CSC2411 - Linear Programming and Combinatorial Optimization[*] Lecture 5: Smoothed Analysis, Randomized Combinatorial Algorithms, and Linear Programming Duality

Notes taken by Mea Wang

February 11, 2005

Summary: In this class, we discuss a few "post-simplex-algorithm" issues. We will first study the smoothed case analysis of Linear Programming problems. We then learn the Seidel's algorithm, a randomized combinatorial algorithm that run in subexponential time, and its extensions. Last, we will be introduced to the duality theorem of Linear Programs.

1 Overview

In the previous lecture, we learned the procedure of the simplex algorithm. The algorithm leads us to an optimum solution. The question now is how long it will take the algorithm to reach this final solution. Moreover, we may wonder how well the algorithm perform under different settings. In this lecture, we will first examine the complexity of the simplex algorithm. Following this analysis, we will learn three randomized combinatorial algorithms which improve the running time, at least for some sets of parameters. Last, we will start our discussion on Linear Program duality.

2 Smoothed Analysis

The worst case analysis estimates the maximum running time T of an algorithm $\mathcal{A}$ on all possible inputs I_n of a given length n, whereas the average case analysis provides the average running time of a program on a distribution of inputs of a given length. The smoothed analysis was invented a few years ago motivated by the strictness of the worst-case and average-case analyses. The smoothed analysis presents the maximum running time of cases within certain range of the average cases. The range is defined

[*] Lecture Notes for a course given by Avner Magen, Dept. of Computer Sciecne, University of Toronto.

by a scalar variable σ. We can intuitively think of σ as the distance away from the average case we are interested in. It enabled us to study the behavior of an algorithm under different settings. In mathematical symbols, these analyses are

$$\begin{aligned}
\text{worst case}(\mathcal{A}; n) &= \max_{w \in I_n} T(x) && (1)\\
\text{average case}(\mathcal{A}; n) &= \text{avg}_{w \in I_n} T(x) && (2)\\
\text{smoothed case}(\mathcal{A}; n, \sigma) &= \max_{w \in I_n} \text{avg } T(\text{w in } \sigma \text{ neighborhood of x}) && (3)
\end{aligned}$$

It is easy to see that Eqn. 1 $\geq$ Eqn. 3 $\geq$ Eqn. 2. For Linear Programming problems, the algorithm is actually $\mathcal{A}(Ax \leq b)$, and all possible inputs I_n are the all possible combinations of A, b, and c. In general, the parameter σ varies depending on the problem being solved. In the smoothed case analysis, we let G be a matrix of independently chosen Gaussian random variables of mean 0 and variance 1. Since we seek to analyze the behavior of the algorithm within the σ neighborhood, we vary the constrain matrix A by a σG. Hence, the algorithm is $\mathcal{A}((A + \sigma G)x \leq b)$.

Theorem 2.1. *[ST01] Smoothed case* $($*Simplex*$; n, \sigma) = \text{poly}(m, n, 1/\sigma)$.

Theorem 2.1 essentially says that the simplex algorithm solves every Linear Program in time polynomial in m, n, and $1/\sigma$ as long as we perturb the program in some appropriate way scaled by σ. In fact, the worst case is a special case of the smoothed case in which $\sigma = 0$. If σ is so large so that A becomes insignificant, we obtain the average-case analysis. When σ is polynomially small, the analysis is a combination of the worst-case and average-case analyses. In Linear Programming problems, the worst case is exponential and the average case is polynomial. Theorem 2.1 tells us the simplex is better than the “just in average”.

3 Randomized Combinatorial Algorithms

Let’s consider a Linear Program $Ax \leq b$ with small number n of variables and m of constrains. Assume H is the set of corresponding linear constraints. Fig. 1 shows a polytope formed by a set of linear constraints for an instance of this problem. Recall that if x is a vertex, then n of the linearly independent inequalities are satisfied as equalities. In fact, the solution to the problem with only these n constraints is the same as the original problem. We refer to these constraints as the *critical constraints*. In Fig. 1, the critical constraints are represented by h_1 and h_2. The cone bounded by the hyperplanes h_1 and h_2 has the same solution as the original problem.

