Lecture 22: Primal-Dual Algorithm for Shortest Paths

Lecturer: *Sundar Vishwanathan*
Computer Science & Engineering Indian Institute of Technology, Bombay

In this lecture we will design an algorithm for the classical shortest path problem.
Input: A directed graph with positive integer weights w_{uv} on edges; two special vertices, $s, t \in V$.
Output: A shortest path from s to t.

Our first step is to write an ILP. We choose one variable per edge, x_{uv}. If x_{uv} is picked, $x_{uv} = 1$ else $x_{uv} = 0$.

The cost function is min $\sum_{u,v} x_{uv} w_{uv}$ We will assume that there are no edges entering s or leaving t. The Constraints:

1. $\sum_u x_{su} = 1$ (The number of edges leaving s is 1)
2. $\sum_v x_{vt} = 1$ (The number of edges entering t is 1)
3. $\forall p \in V - \{s, t\}$ $\sum_q x_{pq} - \sum_r x_{rp} = 0$ (For all other vertices, the number of edges leaving them equals the number of edges entering them.)
4. For every edge uv, $0 \leq x_{uv} \leq 1$, x, x_{uv} integral.

We drop the constraint $x_{uv} \leq 1$ since it does not affect the outcome. We then drop the integrality constraint and write the dual. Writing the dual is a little bit tricky (watch the signs carefully). The resulting dual will have one variable for each vertex in the graph.

Why the dual? The reason is that the cost now appears in the RHS. And in the template, in the crucial design step, we only pick a few of the constraints and the RHS is zero. In other words, we will be left with designing a shortest path algorithm for a subgraph when the cost on the edges is zero. Clearly that is easy.

1 The Dual

$$\max y_t - y_s$$
$$\forall u, v \; y_v - y_u \leq w_{uv}$$

Where does the minus sign come in to yield $-y_s$?

Again, we will try and only increment the y_us. Looking at the cost function, we see that at each stage we will increment y_t.

Algorithm

We start with all y_us as zero. In the first step, we need to increment y_t. We can continue to increment this till the minimum weight edge entering t becomes tight.

Can you now design the generic step that you will put in a loop?

Consider all edges which are tight. That is edges for which $y_v - y_u = w_{uv}$. We need to increment y_t subject to the condition that for all such edges the increment at y_v is at most the increment at y_u. It is easy to see that if we increase the value of y_t by some amount we need to increase the value of every vertex u such that there is a path from u to t via tight edges.

This we put in a loop and here is the algorithm.

1. *Initialisation.* Set all y_us to zero.
2. *Iterative Step.* Consider the graph $G' = (V, E')$ of tight edges. In this graph find all vertices from which t is reachable. Increase the value of y for all these vertices till some edge becomes tight. Stop when s is in G'.

Finally we may have more than a path among the tight edges. So we need to extract a path. As before we will need a reverse delete step.
Reverse Delete: In this case you can either do the reverse delete as in the MST or just find any path from s to t among tight edges.

Marbles and Strings analogy

Here is some intuition. This is for undirected graphs. Consider the vertices of the graph as marbles and the edges as inextensible strings (wound to the marbles) with lengths equal to the value of the weights of the corresponding edges. Hold s in one fist and t in the other. Now pull them apart as much as you can. Assume that you have extensible hands. How far apart can you pull them? The answer is as much as the distance of a shortest path between s and t. You can see that the problem of pulling them as far apart as possible is a maximization problem. The answer is the optimum of a minimization problem. These are the primal and dual problems we have written. Indeed, y_t tells you how far apart you can pull t. The value y_u for vertex u is interpreted as the distance of node u from the node s.

Once you have pulled them apart, which strings will be tight? Precisely those that take part in a shortest path between s and t. This is complementary slackness. In the dual, for these strings we have equality. In the primal, we will pick these in a shortest path- $x_e > 0$.

This also gives an intuitive explanation of Dijkstra's algorithm.

We start by assigning the values of $y_u = 0$, $\forall u$ represented by all the marbles being at the one position. All the strings are slack initially.

Now, maintaining the value of all marbles except t at 0, we pull t. That is we raise y_t. This is done till one of the strings joining marble t becomes taut. Suppose it is the string that joins marble m to t. After this point we can no longer pull t any further away from m. if dual feasibilty is to be maintained. So we fix the position of this marble m relative to t and start pulling both t and m together. This process is continued until we find that one of the strings attached to marble s has become taut. At this point, all the strings corresponding to the edges that belong to any shortest path are taut.

Proof that the value of primal optimal is equal to the value of dual optimal:

$$\sum_{(u,v)\, chosen} w_{uv} = \sum_{(u,v)\, chosen} (y_u - y_v) = y_t - y_s.$$