ICS 260—Fall, 2000–Class Notes 2: Single Source Shortest Paths

Notes Originally Developed by Sandy Irani
Modified by Michael B. Dillencourt

1 Single Source Shortest Path Problem

Input: A graph G(V,E), with weights on the edges, $w : e \rightarrow \mathbf{R}$. A designated source node $s \in V$.

Output: The length of the shortest path from s to every other node in V. (Alternatively, output a shortest path tree which denotes the shortest path from s to every other node in V).

2 Dijkstra's Algorithm

[CLR, Sections 25.1 and 25.2]

Dijkstra's algorithm works when the weights on the edges are non-negative.

Intuition: The algorithm maintains a subset of the nodes X which it has already 'inspected'. For each vertex $u \in V$, the algorithm maintains an estimate $d[u]$ on the length of the shortest path from the source s to u. The estimate is the length of the shortest path from s to u that only passes through vertices in X before reaching u. Initially, X just contains the vertex s. In each iteration, one more vertex is included in X until X contains all the vertices.

2.1 Correctness

We will show by induction on the number of nodes in X that

1. $d[u]$ is the length of the shortest path from s to u that passes only through vertices in X.
2. If $u \in X$ and $v \notin X$, then $d[u] \leq d[v]$.

Base Case: $X = \{s\}$

Induction Hypothesis: Suppose u is added to X. Since u is picked to have the minimum d value, condition 2 still holds after statement (13) has been executed in the next iteration of the Main Loop.

Now suppose that the shortest path from s to some vertex v that passes only through vertices in X changes with the addition of u to X. Call the new path p.

It must be the case that u is the last vertex from X in the path p. Suppose not. Suppose it passes through $x \in X$ after u and before v:

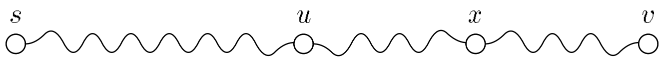
Algorithm $\textsc{Dijkstra}(G, s)$:

Initialization:

$X = \{s\}$
$d[s] = 0$
$prev[s] =$ NULL
for $u \neq s$
    if $(s, u) \in E$
        $d[u] = w(s, u)$
        $prev[u] = s$
    if $(s, u) \notin E$
        $d[u] = \infty$
        $prev[u] =$ NULL

Main Loop:

while $X \neq V$
    Let $u \in V - X$ be such that $d[u]$ is minimum
    $X \leftarrow X \cup \{u\}$
    For all v such that $(u, v) \in E$ and $v \notin X$
        if $d[v] > d[u] + w(u, v)$
            $d[v] \leftarrow d[u] + w(u, v)$
            $prev[v] = u$



This can't be since $d[x] \leq d[u]$ which implies that the path that goes directly to x without passing through u and then to v is shorter than p.

Thus, u is the next to last vertex in p and v is the last vertex in p. This means that (u, v) is an edge and $d[v] > d[u] + w(u, v)$ and $d[v]$ will be updated to be $d[u] + w(u, v)$ which is exactly the length of p. So condition 1 holds after the next iteration of the Main Loop in Dijkstra's algorithm is complete.

Finally, notice that since $d[u] \geq d[x]$ for all $x \in X$, it will still be the case that $d[v] \geq d[x]$ for all $x \in X$. This means that condition 2 will be still satisfied after the next iteration of the Main Loop is complete.

■

2.2 Implementation and Running Time

Initialization takes $O(n)$ time. The nodes can be kept in a priority queue with $d[\cdot]$ as the key value. This means that removing the minimum (as in step 12) and updating a value (as in step 15) can be done in $O(\log n)$ time. The main loop is executed n times, but steps (14-16) are done at most once for every edge. This gives a running time of $O(m \log n)$.

There is an even faster implementation that uses Fibonacci heaps and runs in time $O(m + n \log n)$.

3 Single Source Shortest Paths with Arbitrary Weights

[CLR, Section 25.3]

Is the the shortest path from s to t always well defined if negative weights are allowed?

Not if there are negative cycles — you can keep going around the cycle to get a path of lower value.

