June, 1981 LIDS-P-1102

# A DISTRIBUTED SHORTEST PATH PROTOCOL

Francine B.M. Zerbib and Adrian Segall

Departments of Computer Science and Electrical Engineering
Technion, Israel Institute of Technology

## ABSTRACT

We present a distributed protocol for obtaining the shortest paths between all pairs of nodes in a network with weighted links. The protocol is based on an extension to the Dijkstra (centralized) shortest path algorithm and uses collaboration between neighboring nodes to transfer the information needed at the nodes for the successive construction of the shortest paths. A formal description of the protocol is given by indicating the exact algorithm performed by each node. The validation proofs are greatly simplified by separating the communication mechanism from the computation at the nodes, the latter being the transposition of the Dijkstra shortest path algorithm to the decentralized protocol.

The work of A. Segall was performed on a consulting agreement with the Laboratory for Information and Decision Systems at MIT, Cambridge, Mass. and was supported in part by the Advanced Research Project Agency of the US Department of Defense (monitored by ONR) under Contract N00014-75-C-1183 and in part by the Office of Naval Research under Contract ONR/N00014-77-C-0532 (NR 041-519).

This paper has been submitted to the IEEE Trans. on Communications.

## 1. Introduction

This paper presents a distributed protocol for obtaining shortest paths and distances between nodes in a network. The nodes are assumed to possess a certain memory and computation capability and to be able to collaborate via control messages exchanged between neighbors. Each node builds its tree of shortest paths to all other nodes in the network and, while proceeding with its own algorithm, also helps the other nodes to advance their algorithm.

Each node is assumed to start its algorithm with knowledge of the weights of the adjacent outgoing links and of the identities of the nodes that may potentially be in the network. When the algorithm is completed at a node, it knows which nodes are indeed reachable and the shortest path and distance to each.

The distributed protocol here is based on the Dijkstra algorithm [1], [2] for obtaining shortest paths in a centralized way. An early version of the present distributed protocol was proposed by R.G. Gallager [3] and analysed by D. Friedman [4]. The present version adds features that produce savings in communication and protocol duration as explained in Section 6. In addition, we present a complete description of the algorithm that must be performed by the nodes to participate in the distributed protocol and a rigorous validation of its performance.

The validation process is based on examination of the decentralized protocol vs. the centralized algorithm, where in the first one we distinguish the communication process from the computation part. The first one deals with the construction of a communication mechanism whose purpose is to enable a node to obtain information that initially resides at other nodes. This mechanism is also designed in such a way that nodes screen and summarize the information prior to its transmission to a neighbor. Once the information is correctly transmitted, the computation part is able to construct shortest paths as in the centralized algorithm. We show that, provided that the centralized algorithm is already known and proved (as in the case of the Dijkstra algorithm), such a separation reduces the validation of the distributed protocol to the proof of correctness of the communication mechanism.

The paper is organized as follows : in Section 2 we present several notations and definitions that are used in the rest of the paper, while Section 3 summarizes the Centralized Dijkstra Algorithm (CDA) and its main properties. An extended version of the CDA, introduced and proved in Section 4 leads to the Distributed Dijkstra Protocol (DDP) which is presented in Section 5. Its validation is given partly in the same section and partly deferred to the Appendix. Finally, Section 6 contains several conclusions, calculations of communication complexity and comparisons with previous works.

## 2. Basic notations and definitions

Let $G(V,E)$ be a graph, where $V$ is a set of nodes and $E$ a set of links. The nodes in $V$ are numbered $1,2,\ldots,|V|$, and are referred to by their number. We assume that each link is bidirectional and associate to each direction on a link from $i$ to $j$ a strictly positive weight $d_{ij}$, where the weights of opposite directions may be different. For convenience, we take $d_{ii} = 0$ and if there is no link from $i$ to $j$ we take $d_{ij} = \infty$. A path is a sequence of distinct nodes $\{i_o,i_1,\ldots,i_m\}$ such that there is a link connecting $i_k$ and $i_{k+1}$. Given a path $P$, we define $DIST(P)$ as the sum of the weights along the path. For the purpose of the algorithms of this paper, it is convenient to define a total order $\prec$ on all paths originating at a given node $i$, by using the following recursive definition :

Definition 2.1

We say that two paths $P_1$, $P_2$ that originate at a node $i$ are such that DIST $(P_1) \prec$ DIST $IP_2)$ if one of the following holds :

a) DIST $(P_1) <$ DIST $(P_2)$

b) DIST $(P_1) =$ DIST $(P_2)$ and $k_1<k_2$ where $k_1,k_2$ are the end nodes of $P_1,P_2$ respectively.

c) DIST $(P_1) =$ DIST $(P_2)$ and $k_1=k_2$ and DIST $(P_1') \prec$ DIST $(P_2')$, where $P_1',P_2'$ are subpaths of $P_1,P_2$ originating at $i$ and terminating at the nodes $k_1'$, $k_2'$ preceding $k_1=k_2$ on each of the paths.

We say that $P_1$ is shorter than $P_2$ if DIST $(P_1) \prec$ DIST $(P_2)$. For any two nonidentical paths $P_1$, $P_2$ originating at a node $i$, either $P_1$ is shorter than $P_2$ or $P_2$ is shorter than $P_1$. Also, with this definition, there is a unique path connecting two given nodes $i$ and $k$ that is shorter than all other paths connecting $i$ and $k$, and this will be called the shortest path. In addition, this definition ensures that if $j$ is a node on the shortest path $P$ from node $i$ to node $k$, then the shortest path from $i$ to $j$ and the shortest path from $j$ to $k$ are both subpaths of $P$. This last property is of importance in the distributed protocol and its validation.

In this paper, an array will be denoted by a capital letter, possibly with a subscript indicating the node where the array is located. For example $N_i$ is the node table at node $i$. The notation $N_i = (N{\cdot}s_i, N{\cdot}d_i, N{\cdot}p_i)$ means that the columns of $N_i$ are $N{\cdot}s_i$, $N{\cdot}d_i$, $N{\cdot}p_i$ and $N_i(x)$ denotes the row $x$ in $N_i$. Also $N{\cdot}s_i(x)$ is the entry in the row $x$ of $N{\cdot}s_i$. Therefore $N_i(x) \leftarrow (0,\infty, \text{nil})$ means $N{\cdot}s_i(x) \leftarrow 0$, $N{\cdot}d_i(x) \leftarrow \infty$, $N{\cdot}p_i(x) \leftarrow \text{nil}$ and $N{\cdot}d_i \leftarrow \infty$ means $N{\cdot}d_i(y) \leftarrow \infty$ for all $y$.

In each of the algorithms presented in this paper, a node $i$ will hold variables $N{\cdot}d_i(k)$, $N{\cdot}p_i(k)$ for each node $k$ that indicate respectively DIST(P) where $P$ is a certain path from $i$ to $k$ and the predecessor of $k$ on $P$. Similarly to Definition 2.1, we use :

Definition 2.2

We say that $N{\cdot}d_i(k_1) \prec N{\cdot}d_i(k_2)$, where $k_1 \neq k_2$, if one of the following holds :

a) $N{\cdot}d_i(k_1) < N{\cdot}d_i(k_2)$

b) $N{\cdot}d_i(k_1) = N{\cdot}d_i(k_2)$ and $k_1 < k_2$

Also, if $j$ is a neighbor of $k$ such that $j \neq N{\cdot}p_i(k)$, we say that $N{\cdot}d_i(j) + d_{jk} \prec N{\cdot}d_i(k)$ if one of the relations below holds :

c) $N{\cdot}d_i(j) + d_{jk} < N{\cdot}d_i(k)$

d) $N{\cdot}d_i(j) + d_{jk} = N{\cdot}d_i(k)$ and $j < N{\cdot}p_i(k)$ .

