j has increased by $1/\Delta^2$. Each time event (1) happens, some good's price has increased, so we assign it to this freezing. Thus after k executions of event (1), the total surplus is at most $m(B) - (k/\Delta^2)$. Thus event (1) can occur at most $m(B)\Delta^2$ times. There can be at most n iterations of the main loop in which event (2) occurs instead of event (1), and in each iteration we need to do n max flow computations to determine the appropriate value of x. The running time follows. $\square$

15.1.2 A polynomial time analysis

Though the algorithm above and its analysis are intuitive, unfortunately the analysis does not give a polynomial time algorithm. A modification of the algorithm gives a polynomial-time algorithm.

By modifying the algorithm so that for a given ϵ, $\forall S \subseteq H$,

$$m(\Gamma(S)) \geq p(S) + \epsilon,$$

and for each component S added to the frozen subgraph,

$$m(\Gamma(S)) \leq p(S) + |S|\epsilon,$$

we can show that the algorithm runs in time $O(MF \cdot n^2 \left(n \log U + \log(m(B)n^2)\right))$. The first condition implies that whenever event (1) happens, the total flow has increased by ϵ. The second condition implies that when the algorithm terminates, the remaining surplus is no more than $n\epsilon$. Thus we run the algorithm first with $\epsilon = m(B)/2n$; at termination, the surplus will be no more than $m(B)/2$. By halving ϵ and repeating, the surplus will be no more than $m(B)/2^i$ after the ith execution of the algorithm. After $O(\log(m(B)\Delta^2)) = O(n \log U + \log(m(B)n^2))$ executions of the algorithm, the surplus is less than $1/\Delta^2$. We can then run the algorithm once more with $\epsilon = 0$ (that is, we can run the original algorithm) to reduce the surplus to 0. This gives the running time stated above.

15.1.3 Open questions

It is an interesting open question whether the original algorithm can be shown to run in polynomial time, or even strongly polynomial time.

CS 361B Advanced Algorithms May 29, 2003

Lecture 16

Lecturer: David P. Williamson *Scribe: Damon Mosk-Aoyama*

16.1 Interior-point methods for linear programming

So far in this course, we have treated algorithms that solve linear programs as black boxes. We have assumed that we have an LP solver that will take a linear program as input, and return an optimal solution to the LP in (weakly) polynomial time. We now return to studying linear programming, and discuss methods for solving linear programs in polynomial time.

Recall the standard form of a linear program that we presented in Lecture 1:

$$\begin{aligned} \text{Min} \quad & c^T x \\ \text{subject to:} \quad & \\ & Ax = b \\ & x \geq 0. \end{aligned}$$

Let n denote the number of variables in the LP, so that $x \in \Re^{n \times 1}$. We can express the dual of this LP as follows, using slack variables $s \in \Re^{n \times 1}$:

$$\begin{aligned} \text{Max} \quad & b^T y \\ \text{subject to:} \quad & \\ & A^T y + s = c \\ & s \geq 0. \end{aligned}$$

The following are the algorithms for linear programming used in theory and practice.

- **Simplex**: Developed in 1947 by Dantzig, the Simplex algorithm is fast in practice, and is commonly used today. An example of an implementation of the Simplex algorithm is CPLEX. As we mentioned in Lectures 1 and 2, there is no known polynomial-time pivot rule for the Simplex algorithm. Many pivot rules that have been studied have been shown to have exponential behavior in the worst case.

- **Ellipsoid method**: The first polynomial-time algorithm for linear programming was the ellipsoid method, which is due to Khachiyan. The presentation of the ellipsoid method in 1979 was front-page news, even in mainstream publications such as *The New York Times*. Despite being an important algorithm in theory, the ellipsoid method is miserably slow in practice, and but is sometimes used for solving control theory problems when the amount of time needed to solve the program is not an issue.

- **Interior-Point methods**: The seminal work on interior-point methods was by Karmarkar in 1984. Since then, there has been a significant amount of follow-up work on interior-point methods. Interior-point methods are polynomial-time algorithms, and are fast in practice. For some linear programs, the best implementations of interior-point methods are faster than the best implementations of the Simplex algorithm.

In the remainder of this lecture, we study interior-point methods. Recall that the Simplex algorithm finds an optimal solution to a linear program by moving between vertices of the feasible region. It advances from one vertex to another by moving along a line defined by constraints of the LP. When it reaches a vertex, it performs a pivot, and then proceeds along another line, repeating this process until it reaches a vertex that is an optimal solution. As such, the Simplex algorithm only moves along the boundaries of the feasible region. In contrast, interior-point methods move around between points in the interior of the feasible region. Figure 16.1 illustrates this difference between the algorithms.

