A walk over the shortest path: Dijkstra's Algorithm viewed as fixed-point computation

Jayadev Misra [1]

Department of Computer Sciences, University of Texas at Austin, Austin, Texas 78712-1188, USA

Abstract

We present a derivation Dijkstra's shortest path algorithm[1]. We view the problem as computation of a "greatest solution" of a set of equations. A UNITY-style computation[0] is then prescribed whose implementation results in Dijkstra's algorithm.

Key words: Design of algorithms, Graph algorithms, Combinatorial problems, Program derivation

0 Introduction

Dijkstra's shortest path algorithm[1] has, by now, become a classic (the cited paper has received such an official designation from the Citation Index Service). Typical descriptions (and derivations) of this algorithm start by postulating that the shortest paths be enumerated in the order of increasing distances from the source. In this note, we present a derivation that is quite different in character. We view the problem as computation of a "greatest solution" of a set of equations. A UNITY-style computation[0] is then prescribed whose implementation results in Dijkstra's algorithm.

The bulk of the work in our derivation is in designing the appropriate heuristics that guarantee termination; this is in contrast to traditional derivations where most of the effort is directed toward postulating and maintaining the appropriate invariant.

[1] E-mail: misra@cs.utexas.edu. Partially supported by the NSF grant CCR–9803842.

1 The Shortest Path Problem

Given is a finite directed graph that has (i) a source node, henceforth, designated by s, and (ii) for each edge (i, j) a non-negative real number, w_{ij}, called its *length*. Length of a path is the sum of edge lengths along the path. It is required to compute the shortest path, i.e., a path of minimum length, from s to every node. Henceforth, "shortest path to a node" means the shortest path from s to that node, and *distance* to a node is the length of the shortest path. The distance to a reachable node from s is a non-negative real and distance to an unreachable node is ∞. For the moment, assume that every node in the graph is reachable from s; the general case, where some of the nodes are unreachable, is taken up in section 1.4. We restrict ourselves to computing the distances to all nodes; shortest paths can be computed using a minor modification of this algorithm.

1.1 Equations, E, for Distances

Let D_k denote the distance to node k; this is a non-negative real number since all nodes are reachable from s. Note that $D_s = 0$. Call i a *predecessor* of k if there is an edge (i, k) in the graph. For a node k, $k \neq s$, whose only predecessors are i, j,

$D_k = \min(D_i + w_{ik}, D_j + w_{jk})$.

This is because the shortest path to k passes through either i or j, and any initial segment of a shortest path is a shortest path to the corresponding node. Therefore, D – where D is a vector, with the nodes ordered in some fixed manner – is the unique solution for the unknowns d in the following equations, E.

E:: $d_s = 0$,
$\langle \forall j : j \neq s : d_j = (\min\ i : i \text{ is a predecessor of } j : d_i + w_{ij}) \rangle$.

The minimum over an empty set is taken to be ∞. Hence, for a node j without predecessor, $j \neq s$, $d_j = \infty$. Then, node j is unreachable from s, a case discussed in section 1.4.

1.2 Inequalities, F, for Distances; Relaxing the Equations E

An equation of the form $d_k = \min(d_i + w_{ik}, d_j + w_{jk})$ implies that $d_k \leq d_i + w_{ik}$, and $d_k \leq d_j + w_{jk}$. We convert E into a set of such inequalities, one for each edge. Let F be the system of inequalities so constructed along with the

equation $d_s = 0$.

$$\begin{array}{ll} \text{F::} & d_s = 0, \\ & \langle \forall \text{ edge}(i,j) :: d_j \leq d_i + w_{ij} \rangle. \end{array}$$

A solution of F is a vector. (Henceforth, "vector" refers to a vector of the appropriate length all of whose elements are non-negative reals or ∞.) It is clear that any solution of E – the only solution of E is D – is a solution of F. However, F may have many more solutions; for instance, a vector of all zeroes is a solution of F.

1.3 Distance Vector is the Greatest Solution of F

Define a partial order $\sqsubseteq$ over vectors as follows. For vectors u, v,

$$u \sqsubseteq v \equiv (\forall i :: u_i \leq v_i).$$

1.3.0.1 Theorem GS The distance vector, D, is the greatest solution of F, i.e., D is a solution of F and for any solution d, $d \sqsubseteq D$.

Proof: As stated earlier, D is a solution of F. We prove that $d \sqsubseteq D$, for any solution d of F. Let h_j be the number of edges in the shortest path to node j; if there are multiple shortest paths to j then one with the fewest edges determines h_j. Since every node is reachable from s, h_j is defined for all j. First, we prove the following proposition, by induction on natural numbers n.

$$\begin{array}{ll} \text{H::} & (\forall n :: \\ & \quad (\forall j :: h_j = n \Rightarrow d_j \leq D_j) \\ &) \end{array}$$

$n = 0$: We have to show $(\forall j :: h_j = 0 \Rightarrow d_j \leq D_j)$. From the definition of h, $h_j = 0 \Rightarrow j = s$, and $j = s \Rightarrow d_j \leq D_j$, because $d_s = 0$ (from F) and $D_s = 0$.