Perhaps what we want to do then is find the shortest **simple** path from s to t. A simple path is a path which does not repeat a node. Unfortunately, this problem is NP-hard. (More to come on what specifically that means later in the quarter).

If there are no negative cycles, an algorithm for SSSP will return TRUE with the shortest paths from s to every other vertex in the graph. If there is a negative cycle, it will return FALSE.

3.1 The Bellman-Ford Algorithm

Figure 3.1 shows the Bellman-Ford algorithm which is one of the most common algorithms for SSSP with arbitrary weights:

```
Algorithm BellmanFord(G, s):
Initialization:
d[s] = 0
prev[u] = NULL
for u ≠ s
    d[u] = ∞
    prev[u] = NULL

Main Loop:
while i = 1 to n − 1
    for each (u, v) ∈ E
        if d[v] > d[u] + w(u, v)
            d[v] ← d[u] + w(u, v)
            prev[v] = u

Verification
for each (u, v) ∈ E
    if d[v] > d[u] + w(u, v)
        return FALSE
return TRUE
```

3.2 Correctness

Let $sp[u]$ be the length of the shortest path from s to u in G (if there are no negative cycles in G).

We will show the following two facts

- If G has a negative cycle, then there is no assignment for $d[\cdot]$ which satisfies

$$(*) \qquad\qquad d[v] \le d[u] + w(u, v) \quad \text{for all } (u, v) \in E.$$

- If there is no negative cycle in G, then at the end of Bellman-Ford $d[u] = sp[u]$ for all $u \in V$. Furthermore, condition (*) is satisfied.

Proof of Fact 1. Proof by contradiction. Suppose that we have a negative cycle and the conditions in (*) are satisfied for some values of $d[\cdot]$.

Suppose we have a negative cycle $C = (v_1, v_2, \ldots, v_k, v_1)$. This means that

$$w(v_k, v_1) + \sum_{i=1}^{k-1} w(v_i, v_{i+1}) < 0.$$

Now we will define another weight $w'(u, v) = d[u] - d[v] + w(u, v)$. Condition (*) says that $w'(u, v) \ge$

0 for all $(u, v) \in E$. This would indicate that

$$w'(v_k, v_1) + \sum_{i=1}^{k-1} w'(v_i, v_{i+1}) \geq 0.$$

However, this is a contradiction, since

$$\sum_{e \in C} w(e) = \sum_{e \in C} w'(e).$$

∎

Proof of Fact 2. First we establish that $d[v]$ is an upper bound for the shortest path lengths. That is $d[v] \geq sp[v]$ for all v. This is most easily done by induction on the number of steps. It is certainly true after $d[\cdot]$ is initialized. Now suppose that we have an edge (u, v) and $d[v]$ is updated to $d[u] + w(u, v)$ in step (9). Then

$$sp[v] \leq sp[u] + w(u, v) \leq d[u] + w(u, v) = d[v].$$

Now we will establish the main idea behind the proof

Claim 1 *Suppose there is path from s to u with at most i edges whose length is $sp[u]$. Then after the i^{th} iteration, $d[u] = sp[u]$.*

Since any shortest path has at most $n - 1$ edges, Condition 2 will follow from the claim. This is proven by induction on i.

Basis: $\mathbf{i = 0}$. After initialization $sp[s] = 0$.

Induction: Assume true for i, prove for $i + 1$. Suppose that there is a path p from s to v with $i + 1$ edges whose length is $sp[v]$. Let u be the next to last vertex on the path p.

The portion of the path p from s to u has i edges and has length $sp[u]$. By the induction hypothesis, $d[u] = sp[u]$. (Otherwise there is a shorter path to u and p is not the shortest path to v).

When we consider (u, v) in iteration $i + 1$, if it happens that $d[v] > sp[v]$ beforehand then

$$d[v] > sp[v] = sp[u] + w(u, v) = d[u] + w(u, v).$$

In this case $d[v]$ will be set to $d[u] + w(u, v)$ which is just $sp[v]$.

∎

3.3 Running Time

The running time of Bellman-Ford (as stated above) is $\Theta(nm)$.