Claim 3.1. *The cone formed by the critical constraints has the same optimum solution as the original polytope.*

3.1 Seidel’s Algorithm

Seidel’s algorithm [Sei91] is a simple randomized, incremental algorithm for Linear Programs, that works well for small values of n.

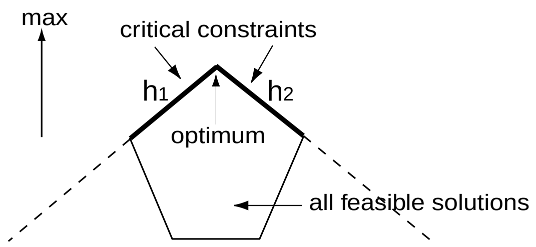

Figure 1: Critical constrains and optimum solution.

Algorithm 3.2 (Seidel's Algorithm (LP(H)).

```
Randomly pick a constraint $h \in H$
$x$ := Seidel's Algorithm (LP($H \setminus \{h\}$))
**if** ($x$ satisfies $h$)
        **return** $x$
**else**
        Project all constraints in $H$ (input of current iteration)
        onto $h$ to form $H'$
        **return** Seidel's Algorithm(new LP($H'$))
```

In Algorithm 3.2, if x satisfies h, it is the optimal solution of the original problem. If x does not satisfy h, the solution must lie on the hyperplane defined by h. That is h is satisfied as an equality in the optimum solution. In this case, we project all other constraints onto h by casting these constraints onto their intersections with h. Last, we solve the new Linear Program recursively with (n-1) variables on (m-1) constraints.

Let's go through the algorithm with a simple example shown in Fig. 2, in which the objective function is directed upward and the polytope is defined by 6 hyperplanes. Obviously, the optimal solution is at the intersection of hyperplanes 3 and 4. In fact, these two hyperplanes correspond to the two critical constraints of this problem.

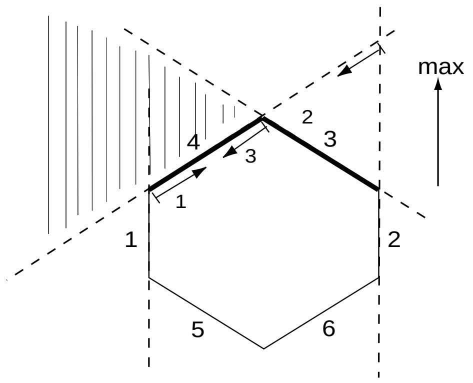

Figure 2: An example for Seidel's algorithm.

Any one of the hyperplanes 1, 2, 5, or 6 can be removed without rendering x infeasible. If $h = 4$ in a particular iteration, the solution x should be in the shaded region and would not satisfy h. We then know that hyperplane 4 is one of the critical constraints, and project all remaining constraints onto it. The polytope is now reduced to a new Linear Program with one less dimension. The optimum solution remains the same in this new Linear Program.

Assume there are k critical constraints out of m constraints. In the best case, we want the algorithm to throw away the $m - k$ non-critical constraints before hitting a critical constraint h. In this case, we have less number of constraints to project onto h. The algorithm terminates faster. In the worse case, the algorithm consider all k critical ones before the non-critical ones. For each critical constraints h, We need to project all non-critical and subset of the critical constraints to it. It takes longer for the algotirhm to terminate. Next, we will examine the complexity of this algorithm.

3.2 Analysis of Seidel's Algorithm

Let A be an $m \times n$ matrix, and $m \geq n$. In every recursion, the number of constraints is reduced by one. It takes the algoritm $O(n)$ time to check satisfiability of x in each recursive call. When x does not satisfy h, it takes the algorithm $O(mn)$ to project remaining constraints onto h. The complexity of this recursive algorithm is as follows:

$$T(n,m) = T(n,m-1) + O(n) + \begin{cases} 0 & \text{if } x \text{ satisfies } h, \\ O(mn) + T(n-1,m-1) & \text{otherwise.} \end{cases} \tag{4}$$