We define the relation $\succ$ in a similar manner.

Throughout the paper, all comparisons will be made according to the $\prec$ relation . For example, a node that achieves $\min_k N{\cdot}d_i(k)$ is the unique node $k^*$ for which $N{\cdot}d_i(k^*) \prec N{\cdot}d_i(k)$ for all $k$. Other notations are :

| | |
|---|---|
| $S_i$ | = set of neighbors of node i |
| $\Lambda$ | = $(\Lambda\cdot n, \Lambda\cdot d)$ adjacency array of some node $\mu$, where $\Lambda\cdot n \subseteq S_\mu$ and $\Lambda\cdot d(x) = d_{\mu,\Lambda\cdot n_\mu}(x)$ |
| MMP(i,k) | = shortest path from i to k (in the sense of Definition 2.1) |
| MMD(i,k) | = DIST (MMP(i,k)) |
| $f_i(k)$ | = first node after i on MMP(i,k) |
| MP(cond, i,k) | = shortest path from i to k under condition cond. |
| MD(cond, i,k) | = DIST (MP(cond, i,k)) |

<u>on U</u> : let $U \subseteq V$ and $i_o \in U$; then a path $\{i_o, i_1, \ldots, i_{m-1}, i_m\}$ is on U if $i_\ell \in U$ for $\ell = 0,1,\ldots,m-1$ (but not necessarily for $\ell = m$),

$R_{ik} = \{x | x \in S_k$ and k is the predecessor of x on MMP(i,x)$\}$ is called the set of sons of k for i. Note that Definition 2.1 ensures that for a given i, every node is the son of exactly one node.

When necessary, we indicate the value of a variable at a given time t by writing t in parentheses following the name of the variable.

The sequence of actions performed by the processor at a node as a result of receiving a message is assumed to be executed without interruption and is referred to as an <u>event</u>. Consequently we may assume that an event takes zero time and that no two events occur at the same time. In addition, in the distributed protocol, messages sent by a node to a neighbor are assumed to arrive correctly and in order within arbitrary nonzero finite time.

## 3. The Centralized Dijkstra Algorithm (CDA)

The Dijkstra algorithm starts with knowledge of the topology of the graph and the weights of the links, and computes shortest distances and paths from a given node $i$ to all the other nodes in the network. The algorithm divides the nodes in three categories : $P_i$ - set of "permanent" nodes, $T_i$ - set of "tentative" nodes and the rest forms the set of "unknown" nodes. The tentative nodes are the neighbors of permanent nodes that are not permanent themselves. At any given instant the algorithm knows the shortest path and distance from $i$ to all permanent nodes $x \varepsilon P_i$ and also the shortest path and distance <u>on $P_i$</u> from node $i$ to all tentative nodes. In each step of the algorithm the tentative node $y$ with the shortest distance to the source node $i$ is made permanent, its neighbors that are not already tentative or permanent are made tentative and the distances to all tentative neighbors of $y$ are updated. In order to facilitate comparison with the other algorithms of this paper, we imagine a main processor at node $i$ that performs the main algorithm helped by a <u>slave</u> (also located at node $i$) whose role is to extract the adjacency array of a given node from the memory and forward it to the main processor.

<u>Assumption on the operation of the slave</u>

$ASK(\mu)$ denotes a request by the main processor to the slave asking for the adjacency array of node $\mu$ containing all neighbors of $\mu$; the assumption is that whenever such a request is released and only as a response to such a request, $ANS(\mu,\Lambda)$ is delivered by the slave within arbitrary finite time, where $\Lambda = (\Lambda \cdot n, \Lambda \cdot d)$, $\Lambda \cdot n = S_\mu$ and for all lines $r$ in $\Lambda$ we have $\Lambda \cdot d(r) = d_{\mu,\Lambda \cdot n(r)}$.

## THE CENTRALIZED DIJKSTRA ALGORITHM (CDA)

### Variables used by the algorithm at node i

| | | |
|---|---|---|
| $N_i$ | : | array $N_i = (N{\cdot}s_i, N{\cdot}d_i, N{\cdot}p_i)$ as described below ($\lvert V\rvert$ rows, 3 columns) |
| $N{\cdot}s_i(x)$ | : | status of node x: 2 = permanent, 1 = tentative, 0 = unknown (all $x \varepsilon V$) |
| $N{\cdot}d_i(x)$ | : | estimated distance to $x$ (all $x \varepsilon V$) |
| $N{\cdot}p_i(x)$ | : | identity of predecessor of $x$ on the path from i to x (all $x \varepsilon V$) |
| $m_i$ | : | the node to be made permanent next. |

### Internal messages to/from processor

| | |
|---|---|
| ASK($\mu$) | = message to slave requesting the adjacency array that contains all neighbors of node $\mu$ |
| ANS($\mu,\Lambda$) | = message from slave providing adjacency array $\Lambda$ of node $\mu$ with $\Lambda{\cdot}n = S_\mu$. |
| START | = command given to the main processor to start algorithm |

### The algorithm at node i

```
Initial state : $N_i = (0,\infty, \text{nil})$
1.  For[1] START
2.          $N_i(i) \leftarrow (1,0, \text{nil})$ ; ASK(i) .
3.  For ANS(i,$\Lambda$)
4.          $N{\cdot}s_i(i) \leftarrow 2$; $\forall x \varepsilon \Lambda{\cdot}n$, set $N_i(x) \leftarrow (1, d_{ix}, i)$;  go to <11> .
5.  For ANS($\mu,\Lambda$) ,  $\mu \neq i$
6.          $N{\cdot}s_i(\mu) \leftarrow 2$ ;
7.          $\forall$ rows r of $\Lambda$, let $x = \Lambda{\cdot}n(r)$ and
8.              if $N{\cdot}s_i(x) < 2$ and $N{\cdot}d_i(\mu) + \Lambda{\cdot}d(r) \prec N{\cdot}d_i(x)$
9.                  then $N_i(x) \leftarrow (1, N{\cdot}d_i(\mu) + \Lambda{\cdot}d(r), \mu)$ ;
10.         if $\forall x$ holds $N{\cdot}s_i(x) \neq 1$, then STOP
11.             else $m_i \leftarrow y^*$ that achieves min $\{N{\cdot}d_i(y) \mid N{\cdot}s_i(y) = 1\}$
12.                 ASK($m_i$)
```

In order to describe the properties of the algorithm, we need the following

Definition 3.1

a) If $k \in P_i \cup T_i$ (i.e. $N \cdot s_i(k) \neq 0$), we say that $k$ is known at $i$. Then the path $(i=i_o, i_1, i_2, \ldots, i_m = k)$ defined by $i_{n-1} = N \cdot p_i(i_n)$ is said to be the path to $k$ known at $i$. This path can be found from table $N_i$ by going backwards from node $k$.

b) If $k \notin P_i \cup T_i$, we say $k$ is unknown at $i$.

c) If $k$ is known at $i$ and the path known is the shortest path MMP(i,k), we say that $k$ is strongly known at $i$.

