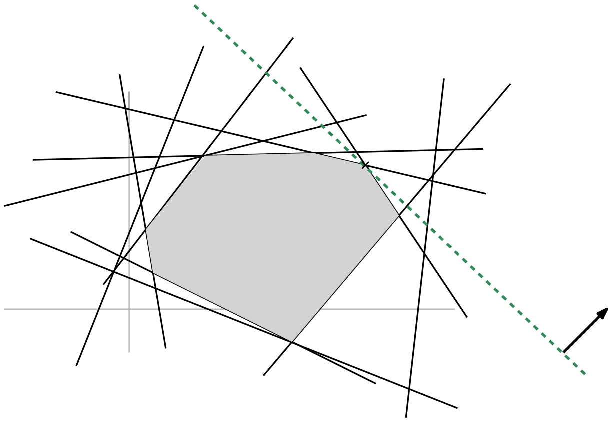
In this lecture we describe a very nice algorithm due to Seidel for Linear Programming in low-dimensional spaces. We then discuss the general notion of Linear Programming Duality, a powerful tool that you should definitely master.

1 Seidel's LP algorithm

We now describe a linear-programming algorithm due to Raimund Seidel that solves the 2-dimensional (i.e., 2-variable) LP problem in $O(m)$ time (recall, m is the number of constraints), and more generally solves the d-dimensional LP problem in time $O(d!m)$.

Setup: We have d variables $x_1, \ldots, x_d$. We are given m linear constraints in these variables $\mathbf{a}_1 \cdot \mathbf{x} \leq b_1, \ldots, \mathbf{a}_m \cdot \mathbf{x} \leq b_m$ along with an objective $\mathbf{c} \cdot \mathbf{x}$ to maximize. (Using boldface to denote vectors.) Our goal is to find a solution $\mathbf{x}$ satisfying the constraints that maximizes the objective. In the example above, the region satisfying all the constraints is given in gray, the arrow indicates the direction in which we want to maximize, and the cross indicates the $\mathbf{x}$ that maximizes the objective.



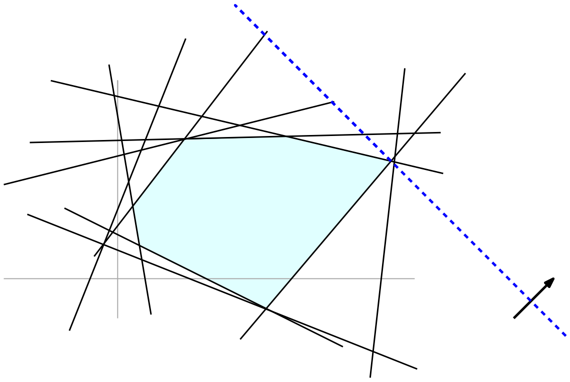
(You should think of sweeping the green dashed line, to which the vector $\mathbf{c}$ is normal, in the direction of $\mathbf{c}$, until you reach the last point that satisfies the constraints.)

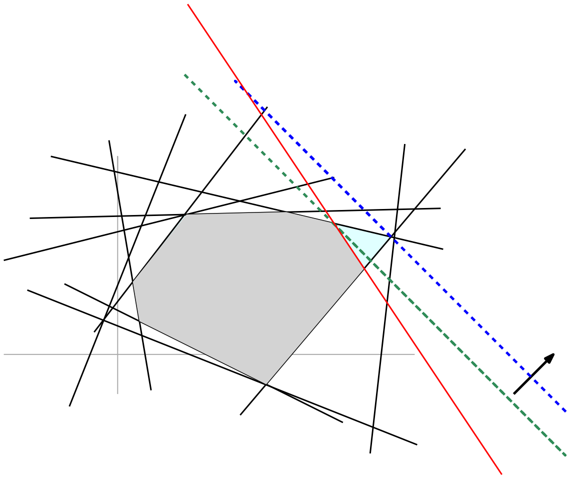
The idea: Here is the idea of Seidel's algorithm. Let's add in the constraints one at a time, and keep track of the optimal solution for the constraints so far. Suppose, for instance, we have found the optimal solution $\mathbf{x}^*$ for the first $m-1$ constraints (let's assume for now that the constraints so far do not allow for infinitely-large values of $\mathbf{c}\cdot\mathbf{x}$) and we now add in the mth constraint $\mathbf{a}_m \cdot \mathbf{x} \leq b_m$. There are two cases to consider:

Case 1: If $\mathbf{x}^*$ satisfies the constraint, then $\mathbf{x}^*$ is still optimal. Time to perform this test: $O(d)$.