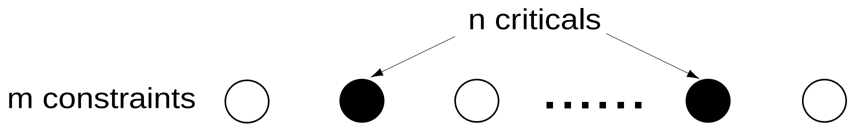

Figure 3: m constraints on n variables

Fig. 3 presents an intuitive way of this analysis. The rank of A can never exceed n, that is at most n out of the m constraints are critical. In other words, every vertex of the polytope is defined by at most n constraints. Hence, the case where x does not satisfy h happens with probability $\frac{n}{m}$. Eqn. 4 is then reduced to:

$$T(n,m) = T(n,m-1) + O(n) + \tfrac{n}{m}(O(mn) + T(n-1,m-1))$$

Claim 3.3. *The Seidel's algorithm runs in exponential time $O(n!m)$.*

Proof. We prove this by double induction on n and m. We do induction on m within the induction step for n.

Basis: Let $n = 1$. It takes the algorithm need to verify the only one constraint with all m variables. Thus, the execution time is $O(m)$ and $O(n!m) = O(m)$.

Inductive step: Assume that for $n \geq 2, \forall 1 \leq i < n$, $T(i,m) = O(i!m)$. We want to show that $T(n,m) = O(n!m)$.

$$T(n,m) = T(n,m-1) + O(n) + \tfrac{n}{m}(O(mn) + T(n-1,m-1))$$

We need to show that $T(n,m-1) = O(n!(m-1))$. We prove this by induction on m.

Basis: Let $m = 1$. $T(n,0) = O(0)$ since the there are no variables left in this case.

Inductive step: Assume that for $m \geq 2, \forall 1 \leq j < m$, $T(n,j-1) = O(n!(j-1))$. We want to show that $T(n,m-1) = O(n!(m-1))$.

$$\begin{aligned} T(n,m-1) &= T(n,m-2) + O(n) + O(n^2) + \tfrac{n}{m-2}T(n-1,m-2) \\ &= O(n!(m-2)) + O(n^2) + \tfrac{n}{m-2}O((n-1)!(m-2)) \\ &= O(n!(m-2)) + O(n^2) + O(n!) \\ &= O(n!(m-1)) \end{aligned}$$

Hence, $T(n,m-1) = O(n!(m-1)$ for all $m \geq 1$.

Now, we continue with the induction on n.

$$\begin{aligned} T(n,m) &= O(n!(m-1)) + O(n) + O(n^2) + \tfrac{n}{m}T((n-1)!(m-1)) \\ &= O(n!(m-1)) + O(n!\tfrac{m-1}{m}) \\ &= O(n!(m-1)) + O(n!) \\ &= O(n!m) \end{aligned}$$

Therefore, $T(n,m) = O(n!m)$ for all $n, m \geq 1$. $\square$

To summarize, the Seidel's algorithm runs in exponential time $O(n!m)$. This upper bound is linear when the dimension n is constant.

Note that the set of critical constraints may or may not be unique. As shown in Fig. 4, the point at the tip of the pyramid is the optimum solution. Let H' be the set of constraints corresponding to the four hyperplanes intersecting at the optimum point. Any three of H' are sufficient to define this optimum point. As soon as the Seidel's algorithm removes the first constraint in H', the remaining three become the critical constraints.

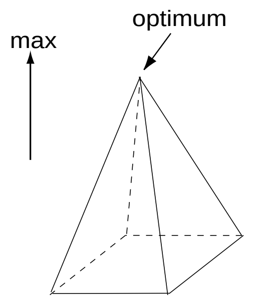

Figure 4: The set of critical constraints may not be unique.

3.3 Matousek Sharir Welzl's Improvement

The Seidel's algorithm throws away the solution x if it does not satisfy h and solves the subproblem with all constraints being projected onto h. In 1992, Matousek, Sharir, and Welzl [MSW92, SW92] identified this flaw and improved the algorithm using a more sophisticated structure. They used information in the recursion regardless the satisfaction of h. This improvement leads to a smaller complexity, $\exp(2\sqrt{n \ln \frac{m}{\sqrt{n}}} + O(\sqrt{n} + \ln m))$.

3.4 Kalai's results

Let $\Delta(n, m)$ be the maximal possible diameter of a polyhedron defined by m constraints on n variables. Then $\Delta(n, 2n) \geq n$ follows. We illustrate this with an example.

