

Routing Algorithms

Unicast routing (or just simply routing) is the process of determining a “good” path or route to send data from the source to the destination. Typically, a good path is one that has the least cost.

Consider a weighted network $G = (V, E, c)$ with positive real-valued edge (link) costs given by the cost function c .

For a source-destination pair $(s, t) \in (V \times V)$, the goal is to find a path $p = (e_1, \dots, e_k)$ from s to t in G that minimizes the cost of the path p , given by $c(p) = \sum_{i=1}^k c(e_i)$.

Shortest path (least-cost) routing.

Two types of routing algorithms: Link state (global) vs. Distance vector (local).

Link State Algorithm

A link state (LS) algorithm knows the global network topology and edge costs.

Each node broadcasts its identity number and costs of its incident edges to all other nodes in the network using a broadcasting algorithm, e.g., flooding.

Each node can then run the (centralized) link state algorithm and compute the same set of shortest paths as any other node. A well-known LS algorithm is the *Dijkstra's algorithm* for computing least-cost paths.

The message complexity and time complexity of the algorithm is determined by the broadcast algorithm.

If broadcast is done by flooding then the message complexity is $O(|E|^2)$.

The time complexity is $O(|E|D)$.

Distance Vector Algorithm (Distributed Bellman-Ford)

Compute the shortest (least-cost) path between s and all other nodes in a given undirected graph $G = (V, E, c)$ with real-valued positive edge weights.

Each node x maintains:

1. a distance label $a(x)$ which is the current known shortest distance from s to x .
2. a variable $p(x)$ which contains the identity of the previous node on the current known shortest path from s to x .

Initially, $a(s) = 0$, $a(x) = \infty$, and $p(x)$ is undefined for all $x \neq s$.

When the algorithm terminates, $a(x) = d(s, x)$, where $d(s, x)$ is the shortest path distance between s and x , and $p(x)$ holds the neighbor of x on the shortest path from x to s .

DBF algorithm

The DBF consists of two basic rules:

Update rule: Suppose x with a label $a(x)$ receives $a(z)$ from a neighbor z .

If $a(z) + c(z, x) < a(x)$, then it updates $a(x)$ to $a(z) + c(z, x)$ and sets $p(x)$ to be z .

Otherwise $a(x)$ and $p(x)$ are not changed.

Send rule: Let $a(x)$ be a new label adopted by x .

Then x sends $a(x)$ to all its neighbors.

The algorithm will work correctly in an asynchronous model also.

Analysis

Theorem 1. *If all nodes work in a synchronous way, then the DBF algorithm terminates after at most n rounds. When it terminates, $a(x) = d(s, x)$ for all nodes x . The message complexity is $O(n|E|)$.*

Proof: Fix a vertex $x \in V$, we prove that the algorithm computes a shortest path from s to x .

Let $P = v_0, v_1, \dots, v_k$, where $v_0 = s$ and $v_k = x$ be a shortest path from s to x . Note that $k < n$.

We prove by induction on i that after the i th round, the algorithm has computed the shortest path from s to v_i , i.e., $a(v_i) = d(s, v_i)$.

The hypothesis holds for $v_0 = s$ in round zero.

Assume that it holds for $j \leq i - 1$. After the i th iteration,

$$a[v_i] \leq a[v_{i-1}] + c(v_{i-1}, v_i) \quad (1)$$

which is the shortest path from s to v_i , since P is a shortest path from s to v_i , and the right hand side is the distance between s to v_i on that path.

Since, in each round $O(|E|)$ messages are exchanged, the total message complexity is $O(n|E|)$.

Computing All-pairs shortest paths

Generalized to compute the shortest path between *all pairs* of nodes, by maintaining in each node x a distance label $a_y(x)$ for every $y \in V$. This is called the **distance vector** of x .

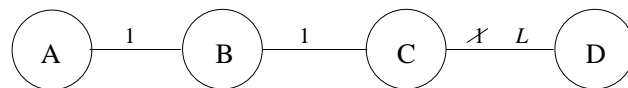
Each node stores its own distance vector and the distance vectors of each of its neighbors.

Whenever something changes, say the weight of any of its incident edges or its distance vector, the node will send its distance vector to all of its neighbors.

The receiving nodes then update their own distance vectors according to the update rule.

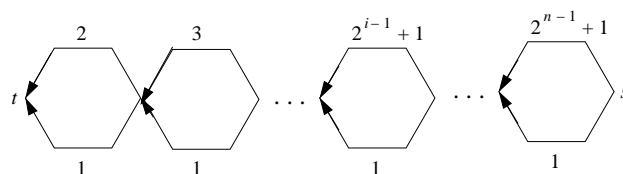
Drawbacks of DBF

A drawback of the DBF algorithm is that convergence time can be made arbitrarily large if the initial distance labels are not correct. “count-to-infinity” problem.



The DBF algorithm can suffer from exponential message complexity in an asynchronous setting.

There are 2^n distinct paths from s to t , each one with a distinct length. It is possible, in an asynchronous environment, to create an execution instance of DBF such that t will receive $\Omega(2^n)$ messages.



An Approximate DBF Algorithm

Awerbuch et al. proposed a simple modification to the DBF algorithm that will result in a polynomial message complexity.

However, the modified algorithm may not compute a shortest path, but will instead compute an *approximate* shortest path, in particular, the paths computed can be worse than the shortest path by *at most* a constant factor.

The only difference between the modified algorithm and the DBF algorithm is in the update rule:

Multiplicative update rule: Let $\alpha = 1 + 1/n$. Suppose x , with a label $a(x)$ receives $a(z)$ from a neighbor z . If $a(z) + c(x, z) < a(x)/\alpha$, then $a(x)$ is updated to $a(z) + c(x, z)$ and $p(x)$ is set to z .

The following theorem gives the performance of the modified DBF.

Theorem 2. *The length of the computed path between s and x (for any node x) at the end of the execution of the algorithm is at most $e \cdot d(s, x)$, where e is the base of the natural logarithm. The number of messages sent is bounded by $O(|E|n \log(n\Delta))$, where Δ is the largest edge cost.*