The fundamental properties of CDA, as well as of the other algorithms of the paper are :

Fundamental properties

a) If $x \in P_i$, then $N \cdot p_i(x) \in P_i$, node $x$ is strongly known at $i$ and $N \cdot d_i(x) = MMD(i,x)$.

b) If $x \in T_i$, then $N \cdot p_i(x) \in P_i$, node $x$ is known at $i$ at MP(on $P_i$, $i,x$) and $N \cdot d_i(x) = MD(\text{on } P_i, i,x)$.

c) STOP occurs in finite time and whenever this happens (i.e., whenever all $N \cdot s_i(x)$ are 0 or 2), the algorithm is completed. At that time $P_i$ = {all nodes reachable from $i$}, $T_i = \phi$, $V-P_i$ = {all nodes nonreachable from i} .

Theorem 3.1 [1], [2]

The fundamental properties a), b), c) hold for CDA and in addition :

d) at all times holds $\bigcup_{x \in P_i} S_x = P_i \cup T_i$

e) nodes $x$ become permanent (i.e. $N \cdot s_i(x) \leftarrow 2$ is performed) in the order of increasing distance from $i$ in the sense of Definition 2.1.

## 4. The Extended Centralized Dijkstra Algorithm (ECDA)

The distributed protocol to be presented in Section 5 is based on an extended version of the centralized Dijkstra algorithm. The former consists of two major mechanisms : the computation at the nodes and the communication between neighbors. The main processor at a node performs the algorithm by using timing and topological information received from the communication mechanism. For purposes of presentation it is convenient to extract from the distributed protocol the communication part and replace it by two imaginary processes : an oracle that provides timing information and a slave that gives topological information, the latter being slightly different from Section 3. The result is the extended centralized Dijkstra algorithm presented below which is not an implementable algorithm, but rater an illustrative one, but it allows us to present separately the computation and the communication mechanisms of the decentralized protocol. The idea is that in Section 5 we show that the communication between neighbors can play the role of both the oracle and the slave.

As in Section 3, the present algorithm finds the shortest paths and their lengths from a given node $i$ to all other nodes in the network. The oracle and the slave, as well as the main processor, are located at node $i$.

Assumptions on the operation of the oracle and the slave

4.1) The oracle may find out (in some yet unspecified way) that some node $\mu$ can be made permanent at some time $t$, even though it is not its turn according to[2] <11> in CDA (see also Theorem 3.1 e)); the assumption is that this can happen only if both $N{\cdot}s_i(\mu)\neq 0$ and $N{\cdot}d_i(\mu)=MMD(i,\mu)$ hold (in words, only if $\mu$ is strongly known).

4.2) ASK($\mu$) denotes now a request by the main processor to the slave for some adjacency array of $\mu$ that includes the set $R_{i\mu}$ of sons of $\mu$; the assumption here is that $ANS(\mu,\Lambda)$ can be delivered by the slave only as a response to such a request and then $R_{i\mu} \subseteq \Lambda\cdot n \subseteq S_\mu$ and $\Lambda\cdot d(r)=d_{\mu,\Lambda\cdot n(r)}$ for all $r$.

4.3) whenever a request $ASK(\mu)$ is released, then $ANS(\mu,\Lambda)$ is delivered by the slave within arbitrary finite time.

The exact algorithm is given below.

THE EXTENDED CENTRALIZED DIJKSTRA ALGORITHM (ECDA)

Variables used by the algorithm at node i

Same as in CDA and in addition :

| | |
|---|---|
| $A_i$ | : array $A_i=(A\cdot n_i)$ (one column, variable length) |
| $A\cdot n_i(r)$ | : node designated by the oracle for which ANS has not been received yet |

Messages to/from processor

Same as in CDA except that $R_{i\mu} \subseteq \Lambda\cdot n \subseteq S_\mu$ and in addition

$ORACLE(\mu)$ = oracle designates node $\mu$.

The algorithm at node i

Initial state : $m_i$ = nil, $N_i = (0,\infty,$ nil), $A\cdot n_i$ = nil

```
1.  For START
2.      $N_i(i) \leftarrow (1,0,nil)$;  $ASK(i)$.
3.  For $ANS(i,\Lambda)$
4.      $N\cdot s_i(i) \leftarrow 2$, $\forall x \in \Lambda\cdot n$,  set $N_i(x) \leftarrow (1,d_{ix},i)$;  go to <18>.
5.  For $ORACLE(\mu)$  (comment: by Assumption 4.1, holds $N\cdot s_i(\mu) \neq 0$)
6.      if $N\cdot s_i(\mu)\neq 2$
7.          then if $\mu\neq m_i$ and $\mu\neq i$ and $\mu\notin A\cdot n_i$
8.                  then $ASK(\mu)$
9.               enter $\mu$ into $A\cdot n_i$
10. For $ANS(\mu,\Lambda)$,  $\mu\neq i$
11.     $N\cdot s_i(\mu) \leftarrow 2$
12.     $\forall$ rows $r$ of $\Lambda$, let $x = \Lambda\cdot n(r)$ and
13.         if $N\cdot s_i(x) < 2$ and $N\cdot d_i(\mu) + \Lambda\cdot d(r) < N\cdot d_i(x)$
14.             then $N_i(x) \leftarrow (1, N\cdot d_i(\mu) + \Lambda\cdot d(r), \mu)$
15.     delete all entries $\mu$ from $A\cdot n_i$
16.     if $\mu=m_i$
17.         then if $\forall x$ holds $N\cdot s_i(x)\neq 1$, then STOP,
18.                 else $m_i \leftarrow y^*$ that achieves min $\{N\cdot d_i(y)|N\cdot s_i(y) = 1\}$
19.                     if $m_i \notin A\cdot n_i$, then $ASK(m_i)$
```

Our goal is to show that the extended algorithm has properties similar to those of Theorem 3.1, but we first need some preliminary properties.

Lemma 4.1

a) <4> is executed at most once, at time $t_i^*$ say .

b) Node $\mu$ enters $A \cdot n_i$ iff ORACLE($\mu$) is received and $N \cdot s_i(\mu) \neq 2$; it stays in $A \cdot n_i$ until ANS($\mu,\Lambda$) is received, at which time all entries $\mu$ in $A \cdot n_i$ (and in $m_i$, if $m_i = \mu$) are deleted and $N \cdot s_i(\mu) \leftarrow 2$.

c) No ANS($\mu,\Lambda$) or ORACLE($\mu$) with $\mu \neq i$ can be received at node i before $t_i^*$.

d) $N \cdot s_i(x)$ is non-decreasing for any given x. After $m_i \leftarrow \mu$ in <18>, the contents of $m_i$ remains unchanged until ANS($\mu,\Lambda$) is received, at which time $N \cdot s_i(\mu) \leftarrow 2$, $\mu$ is deleted from $m_i$ (and from $A \cdot n_i$ if $\mu \in A \cdot n_i$); afterwards $\mu$ will never enter $A \cdot n_i$ or $m_i$. Similarly once $\mu$ is deleted from $A \cdot n_i$ and possibly from $m_i$ as described in b) above, it will never enter $A \cdot n_i$ or $m_i$.

e) For each $\mu$, no more than one ASK($\mu$) is requested and no more than one ANS($\mu,\Lambda$) is received by i.