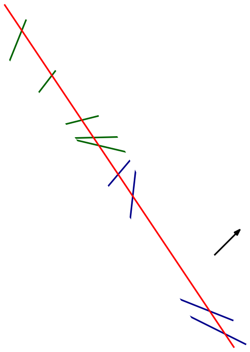
Case 2: If $\mathbf{x}^*$ doesn't satisfy the constraint, then the new optimal point will be on the $(d-1)$-dimensional hyperplane $\mathbf{a}_m \cdot \mathbf{x} = b_m$, or else there is no feasible point.

As an example, consider the situation below, before and after we add in the linear constraint $\mathbf{a}_m \cdot \mathbf{x} \leq b_m$ colored in red. This causes the feasible region to change from the light blue region to the smaller gray region, and the optimal point to move.





Let's now focus on the case $d = 2$ and consider the time it takes to handle Case 2 above. With $d = 2$, the hyperplane $\mathbf{a}_m \cdot \mathbf{x} = b_m$ is just a line, and let's call one direction "right" and the other "left". We can now scan through all the other constraints, and for each one, compute its intersection point with this line and whether it is "facing" right or left (i.e., which side of that point satisfies the constraint). We find the rightmost intersection point of all the constraints facing to the right and the leftmost intersection point of all that are facing left. If they cross, then there is no solution. Otherwise, the solution is whichever endpoint gives a better value of $\mathbf{c} \cdot \mathbf{x}$ (if they give the same value – i.e., the line $\mathbf{a}_m \cdot \mathbf{x} = b_m$ is perpendicular to $\mathbf{c}$ – then say let's take the rightmost point). In the example above, the 1-dimensional problem is the one in the figure below, with the green constraints "facing" one direction and the blue ones facing the other way. The direction of $\mathbf{c}$ means the optimal point is given by the "lowest" green constraint.



The total time taken here is $O(m)$ since we have $m - 1$ constraints to scan through and it takes $O(1)$ time to process each one.

Right now, this looks like an $O(m^2)$-time algorithm for $d = 2$, since we have potentially taken $O(m)$ time to add in a single new constraint if Case 2 occurs. But, suppose we add the constraints in *a random order*? What is the probability that constraint m goes to Case 2?

Notice that the optimal solution to all m constraints (assuming the LP is feasible and bounded) is at a corner of the feasible region, and this corner is defined by two constraints, namely the two sides of the polygon that meet at that point. If both of those two constraints have been seen already, then we are guaranteed to be in Case 1. So, if we are inserting constraints in a random order, the probability we are in Case 2 when we get to constraint m is at most $2/m$. This means that the *expected* cost of inserting the mth constraint is at most:

$$\text{E[cost of inserting } m\text{th constraint]} \leq (1 - 2/m)O(1) + (2/m)O(m) = O(1).$$

This is sometimes called "backwards analysis" since what we are saying is that if we go backwards and pluck out a random constraint from the m we have, the chance it was one of the constraints that mattered was at most $2/m$.

So, Seidel's algorithm is as follows. Place the constraints in a random order and insert them one at a time, keeping track of the best solution so far as above. We just showed that the expected cost of the ith insert is $O(1)$ (or if you prefer, we showed $T(m) = O(1) + T(m-1)$ where $T(i)$ is the expected cost of a problem with i constraints), so the overall expected cost is $O(m)$.

1.1 Handling Special Cases, and Extension to Higher Dimensions

(We will not be testing you on this part, but you should try to understand it all the same.)

What if the LP is infeasible? There are two ways we can analyze this. One is that if the LP is infeasible, then it turns out this is determined by at most 3 constraints. So we get the same as above with $2/m$ replaced by $3/m$. Another way to analyze this is imagine we have a separate account we can use to pay for the event that we get to Case 2 and find that the LP is infeasible. Since that can only happen once in the entire process (once we determine the LP is infeasible, we stop), this just provides an additive $O(m)$ term. To put it another way, if the system is infeasible, then there will be two cases for the final constraint: (a) it was feasible until then, in which case we pay $O(m)$ out of the extra budget (but the above analysis applies to the the (feasible) first $m-1$ constraints), or (b) it was infeasible already in which case we already halted so we pay 0.