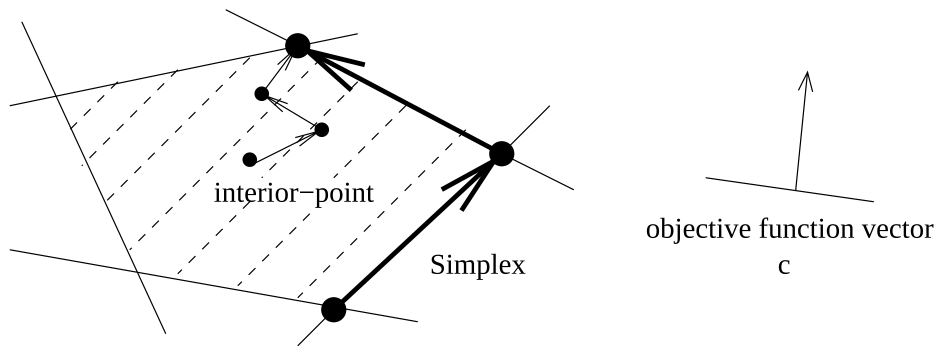


Figure 16.1: The Simplex algorithm moves along the boundaries of the feasible region. Interior-point methods find an optimal solution by moving around the interior of the feasible region.

Recall that if x is a primal feasible solution, (y, s) is a dual feasible solution, and the complementary slackness conditions $x^T s = 0$ are satisfied, then (x, y, s) is optimal for the primal and dual linear programs. Therefore, a solution to the following system is an optimal solution to the linear programs.

$$\begin{aligned}
&A^T y + s = c \\
&Ax = b \\
&x_i s_i = 0 \qquad \forall i = 1, \ldots, n \\
&x \geq 0 \\
&s \geq 0
\end{aligned} \tag{16.1}$$

The first two equations define a system that we can solve using techniques for solving linear systems. Because of the nonlinear complementary slackness equations and the nonnegativity constraints on x and s, however, we must use another approach to solve this system.

16.1.1 Newton steps

Our method for solving this system is based on Newton's method for finding a root of a function. In the one-dimensional, unconstrained case, we have a function $f(x)$, and we begin with an initial point x_0. We then repeatedly update the point. In iteration k, the tangent line to f at the current point x_k is described by $y = f'(x_k)(x - x_k) + f(x_k) = f'(x_k)\Delta x + f(x_k)$, where $\Delta x = x - x_k$. We find a value of Δx such that $y = 0$, and set $x_{k+1} \leftarrow x_k + \Delta x$. Figure 16.2 shows an example of an update in Newton's method. We repeat this process until the value $f(x_k)$ of the function at the current point is sufficiently close to zero.

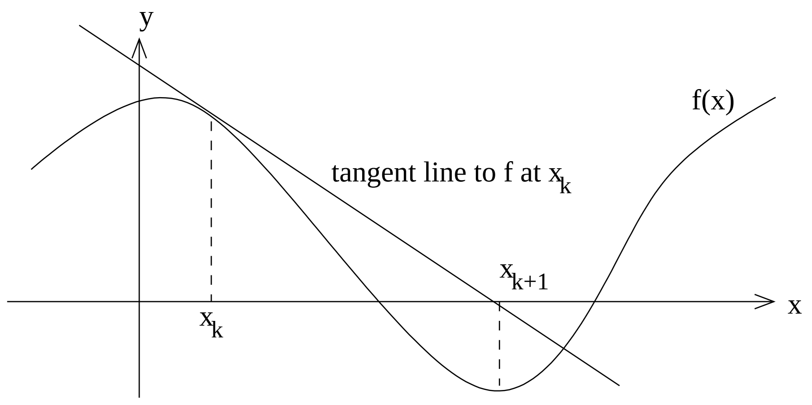


Figure 16.2: An update by Newton's method in the one-dimensional, unconstrained case.

Interior-point methods apply this same approach to the system (16.1). We define a function $F(x, y, s)$ of the primal and dual solutions (x, y, s) to the linear programs.

$$F(x, y, s) = \begin{pmatrix} A^T y + s - c \\ Ax - b \\ XSe \end{pmatrix}$$

$$X = \begin{pmatrix} x_1 & 0 & \dots & 0 \\ 0 & \ddots & 0 & \vdots \\ \vdots & 0 & \ddots & 0 \\ 0 & \dots & 0 & x_n \end{pmatrix}, \quad S = \begin{pmatrix} s_1 & 0 & \dots & 0 \\ 0 & \ddots & 0 & \vdots \\ \vdots & 0 & \ddots & 0 \\ 0 & \dots & 0 & s_n \end{pmatrix}, \quad e = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$$

Our goal is to find solutions (x, y, s) such that $F(x, y, s) = 0$ and $(x, s) \geq 0$. To do this, we make use of the Jacobian J, which is a matrix of partial derivatives.

$$J(x, y, s) = \begin{pmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S & 0 & X \end{pmatrix}$$

Given these definitons, a *Newton direction* $(\Delta x, \Delta y, \Delta s)$ is a solution to the following equation.

$$J(x, y, s) \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta s \end{pmatrix} + F(x, y, s) = 0$$

From now on, we will assume that we have feasible solutions (x, y, s) to the primal and dual linear programs. We will not discuss in detail how to find an initial feasible solution, though any algorithm based on interior-point methods must find an initial feasible solution that has the properties required for the implementation and analysis of the algorithm. Furthermore, we will assume that the solutions (x, y, s) are *strictly feasible*: $x > 0$ and $s > 0$. We will repeatedly update the solutions, eventually obtaining solutions that are closer to satisfying the complementary slackness conditions.