Proof :

a) After $t_i^*$ we have that $N \cdot s_i(i) = 2$ and cannot be changed, hence no ASK(i) can be sent. Therefore a) is proved if we show that no ASK(i) can be sent between execution of <1> and $t_i^*$. Let $t_1$ be the time when the first such ASK(i) is sent. This can happen only in <19> and let $t_2 \leq t_1$ be the first time <18> is entered. At $t_2$, <18> cannot be entered through <16> since $m_i(t_2)$ = nil and also cannot be entered through <4> since $t_2 < t_i^*$, which leads to a contradiction.

b) follows from <9>, <11>, <15>.

c) Suppose that an ANS($\mu,\Lambda$), $\mu \neq i$ is received for the first time at time $t_1 < t_i^*$. Then by Assumption 4.2, ASK($\mu$) was sent at some time $t_2 < t_1$. Now ASK($\mu$) was not sent in <19> since this would imply either that $t_2 = t_i^*$ (if <19> was reached from <4>) or that ANS($\mu_1,\Lambda_1$), $\mu_1 \neq i$ was received at $t_2 < t_1$ (if <19> was reached from <10>). Consequently ASK($\mu$) was sent in <8> as a result of <5>, which implies from Assumption 4.1 that $N \cdot s_i(\mu)(t_2) \neq 0$. But this is a contradiction because the initial state is $N \cdot s_i(\mu) = 0$ and only <4>, <11> or <14> can change this value, which means that $ANS(\mu,\Lambda)$ has been received before $t_2 < t_1 < t_i^*$, while we assumed above that $t_1$ is the first time such a message is received. This proves the first part of c) and the second follows since $N \cdot s_i(\mu)(\tau) = 0$ for all $\mu \neq i$, $\tau < t_i^*$.

d) Since $N \cdot s_i(x)$ can be changed only in <2>, <4>, <11> or <14>, parts a) and c) above imply that $N \cdot s_i(x)$ is non-decreasing. As a result and by <18>, <19>, once $m_i \leftarrow \mu$, the contents of $m_i$ remains unchanged until $ANS(\mu,\Lambda)$ is received, and then $N \cdot s_i(\mu) \leftarrow 2$, $\mu$ is deleted from $m_i$ and possibly $A \cdot n_i$. This part, together with <6> and <9>, or <18> completes th proof of d).

e) follows from the fact that $ASK(\mu)$ is sent either while $\mu$ first enters $A \cdot n_i$ or when $m_i \leftarrow \mu$, whichever comes first

The next Theorem is the equivalent of Theorem 3.1 for the ECDA and summarizes its main properties. The major difference is that d) and e) of Theorem 3.1 do not necessarily hold for ECDA.

<u>Theorem 4.1</u>

Under Assumptions 4.1) and 4.2), the Fundamental Properties a), b) hold for ECDA and in addition :

d) At all times holds $\bigcup_{x \in P_i} R_{ix} \subseteq P_i \cup T_i \subseteq \bigcup_{x \in P_i} S_x$, and if $x \in P_i$ and $y \in R_{ix}$, then $N \cdot p_i(y) = x$.

Provided that Assumption 4.3) holds also, Fundamental Property c) holds.

<u>Proof</u> :

Lemma 4.1 shows that algorithm ECDA works in the same way as CDA except for two features : first, a tentative node can be designated to become permanent not only by a minimization procedure (<11> in CDA) but also by an oracle and second, the list $\Lambda \cdot n$ in messages $ANS(\mu,\Lambda)$, may be a proper subset of $S_\mu$, provided that $\Lambda \cdot n \supseteq R_{i\mu}$. Because of the second feature, ECDA may not improve distances to some neighbors of $\mu$ that are not

sons of $\mu$ (see Assumption 4.2), while CDA does improve them. But since such a neighbor $x$ will finally be reported in some ANS($\mu_1,\Lambda_1$) where $x$ is the son of $\mu_1$, the validity of the algorithm is not affected. Also, Assumption 4.2) shows that d) holds.

Next, suppose that at a given stage the sets $P_i$ and $T_i$ verify a) and b). Then the node in $T_i$ verifying <18> can be transferred to $P_i$ according to CDA, and a node in $T_i$ designated by the oracle can also be transferred to $P_i$ since, by Assumption 4.1), it verifies the fundamental property a) of a node in $P_i$. The subsequent use of this node to reduce the distances of adjacent nodes belonging to $T_i$ restores to $T_i$ its property b), as in the CDA. Observe that Assumption 4.3) has not been used up to this point. In order to prove that c) holds, assume the contrary. Then there is a node $x$ reachable from $i$ with $N \cdot s_i(x) = 0$. Let $y$ be the node that is closest to $i$ on MMP(i,x), with $N \cdot s_i(y) = 0$, and let $z$ be its predecessor on the path. Clearly $z \in P_i$ and d) implies that $R_{iz} \subseteq P_i \cup T_i$. But $y \in R_{iz}$ and $y \in P_i \cup T_i$ which is a contradiction. Since at each step of the algorithm d) holds and a new node is transferred from $T_i$ to $P_i$, and since the intervals between these events are finite by Assumption 4.3), the algorithm will terminate in finite time.

## 5. Distributed Dijkstra Protocol (DDP)

In this section we present a distributed protocol that computes the shortest paths from all nodes to all nodes in the network and is based on the Dijkstra algorithm. Just before entering the protocol, each node is assumed to keep only its own identity, the weights of the outgoing adjacent links and the identities of nodes that are potentially in the network. When a node completes the protocol, it will have the identities of the nodes that are reachable and the shortest path and distance to each. In the distributed protocol, neighboring nodes exchange control messages whose role is to propagate topological and timing information. As such, the operations performed by each node serve a double purpose : advancing the algorithm at the node and helping neighboring nodes to obtain information that will allow them to proceed with their algorithm. In fact it turns out, as we shall see presently, that some of the operations can serve both purposes.

As in the centralized algorithms, a node $i$ maintains the sets $P_i$ of permanent nodes and $T_i$ of tentative nodes, which together form the set of known nodes, while all the others are said to be unknown at $i$. Since the distributed protocol is exactly the ECDA with the communication mechanism replacing the slave and the oracle, we may assume for the moment for illustration purposes that all properties of ECDA hold here also. Now, whenever a new node $\mu$ is to be made permanent (as in <8> or <18> of ECDA), we have that $N \cdot s_i(\mu) = 1$, namely $\mu$ is tentative at $i$ and moreover, it will be shown in Lemma 5.2 that at this time $\mu$ is strongly known at $i$ (Definition 3.1). In the distributed protocol we require that at this time node $i$ sends ASK($\mu$) to the first node $f_i(\mu)$ on MMP($i,\mu$). As such, the communication with this neighbor plays the role of the slave at node $i$ in ECDA.