Example 3.4. The following constraints,

$$\begin{aligned} 0 \leq x_1 &\leq 1 \\ &\vdots \\ 0 \leq x_n &\leq 1 \end{aligned}$$

define an n-dimensional cube as shown in Fig. 5. The path between any two points in this polyhedron involves at most n steps. Since the diameter of a polyhedron is the shortest path between the two points furthest apart, the diameter of this cube is n. Hence, the maximal possible diameter is at least n.

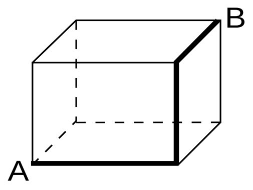

Figure 5: An n-dimensional cube.

Conjecture 3.5 (Hirsch [Dan63]). *In general* $\Delta(n, m) = \Theta(n + m)$

Obviously, the path defined by $\Delta(n, m)$ might not be always moving in the direction of the objective function. In fact, it is the shortest path between two points. The *maximal directed diameter* $\Delta'(n, m)$ is the longest path along the direction of the objective function $\langle c, x \rangle$ between two points in a polyhedron defined by m constraints on n variables. Thus, $\Delta'(n, m) \geq \Delta(n, m)$.

In the simplex algorithm, we start at one vertex of the polyhedron and move to another vertex that has greater value of the objective function until the optimum is found. If a simplex algorithm terminates in $\mathcal{T}$ iterations for every Linear Program with m constraints on n variables, then $\mathcal{T} \geq \Delta'(n, m) \geq \Delta(n, m)$.

In 1992, Kalai's algorithm [Kal92] emerged from the study of the diameter problem for graphs of polyhedra. The algorithm is a randomized simplex algorithm that gives an upper bound on the number of iterations which is $\exp(O\sqrt{n \log n})$. This shows, for the first time, a similar bound on $\Delta'(n, m)$.

4 Duality Theorem

Let's first consider two examples.

Example 4.1. Given the following Linear Program

$$\min -x_2$$

s.t.

$$x_1 + x_2 \leq 8 \tag{5}$$

$$-3x_1 + 2x_2 \leq 6 \tag{6}$$

how do we lower bound the value of the optimum solution?
$3\times$Eqn. 5:

$$3(x_1 + x_2) \leq 24 \tag{7}$$

Eqn. 6+Eqn. 7:

$$3(x_1 + x_2) + (-3x_1 + 2x_2) \leq 30 \tag{8}$$

Solve Eqn. 8:

$$-x_2 \geq -6$$

For any feasible solution $-x_2$ is at least -6.

Example 4.2. Given the following Linear Program

$$\max 5x_1 + 6x_2 + 9x_3 + 8x_4$$

s.t.

$$x_1 + 2x_2 + 3x_3 + x_4 \leq 5 \tag{9}$$

$$x_1 + x_2 + 2x_3 + 3x_4 \leq 3 \tag{10}$$

$$x_i \geq 0, i = 1, 2, 3, 4$$

how do we upper bound the value of the optimum solution?

To make each term in Eqn. 9 and Eqn. 10 bigger than the terms in the objective function, we apply the following operations.
$8\times$Eqn. 9:

$$5x_1 + 6x_2 + 9x_3 + 8x_4 \leq 8x_1 + 16x_2 + 24x_3 + 8x_4 \leq 40 \tag{11}$$

$6\times$Eqn. 10:

$$5x_1 + 6x_2 + 9x_3 + 8x_4 \leq 6x_1 + 6x_2 + 12x_3 + 18x_4 \leq 18 \tag{12}$$

Eqn. 12 provides a tighter upper bound than Eqn. 11. Consider a feasible solution $x_1 = 1$, $x_2 = 2$, $x_3 = x_4 = 0$,

$$5x_1 + 6x_2 + 9x_3 + 8x_4 = 5 + 12 + 0 + 0 = 17$$

Another possible upper bound is 1·Eqn. 9+4·Eqn. 10:

$$5x_1 + 6x_2 + 9x_3 + 8x_4 \leq 5x_1 + 6x_2 + 11x_3 + 13x_4 \leq 17$$

Therefore we have an upper bound of 17.