For feasible solutions (x, y, s) to the linear programs, $F(x, y, s)$ can be expressed as follows.

$$F(x, y, s) = \begin{pmatrix} A^T y + s - c \\ Ax - b \\ XSe \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ XSe \end{pmatrix}$$

This allows us to rewrite the equation defining a Newton direction.

$$J(x, y, s) \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta s \end{pmatrix} = -F(x, y, s)$$
$$\begin{pmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S & 0 & X \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta s \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -XSe \end{pmatrix}$$

This is a system of linear equations that can be solved in $O(n^3)$ time using Gaussian elimination to obtain a Newton direction $(\Delta x, \Delta y, \Delta s)$.

Given a set (x^0, y^0, s^0) of feasible solutions and a Newton direction $(\Delta x, \Delta y, \Delta s)$, we obtain a new set of solutions (x^1, y^1, s^1) by moving from (x^0, y^0, s^0) in the direction $(\Delta x, \Delta y, \Delta s)$. The solutions $(x^0, y^0, s^0) + (\Delta x, \Delta y, \Delta s)$ may not be strictly feasible, however. To ensure that our new solutions are strictly feasible, we set the new collection of solutions to be $(x^1, y^1, s^1) \leftarrow (x^0, y^0, s^0) + \alpha(\Delta x, \Delta y, \Delta s)$. We choose the scaling factor α such that $x^1 = x^0 + \alpha\Delta x > 0$, and $s^1 = s^0 + \alpha\Delta s > 0$.

16.1.2 The central path

A problem may arise if we always use a Newton direction to update our solutions. We may have to choose α to be a very small quantity in order to ensure that the next solutions are strictly feasible. If the current solutions are near a boundary of the feasible region, then we may be forced to use small values of α repeatedly, and so the solutions in consecutive iterations may be near each other. This will cause the number of iterations required for finding optimal solutions to be large. Since the running time of our iterative algorithm is directly proportional to the number of iterations it executes, we would like to choose the directions for the updates to ensure that the maximum value of α that produces strictly feasible solutions is sufficiently large.

We study two ideas for addressing this problem. First, we can bias the update steps towards the interior of the feasible region. Second, we consider a technique that will allow us to keep the update steps bounded away from the boundary of the feasible region.

Biasing Steps Towards Interior of Feasible Region

By modifying a Newton direction so that the update goes towards the interior of the feasible region, we can use larger values of α in our updates, enabling us to move further in individual iterations. Consider the following system:

$$\begin{aligned}
&A^T y + s = c \\
&Ax = b \\
&x_i s_i = \tau \qquad \forall i = 1, \dots, n \\
&x \geq 0 \\
&s \geq 0.
\end{aligned} \tag{16.2}$$

For $\tau = 0$, system (16.2) is the same as the system (16.1) above. If we solve system (16.2) for smaller and smaller values of τ, then, the solutions will approach the solution to system (16.1).

Definition 16.1 Denote the solutions to system (16.2) by $\{(x_\tau, y_\tau, s_\tau) \mid \tau > 0\}$. These solutions are called *the central path*.

It can be shown that the solutions in the central path exist and are unique, assuming A, b, and c satisfy appropriate properties.

The dual linear program has one constraint for each variable in the primal linear program. As such, for each primal variable x_i, the dual LP has a slack variable s_i whose value specifies the slack in the dual constraint corresponding to x_i. We can have $x_i s_i = 0$ only if $x_i = 0$ or $s_i = 0$. If $x_i = 0$, the primal constraint $x_i \geq 0$ is tight, whereas if $s_i = 0$, then the dual constraint corresponding to x_i has no slack and therefore is tight. In this sense, the quantity $x_i s_i$ measures how close the feasible primal and dual solutions (x, y, s) are to a pair of boundaries in the primal and dual feasible regions. If $x_i s_i = 0$, then at least one constraint in the primal and dual linear programs is tight. Since the constraints in a linear program define the boundaries of the feasible region, $x_i s_i = 0$ implies that the solutions (x, y, s) are on a boundary of either the primal or dual feasible region.

By requiring that $x_i s_i$ has the same value τ for all $i = 1, \ldots, n$, the system (16.2) ensures that solutions to it will be effectively at the same distance $x_i s_i = \tau$ from each of n pairs of boundaries in the primal and dual feasible regions. The solutions in the central path are solutions to system (16.2) with $\tau > 0$, and so they are in the interiors of the primal and dual feasible regions.