Next we look at what happens at node $j = f_i(\mu)$ when it receives ASK($\mu$). First, it is shown in Lemma 5.2 that $\mu$ must be strongly known at $j$ too, so that receiving ASK($\mu$) can play the role of the oracle at node $j$ (see Assumption 4.1). Now, $\mu$ can be either permanent or tentative at $j$. In the first case, $j$ can return ANS$(\mu,\Lambda_j)$ to i, where $\Lambda \cdot n_j$ includes the set $R_{j\mu}$ of sons of $\mu$ for j and we show in Lemma 5.2 that $R_{j\mu}$ includes the set $R_{i\mu}$ as required in Assumption 4.2. On the other hand, if $\mu$ is tentative at j, then j can forward ASK$(\mu)$ to the next node $f_j(\mu)$ on MMP$(j,\mu)$ and the procedure can be repeated until ASK$(\mu)$ reaches a node where $\mu$ is permanent. When ANS$(\mu,\Lambda)$ will eventually be received by j, the vector $\Lambda \cdot n$ will include the set $R_{j\mu}$ of sons of node $\mu$. At that time, according to ECDA, $\mu$ can be made permanent at j, even though its turn has not come yet according to Theorem 3.1 e). Also, now j can send ANS$(\mu,\Lambda_j)$ to i, where $\Lambda \cdot n_j \supseteq R_{j\mu} \supseteq R_{i\mu}$ .

We next present the exact algorithm performed by each node in order to implement the protocol.

<u>THE DISTRIBUTED DIJKSTRA PROTOCOL (DDP)</u>

<u>Variables used by the algorithm at node i</u>

Same as in CDA, and in addition :

$A_i = (A \cdot n_i, A \cdot f_i)$ : array (2 columns, variable length), where a row r consists of:

$A \cdot n_i(r) = \mu$ if ASK$(\mu)$ was received and forwarded

$A \cdot f_i(r)$ : denotes link on which ASK$(A \cdot n_i(r))$ was received

$L_i = (L \cdot n_i, L \cdot s_i)$ : array (2 columns, nr. of rows = nr. of links adjacent to i as described below

$L \cdot n_i(\ell)$ = identity of node at the other end of link $\ell$ if $L \cdot s_i(\ell) = 1$ and = nil if $L \cdot s_i(\ell) = 0$.

$L \cdot s_i(\ell) = 0$ before WAKE is received on link $\ell$, = 1 afterwards

$\text{mode}_i = -1$ before i enters the protocol, = 0 afterwards

<u>Messages sent and received by the algorithm at node i</u>

START whose meaning is as in CDA, can be received provided that $\text{mode}_i = -1$ (observe that any number of nodes may asynchronously receive START)

WAKE(i) sends the identity of i to all neighbors (plays the role of ASK(i) of ECDA); receipt of the first WAKE signals node i to enter the protocol unless START was received previously and receipt of WAKE from all neighbors plays the role of ANS$(i,\Lambda)$ of ECDA.

ASK$(\mu)$ requests any adjacency array of $\mu$ that includes its sons

ANS$(\mu,\Lambda_i)$ sends list of nodes $\Lambda_i$ with the same structure as in ECDA

ANS$(\mu,\Lambda)$ received message with the same structure as before

The algorithm at node i

Just before entering algorithm, it is assumed that : $\text{mode}_i = -1$, $A_i$ = empty, $m_i$ = nil, $L_i = (\text{nil}, 0)$, $N_i = (0,\infty, \text{nil})$.

```
1.  For START
2.     $N_i(i) \leftarrow (1,0,\text{nil})$; $\text{mode}_i \leftarrow 0$; send WAKE(i) on all adjacent links.
3.  For WAKE(j) received on link $\ell$
4.     if $\text{mode}_i = -1$, same as <2>.
5.     $L_i(\ell) = (j,1)$
6.     if $\forall x$, holds $L \cdot s_i(x) = 1$
7.        then $N \cdot s_i(i) \leftarrow 2$; $\forall x \in L \cdot n_i$, set $N_i(x) \leftarrow (1, d_{ix}, i)$
8.             same as <20> - <21> with i replacing $\mu$; go to <24>.
9.  For ASK$(\mu)$ received on link $\ell$
10.    if $N \cdot s_i(\mu) = 2$
11.       then same as <20>; send ANS$(\mu,\Lambda_i)$ on $\ell$
12.       else if $\mu \neq i$, $\mu \neq m_i$ and $\mu \notin A \cdot n_i$
13.               then send ASK$(\mu)$ to $\text{first}_i(\mu)$  (defined below)
14.            enter row $(\mu,\ell)$ into $A_i$
15. For ANS$(\mu,\Lambda)$ received on link $\ell$
16.    $N \cdot s_i(\mu) \leftarrow 2$
17.    $\forall$ rows r in $\Lambda$, let $x = \Lambda \cdot n(r)$ and
18.       if $N \cdot s_i(x) < 2$ and $N \cdot d_i(\mu) + \Lambda \cdot d(r) < N \cdot d_i(x)$
19.            then $N_i(x) \leftarrow (1, N \cdot d_i(\mu) + \Lambda \cdot d(r), \mu)$
20.    $\Lambda_i \leftarrow \{(x, D_x) | N \cdot p_i(x) = \mu\}$ where $D_x = N \cdot d_i(x) - N \cdot d_i(\mu)$
21.    $\forall r$ s.t. $A \cdot n_i(r) = \mu$, send ANS$(\mu,\Lambda_i)$ on $A \cdot f_i(r)$ and delete row r from $A_i$
22.    if $\mu = m_i$
23.       then if $\forall x$ holds $N \cdot s_i(x) \neq 1$, then STOP
24.               else $m_i \leftarrow y^*$ that achieves min $\{N \cdot d_i(y) | N \cdot s_i(y) = 1\}$
25.                    if $m_i \notin A \cdot n_i$, then send ASK$(m_i)$ to $\text{first}_i(m_i)$
```

Note : $\text{first}_i(x)$ is a function that returns the identity of the first node after i on the path to x known at i (see Definition 3.1); the corresponding link can be found from Table $L_i$.

Since the validation of the distributed protocol is based on comparison between ECDA and DDP, it is useful at this stage to indicate the corresponding steps:

| ECDA | DDP |
|---|---|
| <1>-<2> | <1>-<2> and <3>-<4> |
| <3> | <6> |
| <4> | <7>-<8> |
| none | <9>-<11> |
| <5>-<9> | <9>,<12>-<14> |
| <10>-<19> | <15>-<25> |

In order to validate the Dijkstra Distributed Protocol (DDP), we only have to show that the communication mechanism satisfies Assumptions 4.1, 4.2 and 4.3. We first need however several preliminary properties similar to those of Lemma 4.1.

Lemma 5.1

a) Each node i in the network executes either <2> or <4> (but not both) exactly once and this happens before node i executes any other part of the algorithm. WAKE is sent on any link before any other message and exactly once. Each node i executes <7>-<8> exactly once, at time $t_i^*$ say, and afterwards no WAKE is received.

b) At node i, row $(\mu,j)$ enters $A_i$ iff ASK($\mu$) is received from j and $N\cdot s_i(\mu)\neq 2$. In this case, row $(\mu,j)$ stays in $A_i$ until either i receives WAKE from all its neighbors (in the case when $\mu=i$) or i receives ANS($\mu,\Lambda$), at which time row $(\mu,j)$ is deleted from $A_i$ (and possibly from $m_i$) and $N\cdot s_i(\mu)\leftarrow 2$. Also, a message ANS($\mu,\Lambda$) is sent from i to j only if i has previously received ASK($\mu$) from j.

c) Before time $t_i^*$, no ASK is sent by $i$, no ANS is sent to $i$, no ASK($\mu$), $\mu \neq i$ is sent to $i$ and no ANS is sent by $i$.

d) $N \cdot s_i(x)$ is non-decreasing for any given $x$. After $m_i \leftarrow \mu$ in <24>, the contents of $m_i$ remains unchanged until ANS($\mu$,$\Lambda$) is received, at which time $N \cdot s_i(\mu) \leftarrow 2$ and $\mu$ is deleted from $m_i$ (and possibly from $A \cdot n_i$). Afterwards $\mu$ will never enter again $A \cdot n_i$ or $m_i$. Similarly, after row ($\mu$,j) is deleted from $A_i$ as described in b) above, $\mu$ will never enter again $A \cdot n_i$ or $m_i$.

e) For each $\mu$, no more than one ASK($\mu$) is requested by $i$ and no more than one ANS(m,$\Lambda$) is received at $i$.

