

Lecture 9 — January 30, 2012

Prof. Will Evans

Scribe: Hao Yang

In this lecture we discussed:

- Johnson's reweighting algorithm
- A* search meets graph theory

1 Johnson's Reweighting Algorithm

Idea

- Reweight edges of input graph so that 1) All new edge weights are ≥ 0 2) All shortest paths are the same.
- Run Dijkstra's Algorithm n times using each vertex as a source.

Observation

Adding a constant to all edge weights to make them positive does not preserve shortest paths since the length of paths that have many edges (but might be shortest paths) are increased more than the length of paths with few edges.

Solution

- Form G' from $G = (V, E)$ by adding a new vertex $s \notin V$ and edges (s, v) for all $v \in V$, with $w(s, v) = 0$.
- Find shortest paths from s to all $v \in V$ using Bellman-Ford algorithm on G' . Let $D[i]$ be the shortest path length from s to i .
- Let $\hat{w}(i, j) = w(i, j) + D[i] - D[j]$ for all $(i, j) \in E$.
- Run Dijkstra's algorithm on G with weight \hat{w} for every source vertex $v \in V$.

For step 1 to 3, $T(n) = O(mn)$. For step 4, $T(n) = O(n(n \log n + m))$ (if Fibonacci Heap is used). In total, $T(n) = O(n^2 \log n + mn)$. When m is small (G is a sparse graph), Johnson's reweighting algorithm is faster than Floyd-Warshall's algorithm whose running time is $O(n^3)$.

Proof

First we need to prove the shortest paths remain the same after the reweighting. \Leftrightarrow Shortest path in G using weights w is shortest path in G using weights \hat{w} . Suppose $a = v_0 v_1 \cdots v_k = b$ is a path from a to b in G . Using w : the path length is $W = w(v_0, v_1) + w(v_1, v_2) + \cdots + w(v_{k-1}, v_k)$. Using \hat{w} , the path length $\hat{W} = \hat{w}(v_0, v_1) + \hat{w}(v_1, v_2) + \cdots + \hat{w}(v_{k-1}, v_k) = w(v_0, v_1) + D[v_0] - D[v_1] + w(v_1, v_2) +$

$D[v_1] - D[v_2] + \dots + w(v_{k-1}, v_k) + D[v_{k-1}] - D[v_k] = W + D[v_0] - D[v_k] = W + D[a] - D[b]$. The path P from a to b that minimizes the path length $w(P)$ is the same path that minimizes the path length $\hat{w}(P) = w(P) + D[a] - D[b]$ since a and b remain the same for all such paths.

Secondly we need to prove $\hat{w}(i, j) \geq 0$ so that we can use Dijkstra's algorithm. Because $D[j]$ is the shortest path length from s to j , then $D[i] + w(i, j) \geq D[j] \Rightarrow w(i, j) + D[i] - D[j] = \hat{w}(i, j) \geq 0$,

2 A* search meets graph theory

Problem Finding shortest path from s to t in a big graph G (edges are positive).

Options

- Preprocess G and store all answers to all queries (s, t) . This takes too much space ($\Omega(n^2)$).
- Run Dijkstra's algorithm on the graph for each query. This is too slow even if we run the algorithm in a bidirectional way.

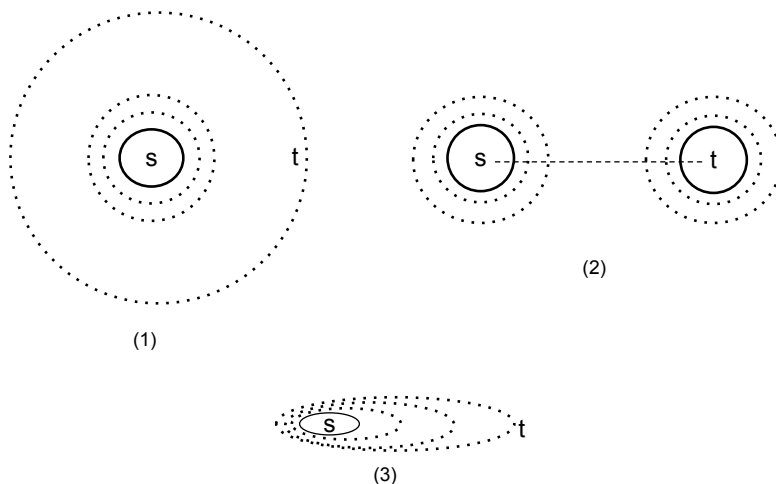


Figure 1: Running Dijkstra's algorithm in three scenarios

As presented in the first two graphs in Figure 1, neither the basic use of Dijkstra's algorithm nor a bidirectional use of it is efficient.

Idea Change edge weights to cause Dijkstra's algorithm to visit fewer vertices (described in the third graph in Figure 1).

Example

Suppose $\pi(v)$ is the shortest path length from v to t , let $\hat{w}(u, v) = w(u, v) - \pi(u) + \pi(v)$. The black numbers in Figure 2 denote the original edge weights of the graph. The red numbers denote the edge weights after reweighting. We can see that the vertices we visited are exactly in order: $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E$.

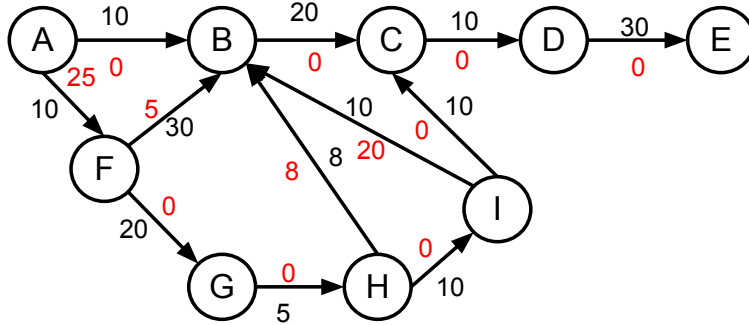


Figure 2: Example of reweighting

In this example, we know exactly the destination t when we reweight the graph, thus Dijkstra’s algorithm using modified edge weights visits exactly vertices on shortest path from s to t . But in reality we do not know what t is when we preprocess the graph.

Solution

- (i) Choose a set L of “landmark” vertices from V .
- (ii) For every $v \in L$, compute the shortest path lengths $\text{dist}(x, v)$ (from x to v) and $\text{dist}(v, x)$ (from v to x) for all $v \in L$.
- (iii) Given the query that to find shortest path from s to t , run Dijkstra’s algorithm using edge weights $\hat{w}(u, v) = w(u, v) - \pi(u) + \pi(v)$ for all $(u, v) \in E$ where $\pi(v) = \max_{x \in L} \{\text{dist}(v, x) - \text{dist}(t, x), \text{dist}(x, t) - \text{dist}(x, v)\}$.

Further information can be found in [1].

Observation

As discussed in the example above, if we are lucky enough to choose the destination t as the landmark, the $\pi(v)$ in step 3 can be simplified to: $\max\{\text{dist}(v, t) - \text{dist}(t, t), \text{dist}(t, t) - \text{dist}(t, v)\} = \text{dist}(v, t)$, which is exactly the $\pi(v)$ we used in the example above.

References

- 1 A. V. Goldberg and C. Harrelson, Computing the Shortest Path: A* Search Meets Graph Theory, Microsoft Research Technical Report, MSR-TR-2004-24, July 2004.