We will use Newton's method to find a solution to the system (16.2) with parameter value τ. Over the course of updating the solutions (x, y, s) to the linear programs, we will reduce τ, causing our solutions to converge to optimal solutions to the linear programs. Suppose that in some iteration, the current solutions are (x, y, s). We consider the choice $\tau = \frac{1}{n}\sum_{i=1}^{n} x_i s_i = \frac{1}{n}x^T s$; that is, given our current solution, solving the system for this value of τ makes the $x_i s_i$ equal for all i. Recall that $x^T s$ is the duality gap between the feasible solutions x and (y, s). We define $\mu \equiv \frac{1}{n}x^T s$.

To balance the movement towards the central path against the movement toward optimal solutions, we maintain a *centering parameter* $\sigma \in [0, 1]$. We will update our current solutions by moving towards a solution to the system (16.2) with $\tau = \sigma\mu$. We accomplish this by setting the update direction $(\Delta x, \Delta y, \Delta s)$ to be a solution to the following linear system.

$$\begin{pmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S & 0 & X \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta s \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -XSe + \sigma\mu e \end{pmatrix}$$

If $\sigma = 1$, then our update will move towards the center of the feasible region. On the other hand, if $\sigma = 0$, then our update step is in the direction of optimal solutions to the linear programs. A step with $\sigma = 1$ is referred to as a centering step, and a step with $\sigma = 0$ is referred to as an affine-scaling step. The choice of the centering parameter σ provides us with a trade-off between moving towards the central path and moving toward optimal solutions to the linear programs.

Some interior-point methods examine only the feasible region of the primal linear program, and as such are known as pure primal interior-point methods. The general interior-point algorithm that we present here examines the feasible regions of both the primal and the dual linear programs, and so we refer to it as a primal-dual interior-point algorithm.

Primal-Dual Interior-Point

```
$(x^0, y^0, s^0) \leftarrow$ initial feasible point $(x^0, s^0 > 0)$
$\mu^0 \leftarrow \frac{1}{n}(x^0)^T s^0$
$k \leftarrow 0$
While $\mu^k > \epsilon$
    Solve $\begin{pmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S^k & 0 & X^k \end{pmatrix} \begin{pmatrix} \Delta x^k \\ \Delta y^k \\ \Delta s^k \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -X^k S^k e + \sigma^k \mu^k e \end{pmatrix}$
    $(x^{k+1}, y^{k+1}, s^{k+1}) \leftarrow (x^k, y^k, s^k) + \alpha^k(\Delta x^k, \Delta y^k, \Delta s^k)$
       where $\alpha^k$ is such that $x^{k+1}, s^{k+1} > 0$
    $\mu^{k+1} \leftarrow \frac{1}{n}(x^{k+1})^T s^{k+1}$
    $k \leftarrow k + 1$
```

This general framework omits several details that must be addressed in any implementation of an interior-point algorithm for linear programming. In particular, we have not specified how the the centering parameter σ^k is chosen, as different interior-point algorithms use different methods to select σ^k. Furthermore, we have not considered how to set the threshold ϵ, find an initial solution to the LP, or find the optimal solution to the LP given the solution that the algorithm returns.

Keeping away from the boundary

During the course of iterating through a sequence of solutions to the linear programs, we can keep the solutions away from the boundary by ensuring that they remain in a *neighborhood* of the central path. Because solutions in the central path are essentially at the same distance from n boundaries of the feasible regions, by maintaining solutions near the central path, we can prevent them from approaching the boundaries of the feasible regions.

There are several common types of neighborhoods used by interior-point algorithms. For a parameter θ, a neighborhood that uses the L_2 norm to measure distance is defined as $N_2(\theta) = \{\text{strictly feasible } (x, y, s) \mid \|XSe - \mu e\| \leq \theta\mu\}$. Note that $\|XSe - \mu e\| \leq \theta\mu$ if and only if $\sum_{i=1}^{n}(x_i s_i - \mu)^2 \leq \theta^2\mu^2$. Figure 16.3(a) shows an example of a neighborhood $N_2(\theta)$. We can also define a neighborhood based on the one-sided infinity norm as $N_{-\infty}(\gamma) = \{\text{strictly feasible } (x, y, s) \mid x_i, s_i \geq \gamma\mu \quad \forall i = 1, \ldots, n\}$.