$n + 1$: We show that $h_j = (n+1) \Rightarrow d_j \leq D_j$. From equations E and the definition of h, $D_j = D_i + w_{ij}$ and $h_j = h_i + 1$, for some i. Therefore, $h_i = n$.

$$\begin{array}{lll} & d_j & \\ \leq & \{d \text{ is a solution of F; hence, } d_j \leq d_i + w_{ij}\} & \\ & d_i + w_{ij} & \\ \leq & \{\text{applying induction: } h_i = n \Rightarrow d_i \leq D_i\} & \\ & D_i + w_{ij} & \\ = & \{D_j = D_i + w_{ij}\} & \end{array}$$

D_j

For every node j there is some n such that $h_j = n$. Hence, for every j, $d_j \leq D_j$, from H.

1.4 Unreachable nodes

Our treatment, so far, has assumed that all nodes are reachable from s. Under that condition there is a unique solution to E, which is the distance to the nodes. If there are unreachable nodes, there will be multiple solutions to E: for instance, let u, v be two nodes, different from s, that are each other's predecessor, they have no other predecessors, and the lengths of the two edges, (u, v) and (v, u), are both zero. Then E yields the equations, $d_u = d_v$ and $d_v = d_u$, permitting these variables to be set arbitrarily.

It can be shown that D is still the greatest solution of E in that case. Theorem GS is still valid. The proof of H $\Rightarrow$ $(\forall j :: d_j \leq D_j)$ (in the proof of GS) will have a case distinction for reachable and unreachable nodes.

2 Computing the Distances

We first suggest a naive method for computing the greatest solution of F; this method is based on the execution rule from UNITY[0]. The inequality corresponding to edge (i, j) is $d_j \leq d_i + w_{ij}$. This inequality is equivalent to the equation $d_j = \min(d_j, d_i + w_{ij})$. Convert this equation to the following assignment statement, S_{ij}, for edge (i, j):

S_{ij}:: $d_j := \min(d_j, d_i + w_{ij})$.

Note that the only effect of executing S_{ij} is to, possibly, decrease d_j.

The execution strategy is to start in a state where $d_s = 0$ and $d_j = \infty$ for all j, $j \neq s$. Then execute an arbitrary statement in each step, ensuring that every statement is executed eventually. We show, below, that eventually the distances are computed, i.e., $d = D$.

2.1 Fixed Point, Invariant

A *fixed point* is a state where no statement execution has any effect. Execution of $d_j := \min(d_j, d_i + w_{ij})$ has no effect if and only if its left and right sides are

equal in value, i.e., $d_j = \min(d_j, d_i + w_{ij})$, or $d_j \leq d_i + w_{ij}$. Therefore, a fixed point is a solution of F. Using Theorem GS, $d \sqsubseteq D$ at any fixed point.

Now, we show that $D \sqsubseteq d$ is an invariant of the proposed execution. Coupled with $d \sqsubseteq D$ at a fixed point, we have $d = D$ at any fixed point reached by this execution (recall that $\sqsubseteq$ is a partial order). Below, we prove that $D \sqsubseteq d$ is an invariant; in the next section we address the question of reaching a fixed point.

Initially, $D \sqsubseteq d$ holds because, initially, $d_s = 0$ and $d_j = \infty$ for all j, $j \neq s$. We show that execution of a statement, S_{ij}, preserves $D \sqsubseteq d$. The execution of S_{ij} affects only d_j; therefore, it is sufficient to show that $D_j \leq d_j$ is a postcondition of S_{ij} given that the invariant is a precondition. That is,

$$\{D \sqsubseteq d\}\ d_j := \min(d_j, d_i + w_{ij})\ \{D_j \leq d_j\}.$$

Applying the axiom of assignment, we have to show that $(D \sqsubseteq d) \Rightarrow (D_j \leq \min(d_j, d_i + w_{ij}))$.

$$\begin{array}{ll} & D_j \\ = & \{\text{From } D \sqsubseteq d\text{: } D_j \leq d_j\} \\ & \min(d_j, D_j) \\ \leq & \{D \text{ is a solution of F; hence } D_j \leq D_i + w_{ij}\} \\ & \min(d_j, D_i + w_{ij}) \\ \leq & \{\text{From } D \sqsubseteq d\text{: } D_i \leq d_i\} \\ & \min(d_j, d_i + w_{ij}) \end{array}$$

2.2 Reaching A Fixed Point

In section 2.1 we showed that $d = D$ at any fixed point reached by the given computation strategy. It can be shown that picking an arbitrary statement to execute, as long as every statement is executed eventually, reaches a fixed point in a finite number of steps. However, this strategy is wasteful because it may consecutively repeat execution of a statement even though such executions have no effect.