This can be thought of as a search for the best upper bound for a maximization problem or the best lower bound for a minimization problem. We had to choose $y_1, y_2 \geq 0$ as the coefficient in our proof. It was necessary to get a combination that dominated the objective function. Consider the constraints in Example 4.2,

$$\begin{aligned} & y_1(x_1 + 2x_2 + 3x_3 + x_4) + y_2(x_1 + x_2 + 2x_3 + 3x_4) \qquad (13) \\ = \quad & (y_1 + y_2)x_1 + (2y_1 + y_2)x_2 + (3y_1 + 2y_2)x_3 + (y_1 + 3y_2)x_4 \end{aligned}$$

In order to have each term in Eqn. 13 bigger than the terms in the objective function, we need to have

$$\begin{aligned} y_1 + y_2 &\geq 5 \\ 2y_1 + y_2 &\geq 6 \\ 3y_1 + 2y_2 &\geq 9 \\ y_1 + 3y_2 &\geq 8 \end{aligned}$$

Note that the objective function is less than Eqn. 13, which is less than $5y_1 + 3y_2$. Hence, seeking for the tightest upper bound for Example 4.2 is the same as minimizing value of $5y_1 + 3y_2$. This gives rise to a new Linear Programming problem as follows:

$$\min 5y_1 + 3y_2$$

s.t.

$$\begin{aligned} & y_1 + y_2 \geq 5 \\ & 2y_1 + y_2 \geq 6 \\ & 3y_1 + y_2 \geq 9 \\ & y_1 + 3y_2 \geq 8 \\ & y_1, y_2 \geq 0 \end{aligned}$$

Note that the coefficients in the objective function are exactly the numbers on the right side of the inequalities in the original Linear Program. The coefficients on the left side of the inequalities are actually the transpose of matrix A, and the numbers on the right side are the vector c in the original problem. Everything here satisfy the constraints is an upper bound of the original problem. This is called the *dual*, and the original problem is referred to as the *primal*.

Let's return to the diet problem given in the first lecture.

Example 4.3 (the diet problem). A farmer wants to spend as little as possible while keeping the cows healthy. There are n different food types available, the j^{th} food containing $c_j \in \mathcal{R}$ calories per kilogram, $1 \leq j \leq n$, and $a_{ij} \in \mathcal{R}$ milligrams of vitamin i per kilogram, $1 \leq i \leq m$. The cow requires at least $b_i \in \mathcal{R}$ milligrams of vitamin i to stay healthy. Given that the goal is to minimize the money spent while having enough of each vitamin, how should the cow be fed?

The problem in Linear Program form is as follow and is presented in a graphically in Fig. 6.

$$\begin{aligned} & \min\langle c, x\rangle \\ \text{s.t.} \quad & \\ & Ax \geq b \\ & x_i \geq 0, 1 \leq i \leq n \end{aligned}$$

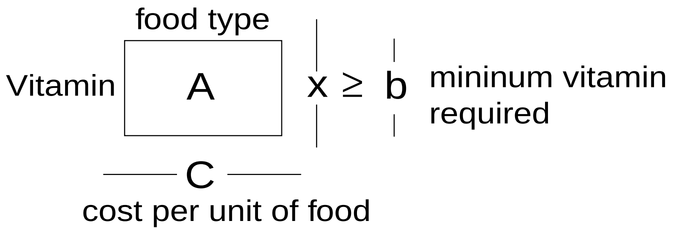

Figure 6: The diet problem.

How do we bound the minimum cost? In other words, how do we convert this to the dual?

Following our observations from Example 4.2, we introduce a vector y and want to minimize $\langle y, b\rangle$. The new constraints are $yA \leq c$. Formally, we have

$$\begin{aligned} & \max\langle y, b\rangle \\ \text{s.t.} \quad & \\ & yA \leq c \\ & y_i \geq 0, 1 \leq i \leq m \end{aligned}$$

To see this results intuitively, we can think of the vector y as the price of vitamin pills in the drug mart and the cost vector c as the price of each unit of food. The pharmacist always wants to maximize his/her profit while being competitive against each food type.

