On the Design and Verification of a Simple Distributed Spanning Tree Algorithm[1]

Manfred Broy
Institut für Informatik
Technische Universität München
Postfach 20 24 20, D-8000 München 2

Abstract

The design of a distributed algorithm for computing a minimal distance spanning tree is carried out as a case study for the systematic derivation of a distributed algorithm in a functional setting. A distributed algorithm is derived and proved correct.

1. Introduction

In designing algorithms for the solution of informally given problems the major steps consist in the adequate formalisation of the problem, the design of an algorithmic solution, its verification and optimization. It has been proved useful not to do all these steps isolated, but structuring and connecting them in some adequate way. Such a proceeding has been widely recognized as possible and demonstrated useful for numerous cases of sequential programs. Maybe, it is less widely recognized that such a proceeding also works for distributed algorithms.

[1]This work was supported by the Sonderforschungsbereich 342 Werkzeuge und Methoden für die Nutzung paralleler Architekturen

In the following we use transformational and functional techniques for developing a simple distributed algorithm for computing a spanning tree in a directed graph for a given root where every path in the tree represents a connection between the root and the resp. node with minimal distance. Of course there are many ways to compute minimal distance spanning trees. Since we want to compute the tree in a distributed way we introduce a set of messages. The spanning tree is computed by associating a program with every node in the graph that communicates with the programs at its neighbour nodes by sending messages and in this way eventually computing a spanning tree.

In the following we first give a formal specification of the problem. Then we construct a solution and verify it. We put special emphasis on the functional style of describing distributed algorithms.

2. The Problem Specification: Spanning Trees with Minimal Distance

We start with a problem specification independent of the question of distributed or sequential algorithms. We specify the concept of a minimal distance spanning tree.

Let a finite set D of nodes be given. A directed graph is specified by the function

$$n: D \to \wp(D)$$

that gives for every node $d \in D$ its set of neighbours n.d.

For specifying the algorithm we use the algebraic structure of sequences. Given a set M we denote by M* the set of finite sequences of elements. On sequences we use the following functions

- ‹› denotes the empty sequence,
- ‹m› denotes for $m \in M$ the one element sequence consisting just of m,
- x^y denotes the sequence being the result of the concatenation of sequences x and y,
- ft.x denotes the *first* (left-most) element of the sequence x (undefined, if x is empty),
- rt.x denotes the *rest* of the sequence, i.e. the sequence without the first element (rt.x is empty for the empty sequence x),

lt.x denotes the *last* element of the sequence x (undefined, if x is empty),

ld.x denotes *lead* of the sequence x, i.e. the sequence without the last element (empty, if x is empty),

#x denotes the length of the sequence x,

S©x denotes for $S \subseteq M$ and sequence x the subsequence of x consisting just of the elements from S.

A *path* from node a to node b in the graph is a nonempty sequence $p \in D^*$ where ft.p = a and lt.p = b and successive nodes are connected by arcs. Formally a sequence p is a path, if it satisfies (for all $x, y \in D^*$, $d, e \in D$) the following formula:

$$x\hat{}\langle d\rangle\hat{}\langle e\rangle\hat{}y = p \Rightarrow e \in n.d\ .$$

Given a graph by the neighbourhood function n a *minimal distance spanning tree* with root $\in D$ defines a tree consisting of paths of minimal length from the root to the nodes. It can be represented by a pair of functions

$$v : D \to D,$$

$$w : D \to \mathbb{N} \cup \{\infty\},$$

where v defines for every node d its predecessor in the tree and w defines the length of the path (of minimal length) from the root and leading over v.d to d. Such a pair of functions represents a minimal spanning tree, if the following property is valid:

w.root = 1 and v.root = root; moreover for every node $d \in D\backslash\{root\}$ the fact $w.d \in \mathbb{N}$ implies that there is a path p of minimal length from root to v.d such that $w.d = 1+\#p$ and $d \in n.v.d$.