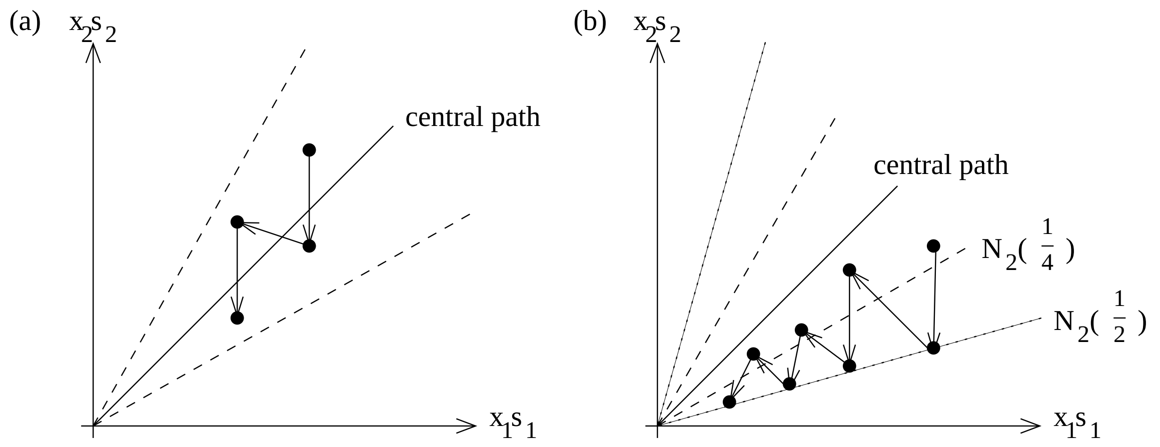


Figure 16.3: (a) A neighborhood of the central path in the case of $n = 2$ variables. (b) The Predictor-Corrector algorithm alternates between predictor steps, in which it moves as far as possible while remaining in $N_2(\frac{1}{2})$, and corrector steps, in which it takes a full step ($\alpha = 1$), returning to $N_2(\frac{1}{4})$.

16.1.3 Types of interior-point algorithms

There are several major types of interior-point algorithms for linear programming.

- **Path-Following**: Path-following algorithms use update steps that follow the central path. The extent to which a path-following algorithm follows the central path is determined by the centering parameter σ. The method of choosing σ distinguishes different path-following algorithms.
  - **Short-Step**: In short-step algorithms, σ is set close to 1 so that the solutions stay near the central path. At most $O(\sqrt{n} \log \frac{1}{\epsilon})$ iterations are needed to achieve $\mu^k \leq \epsilon$. This is the best complexity bound known for an interior-point algorithm.
  - **Long-Step**: In contrast to short-step algorithms, long-step algorithms pick the centering parameter σ to be farther from 1, and as a result the solutions are farther from the central path. The number of iterations required to reduce μ^k so that it is below the threshold ϵ is $O(n \log \frac{1}{\epsilon})$, but long-step algorithms perform better than short-step algorithms in practice.
  - **Predictor-Corrector**: Predictor-corrector algorithms strike a balance between following the central path and moving toward optimal solutions by alternating between steps with $\sigma = 1$ and steps with $\sigma = 0$. These algorithms execute $O(\sqrt{n} \log \frac{1}{\epsilon})$ update iterations, and thus are as fast

in theory as the best known interior-point algorithms. A variant on the predictor-corrector approach is the standard code used in practice.

- **Potential-Reduction**: Potential-reduction algorithms make use of a potential function. The potential function approaches $+\infty$ when $x_i s_i \to 0$ for some $i = 1, \ldots, n$, but $\mu \not\to 0$. This situation occurs when the solutions go near a boundary of the feasible region, but do not approach optimal solutions. The potential function approaches $-\infty$ if and only if the solutions (x, y, s) approach optimal solutions to the linear programs. Potential-reduction algorithms perform update steps to reduce the potential function, eventually moving toward optimal solutions. Karmarkar's seminal interior-point algorithm was a potential-reduction algorithm.

16.1.4 A predictor-corrector algorithm

We now consider a predictor-corrector algorithm that was developed by Mizuno, Todd, and Ye in 1993. In this algorithm, we use two neighborhoods of the central path, $N_2(\frac{1}{2})$ and $N_2(\frac{1}{4})$. An update step with $\sigma = 0$ is referred to as a predictor step. In predictor steps, we choose the next set of solutions to go as far in the Newton direction as possible, subject to the constraint that the new solutions are in $N_2(\frac{1}{2})$. The other type of update step is the corrector step, in which $\sigma = 1$. In a corrector step, we always take a full update step in the Newton direction by setting $\alpha = 1$. This algorithm has the property that these corrector steps always return the current solutions to $N_2(\frac{1}{4})$. Figure 16.2(b) illustrates the update steps taken by the algorithm.

Predictor-Corrector (Mizuno, Todd, Ye 1993)

```
$(x^0, y^0, s^0) \in N_2(\frac{1}{4})$
$k \leftarrow 0$
While $\mu^k > \epsilon$
    If $k$ is even
        Find $(\Delta x^k, \Delta y^k, \Delta s^k)$ for $\sigma^k = 0$
        $\alpha^k \leftarrow \max_{\alpha \in [0,1]}\{\alpha \mid (x^k, y^k, s^k) + \alpha(\Delta x^k, \Delta y^k, \Delta s^k) \in N_2(\frac{1}{2})\}$
        $(x^{k+1}, y^{k+1}, s^{k+1}) \leftarrow (x^k, y^k, s^k) + \alpha^k(\Delta x^k, \Delta y^k, \Delta s^k)$
    Else
        Find $(\Delta x^k, \Delta y^k, \Delta s^k)$ for $\sigma^k = 1$
        $(x^{k+1}, y^{k+1}, s^{k+1}) \leftarrow (x^k, y^k, s^k) + (\Delta x^k, \Delta y^k, \Delta s^k)$
    $k \leftarrow k + 1$
```

In the next lecture, we will prove the following lemmas.