3.4 Application: Maximum mean weight cycles

Suppose you have a cycle C in a graph with k edges $e_1, e_2, \ldots, e_k$. The mean weight of the cycle is defined as

$$\frac{\sum_{i=1}^{k} w(e_i)}{k}.$$

The maximum mean cycle problem is as follows:

Input: A graph $G = (V, E)$ with weights on the edges $w : E \to \mathbf{R}$.

Output: λ such that mean weight of the cycle with the maximum mean weight.

- Discussion of how the max mean cycle problem is important for discrete event systems.

Suppose we have a weighted graph G and we want to know if the maximum mean cycle weight is no more than λ.

Define a graph G' which is the same as G, except that the weights on the edges will be $w'(u,v) = \lambda - w(u,v)$. The claim is that the max mean cycle weight of G is at most λ if and only of G' does not have a negative cycle.

Consider a cycle C with k edges $e_1, e_2, \ldots, e_k$. We will show that C has mean weight at most λ in G if and only if it is not a negative cycle in G':

$$\begin{aligned}
\frac{\sum_{i=1}^{k} w(e_i)}{k} &\leq \lambda \\
\sum_{i=1}^{k} w(e_i) &\leq k\lambda \\
0 &\leq k\lambda - \sum_{i=1}^{k} w(e_i) \\
0 &\leq \sum_{i=1}^{k} (\lambda - w(e_i)) \\
0 &\leq \sum_{i=1}^{k} w'(e_i)
\end{aligned}$$

3.5 Application: Systems of Difference Constraints

Source: [AMO, Section 4.2].

3.5.1 System of Difference Constraints

There are n variables $x_1, x_2, \ldots, x_n$ and m inequalities of the form

$$x_{i_k} - x_{j_k} \leq b \qquad \text{for } \; 1 \leq k \leq m.$$

Is there are feasible solution? If so, what is it?

Can solve by setting up a *constraint graph.* There are n nodes, one for each variable and m edges, one for each constraint. If there is a constraint of the the form $x_i - x_j \leq b$, then there is an edge from x_j to x_i labeled with b.

Now we add an extra node s and add an edge with 0 weight from s to each node in the constraint graph. Notice, we have not introduced any extra cycles, since there is no path back to s. Let $d[u]$ be the value which the algorithm returns for the shortest path from s to u.

If the algorithm returns TRUE, then the $d[\cdot]$ values will satisfy the equations. If the algorithm returns FALSE, then there is no solution.

Remember there is no negative cycle if and only if

$$\forall (u,v) \in E \qquad d[v] \leq d[u] + w(u,a).$$

Otherwise no such values exist. These are exactly the conditions of the constraints.

3.5.2 Application for Solving a System of Difference Constraints

Suppose you are trying to schedule telephone operators on a 24 hour cycle. You have studied the demands at different hours of the day and know that at hour i, you need b_i operators on duty for $i = 0, 1, \ldots, 23$. Each operator goes on duty for eight hour shifts which can start at any hour in the day. We want to minimize the number of operators hired.

Let y_i be the number of operators which go on duty at hour i. We want to minimize

$$\sum_{i=0}^{23} y_i,$$

subject to

$$\begin{aligned} y_i &\geq 0 \quad \text{for } \; 0 \leq i \leq 23 \\ y_{i-7} + y_{i-6} + \cdots + y_i &\geq b_i \quad \text{for } \; 7 \leq i \leq 23 \\ y_{17+i} + \cdots + y_{23} + y_0 + \cdots + y_i &\geq b_i \quad \text{for } \; 0 \leq i \leq 6 \end{aligned}$$

Suppose we just want to know if there is a schedule which satisfies the requirements and only employs p operators. We will transform this problem into system of difference constraints. Let $x_i = \sum_{j=0}^{i} y_j$.

Each equation gets transformed as follows:

$$\begin{aligned} x_i - x_{i-1} &\geq 0 \quad \text{for } 0 \leq i \leq 23 \\ x_i - x_{i-8} &\geq b_i \quad \text{for } 8 \leq i \leq 23 \\ p - x_{16+i} + x_i &\geq b_i \quad \text{for } 0 \leq i \leq 7 \end{aligned}$$

This is a system of difference constraints which can be solved using a negative cycle detection algorithm.