This naturally leads to the duality theorem, which we will be introduced to in the next lecture. For every maximization problem, there is a corresponding minimization problem, and vice versa. They provide upper/lower bound for each other. In fact, as we will see, their optimum solutions are the same.

5 Tutorial: February 2

In this tutorial, we will discuss the dual of different forms of Linear Programs, and learn the technique to transform an Linear Program between its primal and dual. We will wrap this tutorial with an application of duality theorem.

5.1 Duality Theorem

Any Linear Program problem can be expressed as either the primal or the dual. Conventionally, we referred to the original problem as the *primal*, and the other one as the *dual*. Given a Linear Program in general form, the primal and dual are defined as follows:

Primal		Dual
$\min\langle c', x\rangle$		$\max\langle \pi', b\rangle$
$a_i' x = b_i$	$i \in M$	$\pi_i \lesseqgtr 0$
$a_i' x \geq b_i$	$i \in \bar{M}$	$\pi_i \geq 0$
$x_j \geq 0$	$j \in N$	$\pi' A_j \leq c_j$
$x_j \lesseqgtr 0$	$j \in \bar{N}$	$\pi' A_j = c_j$

In general, if the primal is feasible, the dual is feasible as well. It is possible to have both the primal and dual infeasible. If one of them is unbounded, the other one must be infeasible.

Here, we want to focus on how to transform between the primal and dual of a Linear Program. The objective functions of the primal and dual are related by the scalar $\pi' Ax$. More precisely, we have $c'x \geq \pi' Ax \geq \pi' b$. The primal and dual of a Linear Program can be visualized as in Fig. 7 . These two tables provide us a good reference on how to map the problem from one form to another.

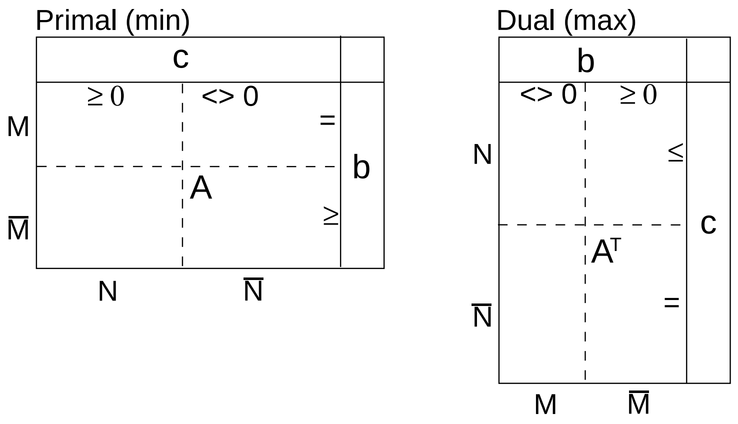

Figure 7: Primal and dual of an LP.

5.2 Application: Shortest Path

Duality theorem has a wide range of applications. We will see its usefulness through an example of the shortest path problem.

Example 5.1. Given a directed graph $G = (V, E)$ (Fig. 8), a cost $c_j \geq 0$ is assigned to each edge $e_j \in E$. We want to find the least-cost path from node s to node t.

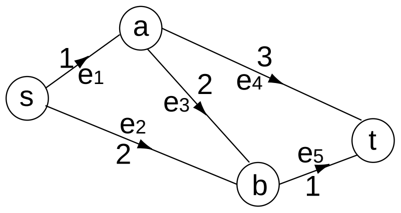

Figure 8: An example of the shortest path problem.

The graph G can be represented by a matrix A, in which each entry is defined as follows,

$$a_{ij} = \begin{cases} +1 & \text{if edge } e_j \text{ leaves node } v_i, \\ -1 & \text{if edge } e_j \text{ arrives node } v_i, \\ 0 & \text{otherwise.} \end{cases}$$

$$A = \begin{pmatrix} +1 & +1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 \\ -1 & 0 & +1 & +1 & 0 \\ 0 & -1 & -1 & 0 & +1 \end{pmatrix}$$

An edge e_j has positive flow $f_j > 0$ if it is selected to construct the path from s to t. The objective is to minimize the total cost of the path. Thus, the Linear Program for the problem is

$$\begin{aligned} & \min\langle c, f\rangle \\ \text{s.t.} & \\ & Af = b \\ & f_i \geq 0, i = 1, 2, 3, 4, 5 \end{aligned}$$

The vector b specifies the flow conservation on each node in the final path. In other words, the entry for s is $+1$ indicating that a unit flow is leaving s, and the entry for t is -1 indicating that a unit flow is arriving at t. The rest of the entries should be 0 since each node should forward all flows it received. A BFS of this problem must have $|E| - |V|$ positive entries.