What about unboundedness? One way we can deal with this is put everything inside a bounding box $-\lambda \leq x_i \leq \lambda$ (so, for instance, if all c_i are positive then the initial $\mathbf{x}^* = (\lambda, \ldots, \lambda)$) where we view λ symbolically as a limit quantity. For example, in 2-dimensions, if $\mathbf{c} = (0, 1)$ and we have a constraint like $2x_1 + x_2 \leq 8$, then we would see it is not satisfied by (λ, λ), intersect the contraint with the box and update to $\mathbf{x}^* = (4 - \lambda/2, \lambda)$.

So far we have shown that for $d = 2$, the expected running time of the algorithm is $O(m)$. For general values of d, there are two main changes. First, the probability that constraint m enters Case 2 is now d/m rather than $2/m$. Second, we need to compute the time to perform the update in Case 2. Notice, however, that this is a $(d-1)$-dimensional linear programming problem, and so we can use the same algorithm recursively, after we have spent $O(dm)$ time to project each of the $m-1$ constraints onto the $(d-1)$-dimensional hyperplane $\mathbf{a}_m \cdot \mathbf{x} = b_m$. Putting this together we have a recurrence for expected running time:

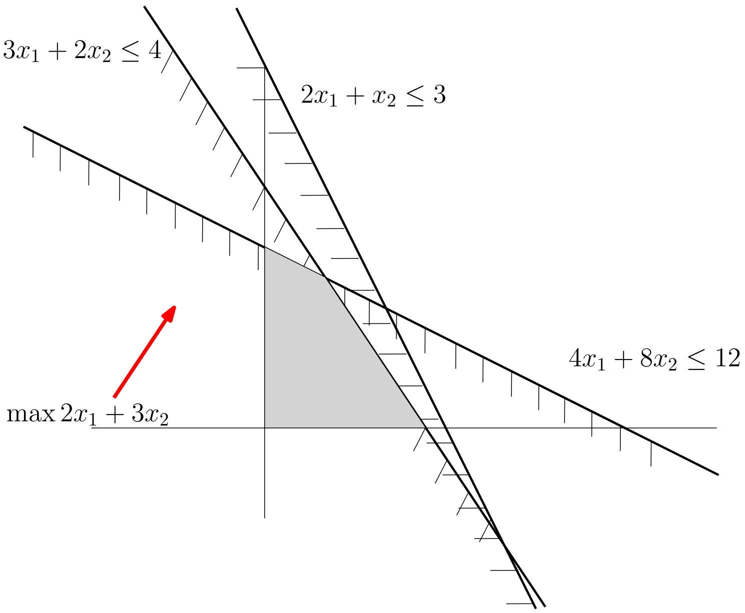
$$T(d, m) \ \leq \ T(d, m-1) + O(d) + \frac{d}{m}[O(dm) + T(d-1, m-1)].$$

This then solves to $T(d, m) = O(d!m)$.

2 Linear Programming Duality

Consider the following LP

$$\begin{aligned} P = \max(2x_1 + 3x_2) \\ \text{s.t.} \quad 4x_1 + 8x_2 \leq 12 \\ 2x_1 + x_2 \leq 3 \\ 3x_1 + 2x_2 \leq 4 \\ x_1, x_2 \geq 0 \end{aligned}$$



In an attempt to solve P we can produce upper bounds on its optimal value.

- Since $2x_1 + 3x_2 \leq 4x_1 + 8x_2 \leq 12$, we know $\mathrm{OPT}(P) \leq 12$.
- Since $2x_1 + 3x_2 \leq \frac{1}{2}(4x_1 + 8x_2) \leq 6$, we know $\mathrm{OPT}(P) \leq 6$.
- Since $2x_1 + 3x_2 \leq \frac{1}{3}((4x_1 + 8x_2) + (2x_1 + x_2)) \leq 5$, we know $\mathrm{OPT}(P) \leq 5$.

In each of these cases we take a positive linear combination of the constraints, looking for better and better bounds on the maximum possible value of $2x_1 + 3x_2$. We can formalize this, letting y_1, y_2, y_3 be the coefficients of our linear combination. Then we must have

$$\begin{aligned} 4y_1 + 2y_2 + 3y_2 \geq 2 \\ 8y_1 + y_2 + 2y_3 \geq 3 \\ y_1, y_2, y_3 \geq 0 \\ \text{and we seek } \min(12y_1 + 3y_2 + 4y_3) \end{aligned}$$

This too is an LP! We refer to this LP as the "dual" and the original LP as the "primal". We designed the dual to serve as a method of constructing an upper bound on the optimal value of the primal, so if y is a feasible solution for the dual and x is a feasible solution for the primal, then $2x_1 + 3x_2 \leq 12y_1 + 3y_2 + 4y_3$. If we can find two feasible solutions that make these equal, then we know we have found the optimal values of these LP.