<u>Proof</u> :

a) The proof of the first two statements is simple and will be omitted (see also [6, protocol PI]). In order to prove the last statement, observe that each node receives exactly one WAKE from each neighbor. Once all messages WAKE have been received, <6> holds and <7>, <8> are executed. Thereafter no WAKE can be received.

b) The part of b) concerning the operation of $A_i$ is easily proved by following the algorithm, (<9>,<10>,<14>,and <3>,<6>-<8>,<21> or <15>,<16>,<21>). In order to prove that ANS($\mu$,$\Lambda$) is sent from i to j only if j has previously sent ASK($\mu$) to i, observe that ANS($\mu$,$\Lambda$) is sent in <11> or <21>. If it is sent in <11>, it is the result of <9> and the statement is proved. If it is sent at <21>, then row ($\mu$,j) $\varepsilon$ $A_i$ and by the first part of b), ASK($\mu$) must have been previously received at $i$ from $j$.

c) First we prove that no ASK($\mu$) can be sent by $i$ before $t_i^*$. Let $t_1 < t_i^*$ be the time when the first ASK($\mu$) was sent by $i$ before $t_i^*$. This cannot happen in <25>, because <25> cannot be reached before $t_i^*$, by a proof similar to lemma 4.1 a). Hence ASK($\mu$) is sent at $t_1$ in <13> as a result of receiving ASK($\mu$) from some node $j$. Now let us look at what happens at node j. When j has sent ASK($\mu$), it was true that i = $\text{first}_j(\mu)$ and since $\mu \neq i$ from <12>, this implies that $N \cdot p_j(x) = i$ for some $x$. But $N \cdot p_j(x)$ could be set to $i \neq j$ only in <19>, as a result of $j$ receiving $ANS(i,\Lambda)$ from some node $k$ and let $t_2 < t_1$ be the time $k$ has sent this message. The following argument shows that this implies that node $i$ has sent $ANS(i,\Lambda)$ in <7> at some time before or at $t_2$: a node $x \neq i$ can send $ANS(i,\Lambda)$ at <11> or <21> and at that time $N \cdot s_x(i) = 2$; now $N \cdot s_x(i)$ can be set to 2 only in <16>, as a result of receiving $ANS(i,\Lambda)$ from some node $y$ who sent $ANS(i,\Lambda)$ before $x$ did, and we repeat the argument with $y$ instead of $x$. The only other way is $N \cdot s_x(i) \leftarrow 2$ in <7> and then $x=i$ proving the claim that $i$ has sent $ANS(i,\Lambda)$ before or at $t_2$. However this is a contradiction, since $i$ executes <7> only once at $t_i^*$ and $t_2 < t_i^*$. This completes the proof that no $ASK(\mu)$ can be sent by $i$ before $t_i^*$. Now, no $ANS(\mu,\Lambda)$ can be sent to $i$ before $t_i^*$, because by b) this would imply that $i$ has previously sent $ASK(\mu)$. Also, no $ASK(\mu)$, $\mu \neq i$ can be sent to $i$ before $t_i^*$, since the sending node $j$ must have $i = first_j(\mu)$ and this leads to a contradiction as above. Finally, suppose that $ANS(\mu,\Lambda)$ is sent by $i$ to $j$ at some time $t < t_i^*$. Since <8> is executed at time $t_i^*$, the considered ANS can be sent only in <11> or <21>. If in <11>, observe that $\mu \neq i$, since $N \cdot s_i(i)$ can become 2 only at $t_i^*$ (in <7>) or in <16> as a result of receiving ANS, and the latter cannot occur before $t_i^*$ as already proved. Therefore occurence of <11> or <21> requires that $ASK(\mu)$, $\mu \neq i$ or $ANS(\mu,\Lambda)$ respectively was sent to $i$ before $t$, and we have already proved that both situations cannot happen.

d) and e) are proved as in Lemma 4.1.

The next lemma proves that the communication mechanism of DDP has properties as required by Assumptions 4.1)-4.3) of ECDA. Assumption 4.1) is covered by a) parts c), d), e) cover Assumption 4.2) and f) corresponds to Assumption 4.3). Part b) is a stronger statement than a) and describes the coordination of the communication mechanism, thereby providing a tool for the proof of all other properties. The fact that DDP works according to the Dijkstra algorithm is shown in Theorem 5.1.

Lemma 5.2

a) ASK($\mu$) can be received by $i$ from $j$ only if $i = f_j(\mu)$ (see Definition in Section 2) and if $t$ is the time this happens, then $\mu$ is strongly known at $i$ at time $t-$.

b) ASK($\mu$) can be sent by $i$ to $j$ only if $i = f_j(\mu)$, and if $t$ is the time this happens, then both $i$ and $j$ know strongly $\mu$ at time $t$ and $N \cdot s_i(\mu)(t)=1$.

c) ANS($\mu,\Lambda$) can be sent by $i$ to $j$ only if ASK($\mu$) has previously been received by $i$ from $j$. (This has already been proved in Lemma 5.1 b)).

d) In any ANS($\mu,\Lambda$) holds $\Lambda \cdot n \subseteq S_\mu$ and $\Lambda \cdot d(r) = d_{\mu,\Lambda \cdot n(r)}$ for any $r$.

e) ANS($\mu,\Lambda$) can be received by $i$ from $j$ only if $j = f_i(\mu)$ and then $\Lambda \cdot n \supseteq R_{i\mu}$.

f) If ASK($\mu$) is sent by $i$ to $j$, then ANS($\mu,\Lambda$) is received by $i$ from $j$ within finite time.

Theorem 5.1

The fundamental properties a), b), c) presented in Theorem 3.1 hold for DDP and in addition Theorem 4.1 d) holds.

The proof of Lemma 5.2 and Theorem 5.1 appears in the Appendix and proceeds by a common induction. The fact that ECDA has already been proved, allows us to immediately deduce that if the properties of Lemma 5.2 hold up to time $t$, then Theorem 5.1 must hold also. Therefore, all is left is to prove that the properties of Lemma 5.2 (communication properties) hold at a given time $t$ based on the induction hypothesis that Lemma 5.2 and Theorem 5.1 are true up to time $t-$.

## 6. Conclusions

This work presents a distributed version of the Dijkstra shortest path algorithm and its formal proof using a new validation approach for distributed protocols.