4 Algorithms for computing the maximum mean cycle

The Bellman-Ford algorithm is, at present, the algorithm with the best worst-case asymptotic running time of any algorithm which solves SSSP and is *strongly polynomial.* A strongly polynomial algorithm has a running time which is bounded by a polynomial in n and m (i.e. the number of numbers needed to specify the input).

There are other algorithms for computing the maximum mean cycle that perform well whose running time is also a function of w_{max} or $\log(w_{max})$ (assuming integer weights). Note that if an algorithm's running time is bounded by a polynomial in n , m and $\log(w_{max})$, then it is still polynomial in the number of bits necessary to specify the problem.

4.1 Lawler's Algorithm

As we showed earlier, the max mean cycle has mean at most λ if and only if the graph with weights $\lambda - w(u,v)$ has a negative cycle. Now to find the mean of the max mean cycle, we can do a binary search on λ:

Running time is $O(mn)$ per iteration. The number of iterations is $\log((w_{max} - w_{min})/\epsilon)$. So the total running time is $O(mn \log((w_{max} - w_{min})/\epsilon))$.

Note that this algorithm does not find the max mean cycle weight exactly — only to within a precision value of ϵ.

4.2 Karp's Algorithm

Here is another algorithm due to Karp for finding the maximum mean weight cycle. The algorithm actually finds the minimum mean weight cycle, but the two problems (max mean weight cycle and min mean weight cycle) are very closely related. If you want to find the max mean weight cycle but only have an algorithm for min mean weight cycle, you can negate the edge weights of the graph. Then the min mean weight cycle in the transformed graph is the max mean weight cycle in the original graph.

The algorithm is based on the following lemma which we will state without proof.

Algorithm $\textsc{Lawler}(G)$:
Initialization:
(1) $L \leftarrow$ weight of minimum weight arc.
(2) $H \leftarrow$ weight of maximum weight arc.
(3) Let s be an arbitrary node in G

Main Loop:
(4) **while** $H - L > \epsilon$
(5) $\quad \lambda \leftarrow (H+L)/2$.
(6) $\quad$ Let G_λ be the graph obtained from G
(7) $\quad$ such that $w_\lambda(u,v) = \lambda - w(u,v)$

(8) $\qquad$ **if** there is no negative cycle in G_λ,
$\qquad$ (i.e. if $\textsc{BellmanFord}(G_\lambda, s)$),
(9) $\qquad\quad H \leftarrow \lambda$
(10) $\qquad$ **else** $L \leftarrow \lambda$
(11) **return** λ.

Theorem 1 (Karp's Theorem) *Let $D_k(v)$ be the length of the shortest path with exactly k edges from s to v. If no such path exists, then $D_k(v) = +\infty$. The mean weight of the minimum mean weight cycle is exactly*

$$\min_{v \in V} \max_{0 < k \leq n-1} \frac{D_n(v) - D_k(v)}{n-k}.$$

To find the mean weight of the min mean weight cycle, compute the $D_v(\cdot)$'s and then evaluate the expression in the theorem.

`Algorithm` KARP(G):

Initialization:

(1) **for** each $k = 0$ to n do

(2) **for** each node $u \in V$ do

(3) $\quad D_k(u) \leftarrow +\infty$

(4) $D_0(s) = 0$.

Compute Distances:

(5) **for** each $k = 0$ to n do

(6) $\quad$ **for** each node $u \in V$ do

(7) $\qquad$ **for** each edge $(u, v) \in E$ do

(8) $\qquad\quad D_k(v) \leftarrow \min\{D_k(v), D_{k-1}(u) + w(u, v)\}$

Compute Minimum Mean Cycle Wieght:

(9) $\lambda \leftarrow +\infty$

(10) **for** each node $u \in V$ do

(11) $\quad \lambda' \leftarrow -\infty$

(12) $\quad$ **for** each $k = 0$ to $n - 1$ do

(13) $\qquad \lambda' \leftarrow \max\{\lambda', (D_n(u) - D_k(u))/(n - k)\}$

(14) $\quad \lambda \leftarrow \min\{\lambda', \lambda\}$