In this case the feasible solutions $x_1 = \frac{1}{2}, x_2 = \frac{5}{4}$ and $y_1 = \frac{5}{16}, y_2 = 0, y_3 = \frac{1}{4}$ give the same value 4.75, which therefore must be the optimal value.

In general, we might start off with a "primal" LP:

$$\begin{aligned} \text{maximize } & \mathbf{c}^T\mathbf{x} \\ \text{subject to } & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0}, \end{aligned} \tag{1}$$

As above, the $\mathbf{x}$ variables are constrained to non-negative. If you go through an essentially identical exercise, you will get the dual LP is:

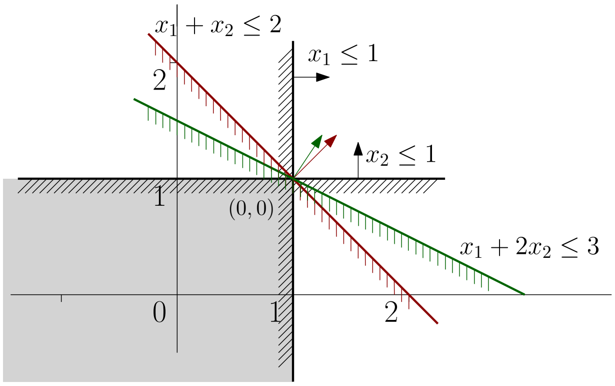
$$\begin{aligned} \text{minimize } & \mathbf{y}^T\mathbf{b} \\ \text{subject to } & \mathbf{y}^T A \geq \mathbf{c}^T \\ & \mathbf{y} \geq \mathbf{0}, \end{aligned} \tag{2}$$

The "magic" theorem that we will not prove in this course, but that is not difficult to show is this: under some mild assumptions (that the primal LP is feasible and bounded) the optimal solution to the dual LP has value exactly equal to the primal optimal value. This is called "*strong duality*"; let's give a geometric intuition of why this is true in the next section.

2.1 A Geometric Viewpoint

Given a set of constraints like $\mathbf{a}_1 \cdot \mathbf{x} \leq b_1$ and $\mathbf{a}_2 \cdot \mathbf{x} \leq b_2$, notice that you can add them to create more constraints that have to hold, like $(\mathbf{a}_1 + \mathbf{a}_2) \cdot \mathbf{x} \leq b_1 + b_2$, or $(0.7\mathbf{a}_1 + 2.9\mathbf{a}_2) \cdot \mathbf{x} \leq (0.7\mathbf{b}_1 + 2.9\mathbf{b}_2)$. In fact, any positive linear combination has to hold.

To get a feel of what this looks like geometrically, say we start with constraints $x_1 \leq 1$ and $x_2 \leq 1$. These imply $x_1 + x_2 \leq 2$ (the red inequality), $x_1 + 2x_2 \leq 3$ (the green one), etc.



In fact, you can create any constraint running through the intersection point $(1,1)$ that has the entire feasible region on one side by using different positive linear combinations of these inequalities.

Now, suppose you have a linear program in n variables (we've switched back from "d" to "n") with objective $\mathbf{c} \cdot \mathbf{x}$ to maximize. As we've discussed, unless the feasible region is unbounded (and let's assume for this entire discussion that the feasible region is bounded), the optimum point will occur at some vertex $\mathbf{x}^*$ of the feasible region, which is an intersection of n of the constraints, and have some value $v^* = \mathbf{c} \cdot \mathbf{x}^*$.