In other words the pair of functions v and w forms a minimal spanning tree, if for all nodes $d \in D$ with path from root to d we have:

$$\langle v^{w.d}.d,\ v^{w(d-1)}.d,\ \ldots\ v^{0}.d\rangle$$

is a path of minimal length from the root, i.e.we assume root = $v^{w.d}.d$, to d where v^i is defined as follows:

$$v^{0}.d = d,$$

$$v^{i+1}.d = v(v^{i}.d).$$

In the following we derive an algorithm for computing such a minimal spanning tree in a distributed way.

3. Computing Spanning Trees by Message Passing

The spanning tree is to be computed in a distributed manner. This is done by attaching to every node in the graph a communicating program that exchanges messages with the programs at its neighbour nodes. Every program in node d maintains two variables containing its father and the length of the minimal path found so far. In particular the state of each program is uniquely determined by these program variables. Initially the values of these variables are arbitrary and ∞ resp. By exchanging messages the programs are computing better and better approximations for a predecessor ("father") node and a path length that eventually define a spanning tree in that way.

We consider as messages elements of the set A of triples, where the triples have the form

act(src : D, des : D, lgt : $\mathbb{N}$).

Accordingly a message is a triple that can be selected by the selector functions src ("*source*"), des ("*destination*"), lgt ("*length*"). By a message a node src that has got information (by some message from some neighbour node) about a better approximation for a shortest path sends all its neighbours des the information of the existence of a path with some length lgt.

Accordingly the message act(b, c, j) is supposed to contain the information (send from node b to its neighbour node c i.e. there is an arc from b to c) that there is a path in the graph from the root root to b and further on to node c of length j. Accordingly a message a is called *valid*, if there actually exists a path p of length lgt.a in the given graph which has the form

root $\rightarrow$ ... $\rightarrow$ src.a $\rightarrow$ des.a

i.e. the following conditions are fulfilled for the path p:

ft.p = root,
#p = lgt.a,
lt.p = des.a,
#p > 1 $\Rightarrow$ src.a = lt.ld.p,
#p = 1 $\Rightarrow$ src.a = root .

A set $S \subseteq A$ of messages is called *fully valid*, if for S the following conditions are fulfilled:

- every message $a \in S$ is valid,
- if message b = act(i, j, w) is valid, then there exists a message a = act(i', j, w') $\in$ S such that a is valid and $w' \leq w$.

From a fully valid set of messages a minimal spanning tree can be obtained immediately. This is formulated by the following theorem.

Theorem: Let S be a set of messages that is fully valid and the functions

$$v: D \to D,$$

$$w: D \to \mathbb{N},$$

be given such that for all $d \in D$

$$w.d = \min \{lgt.a: a \in S \wedge des.a = d\},$$

$$\exists\, a \in S: lgt.a = w.d \wedge des.a = d \wedge src.a = v.d,$$

then (v, w) is a minimal spanning tree.

Proof: Obvious from the definition of a minimal spanning tree. ◊

In the following section we attach the nodes in the graph communicating entities that exchange valid messages until finally a fully valid finite set of messages has been exchanged.

4. The Communicating Entities Attached to the Nodes

In communicating systems we sometimes have to deal with infinite sequences of messages. An infinite sequence of elements from a set M can be understood as a function $\mathbb{N} \to M$. The set of infinite messages is denoted by M^{∞}.

A *stream* is a finite or infinite sequence of messages. The set of streams over M is denoted by

$$M^{\omega} =_{df} M^{*} \cup M^{\infty}.$$

Concatenation carries over to streams in a straightforward way. We just have to observe that for $x \in M^{\infty}$, $y \in M^{\omega}$: $x^{\wedge}y = x$ and lt.x is undefined. All other

functions on finite sequences carry over to streams in a straightforward way besides lt (which is undefined on infinite streams) and ld (which is the identity on infinite streams).

On streams we define the prefix ordering $\sqsubseteq$ by

$$x \sqsubseteq y \quad \text{iff} \quad \exists\, z \in M^{\omega}: x \hat{} z = y\ .$$

Equipped by $\sqsubseteq$ the set M^{ω} forms a partially ordered set where directed sets have least upper bounds. A *stream processing function* is a prefix continuous function on streams.