As in Friedman [4], we take advantage of the fact that adjacency arrays of new permanent nodes need not contain all neighbors of that node, a property that reduces the amount of computation of shortest paths as well as the lengths of messages of the type ANS$(\mu,\Lambda)$. In addition, considering the fact that tentative nodes need not become permanent in order of increasing distances and that any new permanent node $\mu$ at some node $i$ is also strongly known at all nodes on the shortest path from $i$ to $\mu$, we set up a communication procedure which speeds up the protocol as compared to Friedman [4], without increasing the communication complexity. For example, in the network of Fig. 1 with all weights = 1, if the communication between nodes 3 and 2 is slow, then in our protocol nodes 3 and 1 will add nodes 4,5,6 to the list of permanents, while in the Gallager-Friedman protocol [4], they will first wait for node 2 to become permanent at 3.

The communication complexity of our algorithm is computed as follows : Each node sends a WAKE message to its neighbors requiring a total of $2|E|$ WAKE messages. Each node $i$ sends exactly one ASK$(\mu)$ and one ANS$(\mu,\Lambda)$ for each node $\mu \neq i$ in $V$. (Notice that these messages are sent on the tree of shortest paths to $\mu$). Thus $|V|(|V|-1)$ messages of each kind are sent and therefore the total number of messages required by the protocol is $2(|E| + |V|(|V|-1) \approx 2(|E| + |V|^2)$. In the sequel we neglect message headers and denote by $w$ the number of bits necessary to encode a link weight. Then, since it takes $\log|V|$ bits to encode a node identity, WAKE$(j)$ and ASK$(\mu)$ messages are responsible for transmission of approximately $(2|E| + |V|^2) \log |V|$ bits (all logarithms are base 2). On the other hand, $\Lambda \cdot n$ in message ANS$(\mu,\Lambda)$ contains up to $|S_\mu| \simeq \frac{2|E|}{|V|}$ nodes (and this happens in general in messages sent on links close to $\mu$) and down to $|R_{i\mu}|$ nodes, mostly in messages sent on links far from $\mu$. Therefore an upper bound for the total number of bits sent in ANS$(\mu,\Lambda)$ messages is $|V|(|V| - 1)\left(\frac{2|E|}{|V|} + 1\right)(\log |V| + w) \approx 2|E||V|(\log |V| + w)$ bits. In the same way, if we had $\Lambda \cdot n = R_{i\mu}$ in each message ANS$(\mu,\Lambda)$, then each node $\nu$ travels exactly twice on each branch of the tree of shortest paths to $\nu$ (once as $\mu$ and once in $\Lambda$), so that the minimum total number of bits sent in ANS messages is $|V|^2(2\log|V| + w)$.

As said before, the communication complexity of the Gallager-Friedman [4] algorithm is similar to ours. Another comparison can be made with the Gallager protocol [3], [6] for obtaining minimum hop paths. The number of required messages in that protocol is $2|E|\ (\ell+1)$, where $\ell$ is the average depth of the minimum hop tree in the network and the total number of bits is $2|E||V| \log |V|$. Our protocol reduced to this particular case (all weights are unity and hence $w = 0$) requires approximately $2(|E|+ |V|^2)$ messages and the total number $B$ of bits is bounded approximately by :

$$(2|E| + 3|V|^2)\log|V| \leqslant B \leqslant (2|E||V| + 2|E| + |V|^2) \log|V|$$

On the other hand, our protocol may advance faster than Gallager's [3] for the same reasons as in the comparison with the Gallager-Friedman protocol.

## Appendix

This Appendix contains the proofs of the properties of DDP that do not appear in the body of the paper. The properties are proved in a different order than as presented in the paper, because in the latter they are given in an order that is appropriate for illustration.

Proposition A.1

i) In any $ANS(\mu,\Lambda)$ holds $\Lambda{\cdot}n \subseteq S_\mu$ .

ii) If $N{\cdot}p_i(x) \neq nil$, then $N{\cdot}p_i(x) \varepsilon S_x$.

Proof

Assertion i) is part of Lemma 5.2 d), while ii) is necessary for the proof, which proceeds by a common induction on time. Both claims clearly hold when the first node in the network enters the algorithm. Suppose now that they hold until time $t-$ and observe that the events that can affect the claims are <6>, <10> or <15>, at a node $i$ say. For the first case, <5> assures that only neighbors of $i$ enter $L{\cdot}n_i$ and hence ii) is preserved, which implies that in <8> only neighbors of $i$ enter $\Lambda{\cdot}n_i$, preserving i). In the second case, $N_i$ is not altered at $t$, hence ii) is not affected and in <11> only nodes $x$ with $N{\cdot}p_i(x) = \mu$ enter $\Lambda_i$, so that i) holds. Finally if $ANS(\mu,\Lambda)$ is received at $i$ at time $t$ in <15>, we have $\Lambda{\cdot}n \subseteq S_\mu$ by the induction hypothesis. In <19>, $N{\cdot}p_i(x) \leftarrow \mu$ only if $x \varepsilon \Lambda{\cdot}n$, hence ii) continues to hold and in <20>, node $x$ enters $\Lambda{\cdot}n_i$ only if $N{\cdot}p_i(x) = \mu$, hence i) continues to hold for $\Lambda_i$.

Proposition A.2

i) Lemma 5.2 a)

ii) Lemma 5.2 e)

iii) In any message $ANS(\mu,\Lambda)$ holds $\Lambda{\cdot}d(r) = d_{\mu,\Lambda\cdot n(r)}$ for all rows $r$ in $\Lambda$ (this is the yet unproved parts of Lemma 5.2 d).

iv) Fundamental properties a), b) hold.

v) Theorem 4.1 d) holds.

vi) Any node that is strongly known at $i$ at time $t_1 < t$ is also strongly known at $i$ at $t$.

vii) Lemma 5.2 b).

Proof

Note that except for part of the termination (Fundamental Property c)), all properties that have not been proven yet are included here. The proof proceeds by a common induction, assuming that all properties hold in the entire network up to time $t-$ and proving that they continue to hold at time t.

i) Let $t_1 < t$ be the time when $j$ has sent the message ASK($\mu$). By vii) applied at time $t_1$ at $j$, we have that $i = f_j(\mu)$ and also that $\mu$ is strongly known at $i$ at time $t_1$. As a result, vi) implies that $\mu$ is strongly known at $i$ at time $t-$.

ii) Node $j$ has sent ANS($\mu,\Lambda$) in <8>, <11> or <21> and let $t_1 < t$ be the time this happened. In the first case $\mu = j$ and $\Lambda \cdot n = S_j$ and hence the claim holds. If ANS was sent in <11> or <21>, then $\Lambda \cdot n$ contains all nodes $x$ with $N \cdot p_j(x)(t_1) = \mu$ and it is also true that $N \cdot s_j(\mu)(t_1) = 2$ (i.e. $\mu \in P_j$). Consequently v) applied at time $t_1$ at $j$ implies that any $y$ in $R_{j\mu}$ is in $\Lambda \cdot n$, meaning that $R_{j\mu} \subseteq \Lambda \cdot n$. Now the fact that ANS is sent by $j$ to $i$ implies by Lemma 5.1 b) that $j$ has previously received ASK($\mu$) from $i$ and hence i) implies that $j = f_i(\mu)$. This last fact, together with the remarks following Definition 2.1 say also that $R_{i\mu} \subseteq R_{j\mu}$, completing the proof.