Consider the n *inequality* constraints that define the vertex $\mathbf{x}^*$, say these are

$$\mathbf{a}_1 \cdot \mathbf{x} \leq b_1, \quad \mathbf{a}_2 \cdot \mathbf{x} \leq b_2, \quad \ldots, \quad \mathbf{a}_n \cdot \mathbf{x} \leq b_n,$$

so that for each $i \in \{1, 2, \ldots, n\}$ the point $\mathbf{x}^*$ satisfies the *equalities*

$$\mathbf{a}_1 \cdot \mathbf{x} = b_1, \quad \mathbf{a}_2 \cdot \mathbf{x} = b_2, \quad \ldots, \quad \mathbf{a}_n \cdot \mathbf{x} = b_n.$$

An interesting fact is that just as in the simple example above, if you take these n inequality constraints that define the vertex $\mathbf{x}^*$ and look at all positive linear combinations of these, you can again create any constraint you want going through $\mathbf{x}^*$ that has the entire feasible region on one side. One such constraint is $\mathbf{c} \cdot \mathbf{x} \leq v^*$. It goes through $\mathbf{x}^*$ (since we have $\mathbf{c} \cdot \mathbf{x}^* = v^*$) and every point in the feasible region is contained in it (since no feasible point has value more than v^*). So it is possible to create the constraint $\mathbf{c} \cdot \mathbf{x} \leq v^*$ using some positive linear combination of the $\mathbf{a}_i \cdot \mathbf{x} \leq b_i$ constraints.

Why is this interesting? We've shown a *short proof* (a "succinct certificate") that $\mathbf{x}^*$ is optimal. Indeed, if I gave you a solution $\mathbf{x}^*$ and claimed it was optimal for the given constraints and the objective function $\mathbf{c} \cdot \mathbf{x}$, it is not clear how I would convince you that indeed this is the highest value I can get. In 2-dimensions I could draw a figure, but in higher dimensions things get more difficult. But what we've just shown is that I can exhibit a positive linear combination of the given constraints that creates the constraint $\mathbf{c} \cdot \mathbf{x} \leq v^* = \mathbf{c} \cdot \mathbf{x}^*$, hence showing we can't do any better.[1]

How do we find this positive linear combination of the constraints? Hey, it's actually just another linear program. Let's see how. Suppose we want to find the best possible bound $\mathbf{c} \cdot \mathbf{x} \leq v$ for as small a value v as possible. So, let's say the original LP had the m constraints

$$\mathbf{a}_1 \cdot \mathbf{x} \leq b_1, \quad \mathbf{a}_2 \cdot \mathbf{x} \leq b_2, \quad \ldots, \quad \mathbf{a}_m \cdot \mathbf{x} \leq b_m,$$

written compactly as $A\mathbf{x} \leq \mathbf{b}$.[2] What's our goal? We want to find positive values $y_1, y_2, \ldots, y_m$ such that

$$\sum_i y_i \mathbf{a}_i = \mathbf{c}.$$

From this positive linear combination we can infer the upper bound

$$\mathbf{c} \cdot \mathbf{x} = (\sum_i y_i \mathbf{a}_i) \cdot \mathbf{x} \leq \sum_i y_i b_i.$$

And we want this upper bound to be as "tight" (i.e., small) as possible, so let's solve the LP:

$$\min \sum_i b_i y_i \qquad \text{subject to} \qquad \sum_i y_i \mathbf{a}_i = \mathbf{c}.$$

(In matrix notation, if $\mathbf{y}$ is a $m \times 1$ column vector consisting of the y_i variables, then we want to minimize $\mathbf{y}^T\mathbf{b}$ subject to $\mathbf{y}^T A = \mathbf{c}$.)

Let us summarize: we started off with the "primal" LP,

$$\begin{aligned} &\text{maximize } \mathbf{c}^T\mathbf{x} \\ &\text{subject to} \quad A\mathbf{x} \leq \mathbf{b} \end{aligned} \tag{3}$$

[1] It's like showing that a flow is maximal by exhibiting a matching min-cut proving we can't do better. Or showing a minimax optimal policy for a row-player in a zero-sum game is best possible by showing a matching minimax optimal policy for the column player. This analogy is actually no coincidence: maxflow-mincut is a special case of what we're showing.

[2] We use "$\geq$" for vectors to mean that the LHS is greater than or equal to the RHS in every coordinate, and similarly for "$\leq$".

and were trying to find the best bound on the optimal value of this LP. And to do this, we wrote the "dual" LP:

$$\begin{aligned} \text{minimize } & \mathbf{y}^T\mathbf{b} \\ \text{subject to } \quad & \mathbf{y}^T A = \mathbf{c}^T \\ & \mathbf{y} \geq \mathbf{0}. \end{aligned} \tag{4}$$

Note that this primal/dual pair looks slightly different from the pair (1) and (2). There the primal had non-negativity constraints, and the dual had an inequality. Here the variables of the primal are allowed to be negative, and the dual has equalities. But these are just cosmetic differences; the basic principles are the same.