We specify in the following a communicating program for every node of the given graph. Semantically such a program is represented by a stream processing function. The programs receive messages from its neighbours and sometimes sends messages in response to its neighbours.

Whether a message is sent as a response is determined by the state of the programs. In the computation of a minimal spanning tree a state of the programs is described (apart from its location at a certain node) by a pair of values from

$$D \times (\mathbb{N} \cup \{\infty\}).$$

The first element of the pair is called the *current father* and the second value is called the *current length*. In every state of the computation every program at a node has this way some approximation of a spanning tree consisting of a father node and the length of a path to root. The program at a node d starts its computation with the assumption that its father is root and the distance is ∞.

4.1 Echos to Communications

If the program at node i with current path of length w gets a message a (i.e. we have des.a = i), it sends a finite sequence echo(a, w) of messages containing one message for each of its neighbours. We specify the function

$$\text{echo} : A \times \mathbb{N} \to A^{*}$$

as follows:

$$\#(\{act(i, j, w')\} \copyright \text{echo}(a, w)) = \begin{cases} 1 & \text{if } lgt.a < w \wedge i = des.a \wedge j \in n.i \wedge w' = 1 + lgt.a \\ 0 & \text{otherwise} \end{cases}$$

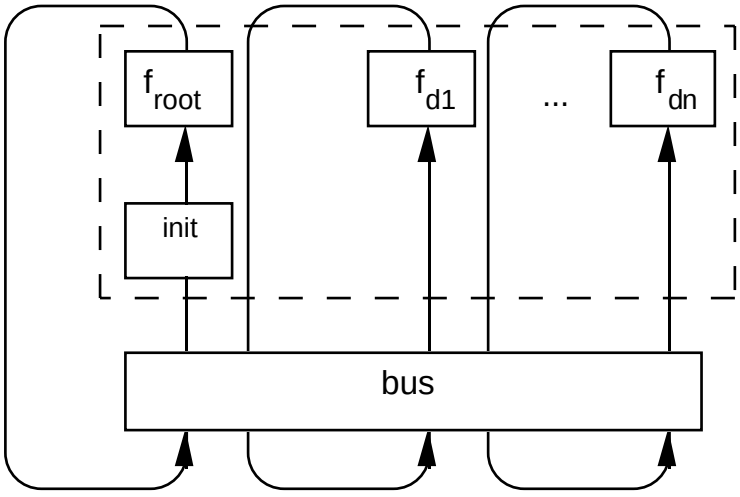
The axiom specifies that the node does not send any messages in reaction to a message that does not improve of its current length and it sends exactly one message to each of its neighbours as echo to a message that improves its current length.

Note that every message a contains the information about node des.a to which it was sent. Note, moreover, that if the message a is valid all messages in echo(a, w) are valid, too.

4.2 A Network of Communicating Programs

With every node d we associate a stream processing function f.d that models the behaviour of the program associated with this node. The programs communicate over a "bus" which again is characterized by a stream processing function with |D| input streams and |D| output streams. The bus takes the output streams of the programs at the individual nodes as input and produces for each node a stream of messages according to the destinations mentioned in the messages.

This way we design a network of programs of the form



that computes the spanning tree.

For every program at a node we may ask for its input streams (and its output streams) in a computation. A *valuation*

$$e: D \to A^{\omega}$$

associates with every node a stream of messages. We use valuations to represent for every node its input stream (or output stream resp.).

4.3 The Distributer

A *distributer* for the message stream produced by the programs at the nodes of the graph is given by a function bus mapping valuations to valuations. Accordingly the function bus has the functionality

$$\text{bus}: (D \to A^{\omega}) \to (D \to A^{\omega}) .$$

A distributer is called *safe*, if messages are delivered to the correct address and only messages are delivered that have been sent, i.e. for all valuations e: $D \to A^{\omega}$ we have for all messages a and all nodes j:

$$\#\{a\}©(\text{bus}.e).j \geq 0 \;\Rightarrow\; \text{des}.a = j,$$

$$\sum_{\substack{d \in D \\ \text{des}.m = j}} \#(\{m\}©e.d) \;\geq\; \#(\{a\}©(\text{bus}.e).j) \cdot$$

In addition the function bus is called *live* for a given valuation e, if all messages finally are delivered, i.e. if

$$\sum_{d \in D} \#e.d = \sum_{d \in D} \#(\text{bus}.e).d.$$

We write L(bus, e), if the function bus is safe and it is live for the valuation e.

