Shortest Paths in Directed Graphs

Chester P. Lampwick
clampwic@iastate.edu

Derrick Stolee
dstolee@iastate.edu

August 25, 2013

1 Problem

A *uv-path* in a digraph G is a sequence of vertices $v_0 v_1 \dots v_k$ such that $v_0 = u$, $v_k = v$ and for each $i \in \{1, \dots, k\}$, the pair $v_{i-1}v_i$ is a directed edge in G. For a weight function w, the weight of the path is $\sum_{i=1}^{k} w(v_{i-1}, v_i)$. Our goal is to minimize this weight among all st-paths.
We will assume that $V(G) = [n] = \{0, \dots, n-1\}$, $s = 0$, and $t = n-1$. For each edge $ij \in E(G)$, we have a real variable $x_{i,j}$ whose value is nonnegative. These variables represent how much of an edge we use in an st-path.
We treat the feasible solutions as a *flow* from s to t of value one. Thus, we have *conservation* constraints at every vertex i with $s < i < t$: $\sum_{j:ji\in E(G)} x_{j,i} - \sum_{j:ij\in E(G)} x_{i,j} = 0$. This enforces that no flow is gained or lost at any vertex other than s or t. We then guarantee that exactly 1 unit of flow reaches the vertex t: $\sum_{j:jt\in E(G)} x_{j,t} - \sum_{j:tj\in E(G)} x_{t,j} = 1$. If we sum up all of these constraints, every edge $x_{i,j}$ where $i \neq s$ appears exactly once as $+x_{i,j}$ and once as $-x_{i,j}$, and hence we are left with the constraint $\sum_{i:si\in E(G)} x_{s,i} - \sum_{i:is\in E(G)} x_{i,s} = 1$. This guarantees that exactly 1 unit of flow leaves the vertex s. Since this last constraint is dependent on the other constraints, we omit it in our primal formulation. Finally, we aim to minimize the weight of our path by minimizing the sum $\sum_{ij\in E(G)} w(i,j)x_{i,j}$. This results in the following linear program:

$$
\begin{aligned}
\min \sum_{ij\in E(G)} & w(i,j)x_{i,j} \\
\text{subject to } (\sum_{i:si\in E(G)} x_{s,i} - \sum_{i:is\in E(G)} & x_{i,s} = 1) \\
\sum_{j:j1\in E(G)} x_{j,1} - \sum_{j:1j\in E(G)} & x_{1,j} = 0 \\
\sum_{j:j2\in E(G)} x_{j,2} - \sum_{j:2j\in E(G)} & x_{2,j} = 0 \\
& \vdots \quad \vdots \\
\sum_{j:j(n-2)\in E(G)} x_{j,n-2} - \sum_{j:(n-2)j\in E(G)} & x_{2,n-2} = 0 \\
\sum_{j:jt\in E(G)} x_{j,t} - \sum_{j:tj\in E(G)} & x_{t,j} = 1 \\
& x_{i,j} \geq 0
\end{aligned}
$$

To take the dual, observe that every constraint of the primal is associated with a vertex $i \in [n]$, and hence the dual has a variable y_i for every vertex $i \in [n]$. Since the primal has equality constraints, these variables are free. (However, since the constraint associated with the vertex s is vacuous, we can assume the variable $y_s = 0$ and omit it from the dual problem.) Every variable in the primal is associated with a specific edge $ij \in E(G)$, and the variable $x_{i,j}$ appears in two constraints (with coefficient -1 in the constraint for i and with coefficient $+1$ in the constraint for j). Since the variable $x_{i,j}$ is nonnegative, the constraint for ij is an upper bound. Thus, for every edge $ij \in E(G)$, the dual problem has a constraint $y_j - y_i \leq w(i,j)$. Finally, since the right-hand side of the primal has value 0 except for the constraint at t, we have our optimization function equal to y_t. Therefore, our dual problem is given as

$$\begin{aligned} &\max y_t \\ \text{subject to } & y_j - y_i \leq w(i,j) \qquad\qquad \text{for all } ij \in E(G) \\ y_1, \dots, y_{n-2}, & y_t \quad \text{free} \\ & y_s = 0 \end{aligned}$$

A combinatorial interpretation of this dual problem is that we wish to place the vertices along the number line. We want to maximize the position of t subject to the vertex s fixed at 0. Our constraints limit how far to the right a vertex can be to its incoming neighbors. That is, if ij is an edge, then j can be no more than $w(i,j)$ to the right of i. Essentially, if we imagine the edges as ropes of given lengths and we "stretch" the graph by pulling t in the positive direction, the edges will eventually become "tight" and prevent the vertices from extending farther.

The way we will produce the dual solution is to assign y_i to be the length of a shortest path from s to i.

Lemma 1. *Let $M = \sum_{ij \in E(G)} w(i,j)$. If there exists a path from s to t, then assigning*

$$y_i = \begin{cases} d(s,i) & \textit{if } d(s,i) < \infty \\ M & \textit{otherwise} \end{cases}$$

is an optimal dual solution. If there is no path from s to t, then the dual problem is unbounded.