Lemma 16.1 For k even and $(x^k, y^k, s^k) \in N_2(\frac{1}{4})$, $\mu^{k+1} \leq (1 - \frac{0.4}{\sqrt{n}})\mu^k$.

Lemma 16.2 For k odd, $\mu^{k+1} = \mu^k$ and $(x^{k+1}, y^{k+1}, s^{k+1}) \in N_2(\frac{1}{4})$.

The analysis of the running time of this algorithm is based on the following theorems.

Theorem 16.3 Given $(x^0, y^0, s^0) \in N_2(\frac{1}{4})$ with $\mu^0 = C$, after at most $O(\sqrt{n}\log\frac{C}{\epsilon})$ iterations, $\mu^k \le \epsilon$.

Proof: For $k \ge 5\sqrt{n}\ln\frac{C}{\epsilon}$,

$$\mu^k \le C\Big(1 - \frac{0.4}{\sqrt{n}}\Big)^{\frac{5}{2}\sqrt{n}\ln\frac{C}{\epsilon}} \le Ce^{-0.4(\frac{5}{2}\ln\frac{C}{\epsilon})} = C\big(\frac{\epsilon}{C}\big) = \epsilon$$

The first inequality follows from Lemmas 16.1 and 16.2, and to obtain the second inequality we use the fact that $(1 - \frac{x}{k})^k \le e^{-x}$. □

Let L denote the number of bits in (A, b, c), the inputs to the algorithm.

Theorem 16.4 A starting point $(x^0, y^0, s^0) \in N_2(\frac{1}{4})$ can be found such that $\mu^0 \le \frac{1}{n}2^{O(L)}$.

Theorem 16.5 If (x, y, s) is a set of solutions such that $\mu \le \frac{1}{n}2^{-2L}$, then any vertex x^* of the primal feasible region such that $c^Tx^* \le c^Tx$ is an optimal solution to the primal linear program. Moreover, such a vertex x^* can be found in $O(n^3)$ time.

These theorems imply the following upper bound on the running time of the Predictor-Corrector algorithm.

Corollary 16.6 An optimal solution to the linear program can be found in $O(\sqrt{n}L)$ iterations. The overall running time of the Predictor-Corrector algorithm is $O(n^{3.5}L)$.

CS 361B Advanced Algorithms June 9, 2003

Lecture 17

Lecturer: David P. Williamson *Scribe: Vivek Farias*

17.1 Interior-point methods (cont.)

Let us first review the main results from the previous lecture. Recall that our objective was to solve the LP:

$$\begin{aligned} \text{Min} \quad & c^T x \\ \text{sub. to:} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

with dual

$$\begin{aligned} \text{Max} \quad & b^T y \\ \text{sub. to:} \quad & A^T y + s = c \\ & s \geq 0. \end{aligned}$$

The main idea was to find a set of primal feasible and dual feasible points, and continually update these variables using a Newton method until complementary slackness was achieved. Recall that the Newton step is the solution to the system:

$$\begin{pmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S & 0 & X \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta s \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -XSe + \sigma\mu e \end{pmatrix} \tag{17.1}$$

where $\mu = \frac{x^T s}{n}$ and $\sigma \in [0,1]$ is a centering parameter. $\sigma = 1$ implies that we are trying to make all the $x_i s_i$ values the same, while $\sigma = 0$ means that we are trying to move to an optimal solution. We defined notion of a central path, and a neighborhood of the central path, $N_2(\theta) = \{(x,y,s) : ||XSe - \mu e|| \leq \theta\mu, \text{ strictly feasible}\}$.

We also outlined the following interior-point algorithm for solving linear programs.

Predictor-Corrector (Mizuno, Todd, Ye 1993)

```
$(x_0, y_0, s_0) \in N_2(\frac{1}{4})$
In even iterations find $(\Delta x, \Delta y, \Delta s)$ with $\sigma = 0$
Update: $(x^{k+1}, y^{k+1}, s^{k+1}) \leftarrow (x^k, y^k, s^k) + \alpha(\Delta x, \Delta y, \Delta s)$
        for $\alpha \in [0,1]$ as large as possible such that $(x^{k+1}, y^{k+1}, s^{k+1}) \in N_2(\frac{1}{2})$
In odd iterations find $(\Delta x, \Delta y, \Delta s)$ with $\sigma = 1$
Update: $(x^{k+1}, y^{k+1}, s^{k+1}) \leftarrow (x^k, y^k, s^k) + (\Delta x, \Delta y, \Delta s)$
Repeat until $\mu \le \epsilon$.
```

17.1.1 The complexity of the predictor-corrector algorithm

In the sequel we will prove the following lemmas, in order to establish a complexity bound for Predictor-Corrector.