It is possible to have a non-integral optimum solution s'. In fact, we can derive a integral (a BFS) optimum solution from s'. To do so, we let the set $S = (s_1, \cdots, s_n)$ be all BFSs. Since all feasible solution are in the convex hall of the BFSs, then $s' = \sum_i \lambda_i s_i$ for $0 \leq \lambda_i \leq 1$ and $\sum_i \lambda_i = 1$. Let j be the index corresponding to the BFS with lowest cost. Then

$$c \cdot s' = \sum_i \lambda_i (c \cdot s_i) \geq (c \cdot s_j) \sum_i \lambda_i = c \cdot s_j$$

which shows that s_j is not only a BFS, but also the optimum integral solution.

The dual form of this problem is then

$$\max \langle \pi', b \rangle$$

s.t.

$$\pi' A \leq c$$
$$\pi'_i \lesseqgtr 0, i \in V$$

We can rewrite the Linear Program as

$$\max \pi_s - \pi_t$$

s.t.

$$\pi_i - \pi_j \leq c_{ij}, \forall (i,j) \in E$$
$$\pi_i \lesseqgtr 0, i \in V$$

Now, we examine the feasibility of this dual. Let d_{ij} be the shortest path between any two points in G. For each vertex v in V, let $\pi_v = d_{vt}$. The constraints of the dual become $d_{it} - d_{jt} \leq c_{ij}$, which satisfies the triangle inequality as shown in Fig. 9. In words, if there exists a shortest path from node i to t, its cost must be no more than the cost of the path from i to t via node j. Hence, the constraints are feasible.

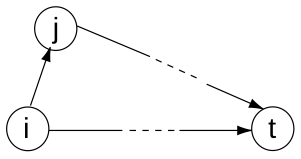

Figure 9: Triangle inequality.

At last, we want to show that the optimum solution of the dual is the shortest path from s to t. Since d_{st} is a feasible solution to the dual, $d_{st} \leq \hat{\pi}' b$, where $\hat{\pi}' b$ is the optimum solution. Obviously, the shortest path from s to t is the path $s \rightarrow b \rightarrow t$. According to the dual, the following two inequality follows,

$$\pi_s - \pi_b \leq c_{sb} \tag{14}$$
$$\pi_b - \pi_t \leq c_{bt} \tag{15}$$

Eqn. 14 + Eqn. 15 gives us $\pi_s - \pi_t \leq c_{sb} + c_{bt} = d_{st}$. Since $\pi' b = \pi_s - \pi_t$, we have $\pi' b \leq d_{st} \leq \hat{\pi}' b$. Hence, d_{st} is the optimum solution. Therefore, the optimum solution of the dual is the shortest path from s to t.

References

[Dan63] G. B. Dantzig. Linear programming and extensions. *Princeton University Press, Princeton, NJ*, 1963.

[Kal92] G. Kalai. A subexponential randomized simplex algorithm. *in Proceedings of 24th Annual ACM Symposium on Theory of Computing*, pages 475–482, 1992.

[MSW92] J. Matoušek, M. Sharir, and E Welzl. A subexponential bound for linear programming. *in Proceedings of 8th Annual ACM Symposium on Computational Geometry*, pages 1–8, 1992.

[Sei91] R. Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Computational Geometry: Theory and Applications*, 1:51–64, 1991.

[ST01] D. A. Spielman and S. Teng. Smoothed analysis: Why the simplex algorithm usually takes polynomial time. *in Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing*, pages 296–305, 2001.

[SW92] M. Sharir and E Welzl. A combinatorial bound for linear programming and related problems. *in Proceedings of 9th Symposium on Theoretical Aspects of Computer Science*, 577 of Lecture Notes in Computer Science:569–579, 1992.