iii) Observe first that at any time when iv) holds, if $x \in P_i \cup T_i$ and $N \cdot p_i(x) = \mu$, then $N \cdot d_i(x) = N \cdot d_i(\mu) + d_{\mu x}$ (follows from the definitions of "known"). Now, a node $i$ can build a new message ANS($\mu,\Lambda$) in <8>, <11> or <20>, and any $x$ entering $\Lambda \cdot n_i$ is permanent or tentative. If $N_i(x)$ is not changed just before $x$ enters $\Lambda \cdot n_i$ (this can happen in <11> or <20>) and since iv) holds at time t- by the induction hypothesis, then $\Lambda \cdot d_i(x) = D_x = d_{\mu x}$. If $N_i(x)$ is changed, and this can happen in <7> or <19>, then $D_x = d_{ix}$ or $D_x = \Lambda \cdot d(r)$ respectively, where $x = \Lambda \cdot n(r)$. In the second case, the claim follows from the induction hypothesis on iii).

iv)v) Theorem 4.1 says that if Assumptions 4.1) and 4.2) hold up to and including time $t$, then Fundamental Properties a), b) and Theorem 4.1 d) hold also on this interval. Now observe that Propositions A.1 i), A.2 i), ii), iii) and the fact that ANS is received only as a result of ASK (part of Lemma 5.2 c)), cover Assumptions 4.1) and 4.2). The fact that these properties hold up to and including time $t$ has already been proved under the induction hypothesis, the previous sections of Proposition A.2 and Lemma 5.1 b). Consequently iv), v) hold at time $t$.

vi) follows from the fact that iv) implies that <18> cannot hold for a node that is strongly known at $i$.

vii) First we prove the facts that node i knows strongly $\mu$ at time t, $N \cdot s_i(\mu)(t)=1$ and $j=f_i(\mu)$. Node $i$ sends $ASK(\mu)$ at <13> or <25>. If in <13>, then $ASK(\mu)$ was received by $i$ at $t$ and i) implies that node $\mu$ is strongly known at $i$ at time t-. Hence it is strongly known at $t$, since $N_i(\mu)$ is not changed at time $t$. Also the fact that <10> does not hold implies $N \cdot s_i(\mu) = 1$ and <13> implies by iv) that $j = \text{first}_i(\mu) = f_i(\mu)$. Now if node $i$ sends $ASK(\mu)$ in <25>, then $\mu = m_i$ and $N \cdot s_i(\mu) = 1$. In this case $\mu = m_i$ is the node that minimized $N \cdot d_i$ among tentative nodes and as in the proof of Theorem 4.1 a), $m_i$ is strongly known at $i$ and hence $j = \text{first}_i(\mu) = f_i(\mu)$. Next we show that at time t, node $\mu$ is strongly known at j too. Since a node always strongly knows itself, we need consider only the case $\mu \neq j$. Let $\nu = N \cdot p_i(\mu)(t)$ and observe that from the previous part $j = f_i(\mu) = f_i(\nu)$.

Also note that $N \cdot p_i(\mu)$ was assigned the value $\nu$ in <19> as a result of node $i$ receiving $\mu$ in $\Lambda \cdot n$ of $ANS(\nu,\Lambda)$ at some time $t_1 < t$. Then ii) implies that this ANS was received from $j$ and when it was sent by $j$, it was true that $N \cdot p_j(\mu) = \nu$ and $\nu \in P_j$. Therefore $\mu$ was strongly known at $j$ at that time and from vi) it is strongly known at $j$ at time $t$.

Proposition A.3

i) Lemma 5.2 f)

ii) Fundamental Property c)

Proof

Observe that since Lemma 5.2 f) covers Assumption 4.3), Theorem 4.1 assures that i) implies ii). To prove i) note that Lemma 5.2 b) implies that node $\mu$ is strongly known at $i$ at the time $t$ when $ASK(\mu)$ is sent to $j$ and also $j = f_i(\mu)$. Let $i = i_o, i_1, i_2, \ldots, i_m = \mu$ be the path $MMP(i,\mu)$ to $\mu$ known at $i$. The algorithm dictates that a node $j$ sends $ASK(\mu)$ to $f_j(\mu)$ as soon as it receives $ASK(\mu)$, unless it has sent $ASK(\mu)$ before. Therefore any node $i_k$ sends $ASK(\mu)$ to $i_{k+1}$ at some finite time before or after $t$. Node $\mu$ sends $ANS(\mu,\Lambda)$ to $i_{m-1}$ at $t_\mu^*$ or whenever it receives $ASK(\mu)$ from $i_{m-1}$, whichever comes later. Every node $i_k$ sends $ANS(\mu,\Lambda)$ to $i_{k-1}$ whenever it receives $ASK(\mu)$ from $i_{k-1}$ or upon receipt of $ANS(\mu,\Lambda)$ from $i_{k+1}$, whichever comes later. Consequently any node $i_k$ will eventually send $ANS(\mu,\Lambda)$ to $i_{k+1}$, completing the proof.

Footnotes

1. "For...." stands for "the operations performed by the processor when receiving ....".
2. The notation <·> indicates a line in an Algorithm. If not explicitly said otherwise, the reference is to the Algorithm currently under consideration.

## References

[1] E.W. Dijkstra, A note on two problems in connection with graphs, Numerische Mathematik 1, pp. 269-271, 1959.

[2] S.E. Dreyfus, An appraisal on some shortest path algorithms, Operations Research 17, pp. 395-410, 1969.

[3] R.G. Gallager, personal communication.

[4] D.U. Friedman, Communication complexity of distributed shortest path algorithms, Report LIDS-TH-886, MIT, Feb. 1979.

[5] R.G. Gallager, A shortest path routing algorithm with automatic resynch, unpublished note, March 1976.

[6] A. Segall, Distributed network protocols, submitted to IEEE Trans. on Infor. Theory.

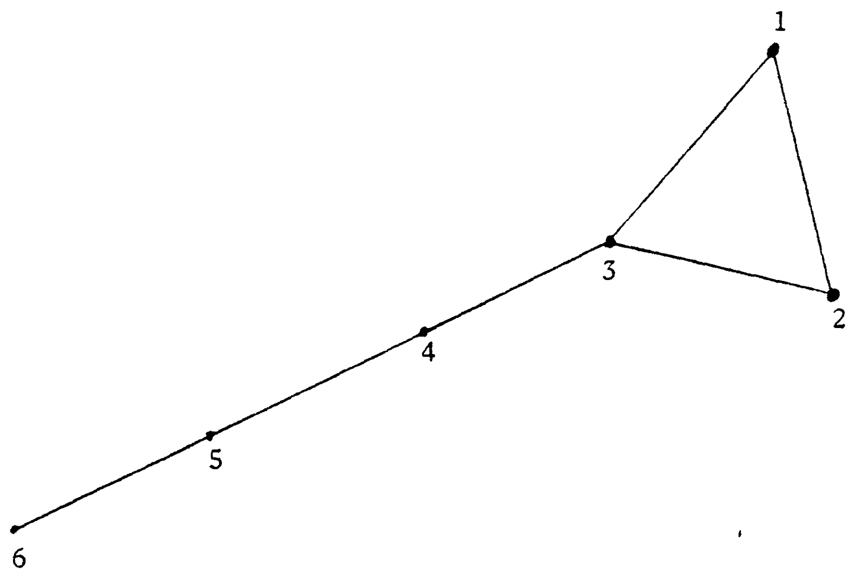

Fig. 1 - Example for Section 6.