4.4 Functional Specification of the Network Behaviour

Next we specify the properties of the stream processing functions associated with the programs at the nodes. The function

$$h: D \times (\mathbb{N} \cup \{\infty\}) \to (A^{\omega} \to A^{\omega})$$

associates with every pair (v, w), representing the state of the program, a stream processing function by the axioms:

$$h(v, w).\langle\rangle = \langle\rangle,$$

$$h(v, w).(\langle a\rangle \hat{} x) = \text{echo}(a, w) \hat{}\ \textbf{if}\ \text{lgt}.a < w \ \textbf{then}\ h(\text{src}.a, \text{lgt}.a).x \ \textbf{else}\ h(v, w).x \ \textbf{fi}$$

It is a straightforward proof that the path length contained in messages of successive nonempty echos is strictly decreasing:

$$h(v, w).x = y\hat{}\alpha\hat{}\beta\hat{}z \wedge \alpha = echo(a, u) \neq \langle\rangle \wedge \beta = echo(b, u') \neq \langle\rangle \Rightarrow u > u',$$

i.e. in every nonempty echo the length mentioned in the sent messages decreases strictly. Accordingly the programs associated with the nodes are all equal and just given by

$$h(root, \infty).$$

The function

$$g: (D \to A^{\omega}) \to (D \to A^{\omega})$$

is defined by the parallel composition of all programs represented by the stream-processing functions in the nodes $d \in D\backslash\{root\}$ for every $s \in (D \to A^{\omega})$:

$$(g.s).d = h(root,\infty).(s.d)$$

The stream processing function

$$init: A^{\omega} \to A^{\omega}$$

is defined by

$$init.x = \langle act(root, root, 1)\rangle\hat{}x.$$

It is used to produce the message that informs the node root that it forms the root and this way initiates the computation. We specify

$$(g.s).root = h(root, \infty).init.s.root\ .$$

Basically g is the parallel composition of all the transducer functions in the particular nodes.

The system of streams of messages produced by the programs in the nodes is defined by the valuation obtained as least fixpoint of the recursive definition:

$$s = g.bus.s \quad \textbf{where} \quad L(bus, s).$$

Here s denotes a valuation that associates with every node the output produced by the programs if composed in the described feedback loop.

4.5 Verification

The set of messages contained in the streams of the valuation s defined above defines a fully valid set, since first of all echos for valid messages consist of valid

messages and the initial message is valid. By induction on the length of a path we prove that if there is a path of length n from root to j, then there is a message a in s with des.a = j and lgt.a ≤ n. This demonstrates that the set is fully valid and thus the correctness of the algorithm.

Termination (finiteness of all the streams in s) can be seen by the following argument. If node j receives a message a with lgt.a = n, then it sends out as echos at most a finite sequence of messages b with lgt.b = n+1. Afterwards it sends messages at most as echos to messages with a length less than n. This proves that every node sends only a finite number of messages (note that every message is produced at most once).

5. Concluding Remarks

In this little case study I very deliberately did not give a completely formal treatment for the proof, but tried to give just enough formalism to keep the derivation and proofs logically complete, but still readable and understandable. It should be clear that the given informal proofs can be transformed into fully formal proofs.

Acknowledgement

The problem of a distributed algorithm for the minimal distance spanning tree was brought to my attention by Leslie Lamport during my stay at DEC SRC in September 1989 in a number of pleasant discussions. I am grateful to Thomas Gritzner for careful reading the manuscript.

References

[Welch et al. 88]
J. Welch, L. Lamport, N. Lynch: A lattice-structured proof technique applied to a minimal spanning tree algorithm. Proc. 7th ACM Symp. on Principles of Programming Languages 1988