Lemma 17.1 For k even and $(x^k, y^k, s^k) \in N_2(\frac{1}{4})$ then $\mu_{k+1} \le (1 - \frac{.4}{\sqrt{n}})\mu_k$.

Lemma 17.2 For k odd and $(x^k, y^k, s^k) \in N_2(\frac{1}{2})$, we have $\mu_{k+1} = \mu_k$ and $(x^{k+1}, y^{k+1}, s^{k+1}) \in N_2(\frac{1}{4})$.

These lemmas enabled us to prove the following theorem in Lecture 16.

Theorem 17.3 We need at most $O(\sqrt{n}\ln(\frac{C}{\epsilon}))$ iterations before $\mu_k \le \epsilon$ if $\mu_0 \le C$.

We first introduce some notation and state a result that will help with the proof of the above lemmas. Define $x(\alpha) = x + \alpha\Delta x$. Similarly define $y(\alpha) = y + \alpha\Delta y$, $s(\alpha) = s + \alpha\Delta s$, and $\mu(\alpha) = \frac{1}{\alpha}x(\alpha)s(\alpha)$.

We state the following lemma without proof.

Lemma 17.4 For $(x, y, s) \in N_2(\theta)$, $\|\Delta X \Delta S e\| \le \frac{\theta^2 + n(1-\sigma^2)}{2^{3/2}(1-\theta)}\mu$.

The following lemma will be useful in what follows.

Lemma 17.5 $\Delta x^T \Delta s = 0$ and $\mu(\alpha) = (1 - \alpha(1-\sigma))\mu$.

Proof: We know from (17.1) that

$$A^T\Delta y + \Delta s = 0$$

and

$$A\Delta x = 0.$$

Hence the first conclusion is immediate; Δx is in the kernel of A, while Δs is in the range of A', so they must be orthogonal. Again from (17.1),

$$S\Delta x + X\Delta s = -XSe + \sigma\mu e$$

$$\implies s_i\Delta x_i + x_i\Delta s_i = -x_is_i + \sigma\mu \;\forall i \tag{17.2}$$

$$\implies s^T\Delta x + x^T\Delta s = -x^Ts + n\sigma\mu \tag{17.3}$$

Hence,

$$\begin{aligned}
\mu(\alpha) &= \frac{1}{n}(x+\alpha\Delta x)^T(s+\alpha\Delta s) \\
&= \frac{1}{n}(x^Ts + \alpha(\Delta x^Ts + s^T\Delta x) + \alpha^2\Delta x^T\Delta s) \\
&= \mu + \frac{\alpha}{n}(-x^Ts + n\sigma\mu) \qquad (17.4) \\
&= (1-\alpha(1-\sigma))\mu,
\end{aligned}$$

where (17.4) follows from (17.3). □

Let us restate and then prove Lemma 17.2 in light of our new notation.

Lemma 17.6 For $\alpha = 1$, $\sigma = 1$, $(x, y, s) \in N_2(\frac{1}{2})$, we have $(x(1), y(1), s(1)) \in N_2(\frac{1}{4})$ and $\mu(1) = \mu$.

Proof: That $\mu(1) = \mu$ follows from Lemma 17.5 with $\alpha = 1$. Now we need to show $\|X(1)S(1)e - \mu(1)e\| \leq \mu(1)/4$. By using (17.2) from Lemma 17.5 we have

$$\begin{aligned}
x_i(\alpha)s_i(\alpha) - \mu(\alpha) &= (x_i + \alpha\Delta x_i)(s + \alpha\Delta s_i) - \mu \\
&= x_is_i + \alpha(\Delta x_is_i + x_i\Delta s_i) + \alpha^2\Delta x_i\Delta s_i - \mu \\
&= (1-\alpha)x_is_i + \alpha^2\Delta x_i\Delta s_i + (\alpha-1)\mu.
\end{aligned}$$

Thus for $\alpha = 1$ we have $\|X(1)S(1)e - \mu(1)e\| = \|\Delta X\Delta Se\|$. By Lemma 17.4, we know that $\|\Delta X\Delta Se\| \leq \frac{\mu}{4} = \frac{\mu(1)}{4}$ from our first conclusion in this proof.