2.2 Shortest Paths

In recitation yesterday, you saw the following LP for computing an s-t shortest path:

$$\begin{aligned} \max \quad & \textstyle\sum_v d_v \\ \text{subject to} \quad & d_s = 0 \\ & d_v - d_u \leq w(u,v) \qquad \forall (u,v) \in E \end{aligned} \tag{5}$$

Since we're setting d_s to zero, we could rewrite the LP as

$$\begin{aligned} \max \quad & \textstyle\sum_{v:v\neq s} d_v \\ \text{subject to} \quad & d_v - d_u \leq w(u,v) \qquad \forall (u,v) \in E, s \notin \{u,v\} \\ & d_v \leq w(s,v) \qquad \forall (s,v) \in E \\ & -d_u \leq w(u,s) \qquad \forall (u,s) \in E \end{aligned} \tag{6}$$

We take the dual of this LP. Let us define $E_s^{out} := \{(s,v) \in E\}$, $E_s^{in} := \{(u,s) \in E\}$, and $E^{rest} := E \setminus (E_s^{out} \cup \mathrm{E}_s^{in})$. For every arc $e = (u,v)$ we will have a variable y_e. We want to get the best upper bound on $\sum_{v\neq s} d_v$ using the constraints, so we should find a solution to

$$\sum_{e\in E^{rest}} y_{uv}(d_v - d_u) + \sum_{e\in E_s^{out}} y_{sv} d_v - \sum_{e\in E_s^{in}} y_{us} d_u = \sum_{v\neq s} d_v \tag{7}$$

(this is like $\mathbf{y}^T A = c$) and we want to minimize

$$\sum_{(u,v)\in E} y_{uv}\, w(u,v). \tag{8}$$

(This is like $\min \mathbf{y}^T\mathbf{b}$.) Hey, the objective function (8) is pretty, but what about the craziness in (7)? It's not so bad, let's see what it is saying. Let's just collect all copies of each of the variables d_v, and it's saying

$$\sum_{v\neq s} d_v \left(\sum_{(u,v)\in E} y_{uz} - \sum_{(v,w)\in E} y_{vw} \right) = \sum_{v\neq s} d_v.$$

and since these equalities must hold regardless of the d_v values, this is really the same as

$$\sum_{(u,v)\in E} y_{uv} - \sum_{(v,w)\in E} y_{vw} = 1 \qquad \forall v \neq s.$$

So the final dual LP is:

$$\begin{aligned} &\min \sum_{u,v \in E} y_{uv}\, w(u,v) && && (9) \\ \text{subject to} \quad & \sum_{(u,v)\in E} y_{uv} - \sum_{(v,w)\in E} y_{vw} = 1 && \forall v \neq s \end{aligned}$$

Hence, if you think of u_{uv} as the flow being sent along the arc (u, v), this is saying that every node except s has excess 1. So s is sending $(n - 1)$ units of flow, one unit ends up at each other node. And we are minimizing the total cost of the flow, where the cost of sending flow on arc e is $w(e)$. Since there are no capacity constraints, we should send flow along the shortest paths to minimize the cost of this unit of flow. The dual LP is finding the shortest path from s to v, for each $v \neq s$.

2.2.1 Relating it to the Other LP from Recitation

Recall that we wrote this LP for the shortest s-t path:

$$\begin{aligned} LP_t := \min \textstyle\sum_e c_e x_e & && (10) \\ \text{subject to} \quad \textstyle\sum_{w:(s,w)\in E} x_{sw} = 1 & && \\ \textstyle\sum_{v:(v,t)\in E} x_{vt} = 1 & && \\ \textstyle\sum_{v:(u,v)\in E} x_{uv} = \textstyle\sum_{v:(v,w)\in E} x_{vw} & \quad \forall w \in V \setminus \{s,t\} && (11) \\ x_e \geq 0. & && \end{aligned}$$

It's sending one unit of flow from s to t. And the objective function value is the shortest s-t path length. If you take one such LP for each $t \neq s$ and sum them up, you get the LP in (9). And it's computing the sum of the shortest path distances from s to everyone else.

Of course, (9) is the dual of the LP (5). Which is also computing the sum of the shortest path distances from s to everyone else. Strong duality strikes again.