Proof. First, assume there is a path from s to t, so $d(s,t) < \infty$. By weak duality, we know that $d(s,t)$ is an upper bound on $\max y_t$, so if this solution is feasible then it is also optimal. If $d(s,i) = \infty$, then all vertices j such that $ji \in E(G)$ also have $d(s,j) = \infty$. Thus, $y_i - y_j = M - M = 0 \leq w(i,j)$. If $d(s,i) < \infty$, then observe $d(s,i) = \min\{d(s,j) + w(j,i) : ji \in E(G)\}$. Thus, $y_i = d(s,i) \leq d(s,j) + w(j,i) = y_j + w(j,i)$ for all such edges $ji \in E(G)$ where $d(s,j) < \infty$. If $ji \in E(G)$ and $d(s,j) = \infty$, then $y_i = d(s,i) < M + w(j,i)$ and the constraint on the edge ji holds. Thus, this solution is feasible.

Now, assume that there is no path from s to t. We assign $y_i = 0$ for all vertices with $d(s,i) < \infty$, and then assign $y_j = y_t$ for all other vertices, and let y_t be arbitrarily large. If $y_j - y_i > 0$, then $y_i = 0$ and $y_j = y_t$. However, this implies that i is reachable from s and j is not, so the edge ij does not exist. Thus, $y_j - y_i \leq 0 \leq w(i,j)$ for all edges $ij \in E(G)$. $\square$

2 Implementation

We implemented Dijkstra's algorithm using the Sage environment, specifically the Sage Graph Library [2]. We used the pseudocode in the Wikipedia article on Dijkstra's Algorithm [1] to help

with our implementation. While using this as a base, we had to slightly adjust our definition of the dual problem in order to more easily extract a dual solution, using Lemma 1.
To run our shortest path algorithm on a graph G, a weight function w, and vertices s and t, run `ShortestPath(G,w,s,t)`. This method returns three values: a number d, a list P, and a dictionary D. The number d is the weight of a shortest st-path in G, with respect to w. The list P is an ordered list of the vertices in an st-path of weight d. The dictionary D provides dual values for the vertices of G, where $D[i]$ stores the distance $d(s, i)$, as given by Lemma 1.
Here is our implementation of Dijkstra's algorithm, as the procedure `ShortestPath`:

```
def ShortestPath(G, s, t, w):
    D = {};
    for i in G.vertices():
        D[i] = Infinity;
    D[s] = 0.0;

    Predecessor = {};
    Predecessor[s] = None;

    Q = [s];
    while len(Q) > 0:
        # select the next vertex to use, by min label
        i = Q[0];
        for j in Q:
            if D[j] < D[i]:
                i = j;
        Q.remove(i);

        for j in G.neighbors_out(i):
            if D[j] > float(D[i]) + float(w[ (i,j) ]):
                D[j] = float(D[i]) + float(w[(i,j)]);
                Predecessor[j] = i;
                if not j in Q:
                    Q.append(j);

    # build the path
    P = [];
    if D[t] < Infinity:
        i = t;
        P.append(i);
        while Predecessor[i] is not None:
            i = Predecessor[i];
            P.append(i);
        P.reverse();

    return D[t], P, D;
```

See the attached Sage worksheet, `reach.sws`, for all details.

3 Solutions

We now discuss our solutions to the given problem instances.

3.1 Instance I1

Instance I1 is small enough that we were able to find the smallest path by hand. We used this as a test case to verify our algorithm.

```
Shortest distance: 9
Shortest path: [0, 1, 4, 6]
Dual solution:
y_{0} = 0, y_{1} = 3, y_{2} = 5, y_{3} = 7, y_{4} = 8, y_{5} = 8, y_{6} = 9
```

3.2 Instance I2

This instance does NOT have an *st*-path and the dual is unbounded. In our dual "solution" we use 100 as a placeholder for "arbitrarily large."

```
Shortest distance: +Infinity
Shortest path: None!
Dual solution:  (0, 7, 5, 7, 9, 3, 100, 8, 12, 4, 6, 15, 7, 6, 9, 100, 100, 11, 100, 100)
```

3.3 Instance I3

```
Shortest distance: 34.36
Shortest path: [0, 8, 3, 7, 16, 36, 33, 50, 68, 74, 86, 81, 99]
Dual Solution: (0.00, 22.71, 17.13, 0.92, 2.99, 10.14, 20.53, 6.12,
0.19, 9.01, 10.19, 13.15, 11.96, 8.31, 4.87, 14.84, 13.63, 13.94, 27.64,
17.80, 21.89, 6.99, 13.45, 10.69, 11.24, 16.36, 18.47, 20.26, 18.23,
15.29, 18.19, 20.38, 21.73, 17.19, 15.03, 16.50, 16.52, 23.05, 17.43,
19.62, 13.98, 11.91, 23.80, 13.77, 19.43, 16.65, 23.28, 18.70, 20.94,
25.87, 17.47, 24.66, 23.19, 30.95, 19.08, 26.04, 22.95, 25.19, 24.38,
22.89, 22.49, 19.32, 23.69, 25.36, 33.10, 23.61, 29.80, 24.63, 21.39,
31.90, 22.71, 31.45, 29.26, 30.20, 24.70, 36.62, 33.96, 31.98, 35.90,
31.13, 31.64, 27.60, 29.26, 32.29, 27.82, 29.16, 27.34, 32.49, 31.18,
34.57, 27.13, 32.89, 33.36, 36.92, 29.98, 38.83, 39.69, 30.03, 31.56,
34.36)
```

3.4 Instance I4

```
Shortest distance: 131.00
Shortest path: [0, 455, 348, 799, 1201, 742, 1089, 1536, 1970, 1901, 1999]
```

References

[1] Dijkstra's Algorithm, *Wikipedia*, http://en.wikipedia.org/wiki/Dijkstra's_algorithm

[2] The Sage Graph Library, http://www.sagemath.org/doc/constructions/graph_theory.html