Hence all that remains is to ensure that strict feasibility is maintained. By the above, we have that

$$\begin{aligned}
\|X(\alpha)S(\alpha)e - \mu(\alpha)e\| &\leq (1-\alpha)\|XS - \mu e\| + \alpha^2\|\Delta X\Delta Se\| \\
&\leq |1-\alpha|\mu/2 + \alpha^2\mu/4,
\end{aligned}$$

where the inequality on the first term follows since $(x, y, s) \in N_2(\frac{1}{2})$ and the inequality on the second term follows as above. Thus note that for any $\alpha \in [0, 1]$ and for all i, $x_i(\alpha)s_i(\alpha) \geq -(1-\alpha)\mu/2 - \alpha^2\mu/4 + \mu > 0$ by simple calculus. Hence $x(1), s(1)$ are strictly feasible, since $x(0), s(0)$ are strictly feasible and for no $\alpha \in [0, 1]$ can $x(\alpha)$ or $s(\alpha)$ be zero. □

We now restate and prove Lemma 17.1.

Lemma 17.7 For $(x,y,s) \in N_2(\frac{1}{4})$, $\sigma = 0$, we have $(x(\alpha), y(\alpha), s(\alpha)) \in N_2(\frac{1}{2})$ for all $\alpha \in [0, \bar{\alpha}]$, where

$$\bar{\alpha} = \min\left(\frac{1}{2}, \left(\frac{\mu}{\delta\|\Delta X \Delta S e\|}\right)^{1/2}\right) \geq \frac{0.4}{\sqrt{n}}.$$

Proof: Since $\sigma = 0$, we have that $\mu(\alpha) = (1-\alpha)\mu$ from Lemma 17.5, so that $\mu(\bar{\alpha}) \leq (1 - \frac{0.4}{\sqrt{n}})\mu$. We would like to quantify $\|\Delta X \Delta S e\|$. As above, we have that

$$\begin{aligned}
\|X(\alpha)S(\alpha) - \mu(\alpha)e\| &\leq (1-\alpha)\|XS - \mu e\| + \alpha^2\|\Delta X \Delta S e\| \\
&\leq \frac{1}{4}(1-\alpha)\mu + \frac{\mu}{8\|\Delta X \Delta S e\|}\|\Delta X \Delta S e\| \\
&= \frac{1}{4}(1-\alpha)\mu + \frac{\mu}{8(1-\alpha)}(1-\alpha) \\
&\leq \frac{1}{4}(1-\alpha)\mu + \frac{\mu}{4}(1-\alpha) \\
&= \frac{1}{2}(1-\alpha)\mu \\
&= \frac{1}{2}\mu(\alpha),
\end{aligned}$$

where the second inequality holds since $(x,y,s) \in N_2(1/4)$ and $\alpha \leq \left(\frac{\mu}{\delta\|\Delta X \Delta S e\|}\right)^{1/2}$ by hypothesis, and the third inequality holds since $1 - \alpha \geq 1/2$. by hypothesis. Thus $(x(\alpha), y(\alpha), s(\alpha)) \in N_2(1/2)$. The proof that $x(\alpha), s(\alpha)$ is strictly feasible follows as in the proof of Lemma 17.2.

By Lemma 17.4,

$$\frac{\mu}{8\|\Delta X \Delta S e\|} > \frac{0.16}{n}.$$

So $\bar{\alpha} \geq \frac{0.4}{\sqrt{n}}$, and thus $\mu(\alpha) \leq (1 - \frac{0.4}{\sqrt{n}})\mu$ as desired. □

17.2 A flavor of the ellipsoid method

We consider LPs of the form:

$$\begin{aligned}
&\text{Min} \quad c^T x \\
&\text{subject to:} Ax \leq b
\end{aligned}$$

What follows is a brief outline of the algorithm:

Ellipsoid (Khachian 1979)

```
Determine an ellipsoid $\epsilon_0$ containing the feasible region
$i \leftarrow 0$
While Volume($\epsilon_i$) $> \epsilon$
    Check if the center of $\epsilon_i$, $x_i$ is feasible
    If not feasible
        Get violated constraint $a_i' x_i > b_i$
    Else            $a_i \leftarrow c$
    Find ellipsoid $\epsilon_{i+1}$ containing $\epsilon_i \cap \{a_i^T x \leq a_i^T x_i\}$
```

There are three things that we essentially need to show to establish that the Ellipsoid Method is a polynomial-time algorithm for LP, namely:

- Bound on the size of the initial ellipsoid.
- Bound on the final ellipsoid.
- That every iteration makes progress.

In fact one can show that the initial ellipsoid is not too large, and a sufficiently small ellipsoid contains exactly one optimal vertex of the LP. One can then round the center x_i of this ellipsoid to the vertex in polynomial time. Furthermore, one can show that after n iterations of the main loop above, the volume of the ellipsoid has dropped by a constant factor, so given reasonable starting and ending volumes, the algorithm runs in polynomial time.

A useful property of the ellipsoid method is that the LP does not need to be specified in advance. Note that it is sufficient if one can produce a violated constraint of the LP in polynomial time. Hence it is possible to solve LPs with an exponential number of constraints in polynomial time if we have a polynomial-time *separation oracle*, which, given an x either declares x feasible for the LP or produces a constraint of the LP which is violated by x.