Define the *measure* of statement S_{ij} to be d_i. Call a statement *active* if its measure has changed since its last execution; the statement is *idle* otherwise. More formally, initially all statements are active. A statement becomes idle by being executed; an idle statement becomes active only if its measure changes. Therefore, idle statement S_{jk} could become active, i.e., d_j could change, only as a result of executing some S_{ij}.

It follows that: (1) execution of an idle statement does not change the system

state, (2) therefore, if all statements are idle then the system state is a fixed point, and (3) execution of S_{ij} can make an idle statement of the form S_{jk} active provided $d_j > d_i + w_{ij}$ prior to the execution, because the measure of S_{jk}, d_j, changes only under this condition.

We propose that only *active statements be picked for execution.* Such a computation reaches a fixed point. We propose below a refinement of this strategy and prove its correctness.

2.3 Refinement of the Execution Strategy: BF-Strategy

Dijkstra's algorithm is the implementation of the following, breadth-first, strategy. We show that this strategy reaches a fixed point.

2.3.0.2 BF-Strategy Pick an active statement of the smallest measure (among all active statements) for execution.

2.3.0.3 Observation: BF-Strategy has the following properties.

(1) An idle statement remains idle.
(2) The following proposition, C, is invariant:
C:: measure of any idle statement $\leq$ measure of any active statement.

Proof: Proposition C holds initially because there is no idle statement.

Let S_{ij} be an active statement of the smallest measure, chosen for execution in a step. First, we show that all idle statements remain idle. We need consider only idle statements of the form S_{jk} because execution of S_{ij} can only change d_j, and thus, possibly, make S_{jk} active. Prior to S_{ij}'s execution the measure of S_{jk}, d_j, is at most the measure of S_{ij}, d_i, from the invariant C. Therefore, the execution of S_{ij},

$$d_j := \min(d_j, d_i + w_{ij})$$

does not change d_j, leaving S_{jk} idle.

Now, we prove that C holds after the execution of S_{ij}. Before the execution of S_{ij} the measure of any idle statement $\leq d_i$ (from C, and that S_{ij} was active). The execution of S_{ij} does not change the measure of any idle statement (see paragraph above) and it makes S_{ij} idle. Therefore, after the execution of S_{ij} the highest measure for any idle statement is d_i. The lowest measure for any active statement, before execution of S_{ij}, was d_i. Execution of S_{ij} may change

the measure d_j for an active statement of the form S_{jk} to $d_i + w_{ij}$. Hence, every active statement's measure $\geq d_i$, thus preserving C.

Since each step increases the number of idle statements (the active statement chosen for execution becomes idle), a fixed point is reached eventually.

2.4 Implementation of the BF-Strategy

We show that the BF-strategy can be implemented in $O(n^2)$ time, where n is the number of nodes.

Let S_{ij} be an active statement of the smallest measure. Then, from the definition of measure, any active statement S_{ik} also has the smallest measure, because both these measures are equal to d_i. Further, execution of S_{ij} leaves an active S_{ik} active with the smallest measure: execution of S_{ij} can, possibly, change d_j to $d_i + w_{ij}$, which is at least d_i, the measure of S_{ik}. Therefore, we propose that once an active statement of the smallest measure, S_{ij}, is identified then S_{ik}, for all k, be executed (if S_{ik} is idle its execution has no effect).

Then the implementation strategy is: (1) find i such that S_{ij} is an active statement of the smallest measure, (2) execute S_{ik}, for all k.

Call (1,2) above a *superstep* with node i. Such a superstep makes S_{ik}, for all k, idle and they remain idle forever. Call node i idle if S_{ik}, for all k, are idle; i is active otherwise. The proposed implementation strategy guarantees that if S_{ij} is chosen in a superstep then node i is active, and following the superstep node i is permanently idle.

A superstep may be implemented in $O(n)$ time. Associate a label, idle or active, with each node; initially all nodes are active. Scan the list of d-values to locate an active node i such that d_i is lowest among all active nodes; this is an $O(n)$ computation. Then, execute S_{ik}, for all k, and mark i idle; this is again an $O(n)$ computation. Since i remains idle afterwards, there are exactly n supersteps before all nodes (and statements) become idle. Hence, the entire algorithm is implemented in $O(n^2)$ time.

Acknowledgement It is with pleasure that I acknowledge the critical comments of Edsger W. Dijkstra, which not only improved the overall presentation but simplified the proof of the main theorem. I am indebted to Rajeev Joshi for several insightful comments, including the observation that the solution to E may not be unique in the presence of unreachable nodes. Perceptive comments from Michel Charpentier, Beverly Sanders and two anonymous referees have helped improve the presentation considerably.

References

[0] K. Mani Chandy and Jayadev Misra. *Parallel Program Design: A Foundation*. Addison Wesley, 1988.

[1] E. W. Dijkstra. A note on two problems in connection with graphs. *Numerische Mathematik*, 1:83–89, 1959.