A Fast Distributed Shortest Path Algorithm for a Class of Hierarchically Clustered Data Networks

John K. Antonio, *Member, IEEE*, Garng M. Huang, *Senior Member, IEEE*, and Wei K. Tsai, *Member, IEEE*

Abstract—**A new distributed algorithm is presented that can be used to solve the single-destination shortest path (SDSP) problem or the all-pairs shortest path (APSP) problem for a class of clustered data networks. The network graph is assumed to be characterized with a balanced hierarchically clustered (BHC) topology. The BHC topology is introduced in this paper and is shown to be a realistic characterization for a large class of interconnected data networks. For certain types of BHC topologies, the SDSP problem can be solved with computation and communication time complexities of $O(\log n)$, assuming one processor is available at each of the n number of nodes. Assuming p processors are available at each node, computation and communication time complexities of $O((n/p)\log n)$ and $O(n \log n)$ are achievable, respectively, for solving the APSP problem. It is also shown that the algorithm converges in an asynchronous environment. Therefore, some of the difficulties associated with synchronizing the order of events can be avoided in the actual implementation of the proposed algorithm.**

Index Terms—**Asynchronous computation, data networks, distributed computation, dynamic programming, hierarchical network topologies, hierarchical routing algorithms, optimal routing, parallel computation, shortest path algorithms, time complexity.**

I. Introduction and Background

SOLVING shortest path problems is the backbone, and in many cases the computational bottleneck, for a large class of congestion control algorithms. In particular, routing algorithms typically rely on shortest path information to determine efficient routes on which to send packets through a data network [4].

For the purposes of this paper, a data network will be modeled as a connected and directed graph where each link is assigned a nonnegative length—the length of a link is typically a measure of the congestion level associated with the link. The shortest path problem involves finding a path of minimum total length that joins two given nodes of a network. We will address two classes of shortest path problems: 1) the single-destination shortest path (SDSP) problem, in which the objective is to find shortest paths from every node to a single destination node and 2) the all-pairs shortest path (APSP) problem, where the goal is to find shortest paths between all pairs of nodes. Because the APSP problem involves solving n versions of the SDSP problem (in parallel), we will focus our attention primarily on the SDSP problem.

The shortest path problem has been studied extensively in the past. Dijkstra's algorithm [6] and the Bellman–Ford algorithm [7] are two classical algorithms for solving shortest path problems. Most of the other algorithms in the literature are variations of either the Dijkstra or Bellman–Ford algorithm [11]. The computation time complexity for the serial version of Dijkstra's algorithm to solve the SDSP problem for an n-node graph is $O(n^2)$. The corresponding complexity for the serial Bellman-Ford algorithm is $O(n^3)$.

The Dijkstra-type algorithms tend to be inherently serial and are not, therefore, easily implementable as distributed algorithms in large data networks. We note that a parallel version of Moore's algorithm, which is a variant of Dijkstra's algorithm, was proposed and tested by Deo, Pang, and Lord [5]. However, this algorithm was designed to operate on a tightly coupled multiprocessor network. The tightly coupled assumption is obviously not valid in large data networks which span vast amounts of geographic area. Thus, a distributed version of this type of parallel algorithm is not an obvious extension.

The Bellman–Ford-type algorithms, on the other hand, can be implemented in a distributed data network. For instance, the original ARPANET routing algorithm [9] is based on a distributed version of the Bellman–Ford algorithm. Assuming one processor is available at each node in a general n-node network, it can be shown that the distributed Bellman–Ford algorithm can solve the SDSP problem with computation and communication time complexities of $O(\bar{d}n)$ and $O(n)$, respectively, where $\bar{d}$ denotes the maximum node degree.[1] By assuming $p(\leq n)$ processors are available at each node, the APSP problem is solved with computation and communication complexities of $O((\bar{d}/p)n^2)$ and $O(n^2)$, respectively. The distributed Bellman–Ford algorithm is apparently the fastest known distributed algorithm for solving the SDSP and APSP problems for general network topologies.

One interesting fact associated with seemingly all of the classical shortest path algorithms is that the assumed network topology is general. A logical question arises concerning the existence of even faster algorithms designed specifically for networks which possess certain structures in their topology. In the following section we define a class of hierarchical

Manuscript received March 23, 1990; revised October 6, 1991. G. M. Huang is supported in part by Texas Advanced Research Program no. 4659 and NSF Grant ECS-8900499. W. K. Tsai is supported in part by NSF Grant ECS-8910328.

J. K. Antonio is with the School of Electrical Engineering, Purdue University, West Lafayette, IN 47907-1285.

G. M. Huang is with the Department of Electrical Engineering, Texas A&M University, College Station, TX 77843-3128.

W. K. Tsai is with the Department of Electrical Engineering, University of California, Irvine, Irvine, CA 92717.

IEEE Log Number 9105492.

[1] The degree of a node is defined as the number of incident links at that node.

topologies, called the balanced hierarchically clustered (BHC) topology, for which we later develop a new algorithm that can be used to solve both the SDSP and APSP problems. The BHC topology is based on the idea of hierarchically clustering large data networks according to a complete balanced-tree.[2] Assuming one processor per node, computation and communication time complexities of $O(\log n)$ are achievable when solving the SDSP problem. For the case of solving the APSP problem, the computation and communication complexities become $O((n/p)\log n)$ and $O(n \log n)$, assuming $p \leq n$ processors are available at each node.

At this point we briefly comment on the asymptotic notation used to describe complexities in this paper. By way of illustration, let $T(n)$ denote a generic time complexity and let $f(n)$ be a positive monotone-nondecreasing function. The equation $T(n) = O[f(n)]$ implies the existence of constants $\bar{\kappa} > 0$ and n_o for which the equation $T(n) \leq \bar{\kappa} f(n)$ is satisfied for all $n \geq n_o$. The notation $T(n) = \Omega[f(n)]$ means that there exists constants $\underline{\kappa} > 0$ and n_o such that $T(n) \geq \underline{\kappa} f(n)$ for all $n \geq n_o$. Finally, if $T(n) = O[f(n)]$ and $T(n) = \Omega[f(n)]$, then we write $T(n) = \Theta[f(n)]$. Now, suppose we have two competing algorithms A_1 and A_2 with time complexities $T_1(n) = O[f_1(n)]$ and $T_2(n) = O[f_2(n)]$, respectively. Based on the definition of the $O(\cdot)$ operator, we note that assuming $f_1(n) < f_2(n)$ for all $n \geq 1$ does not necessarily imply $T_1(n) < T_2(n)$ for all n. However, if we have that $T_1(n) = O[f_1(n)]$ and $T_2(n) = \Theta[f_2(n)]$, then $f_1(n) < f_2(n)$ does imply that $T_1(n) < T_2(n)$ for sufficiently large n.

In order to show superiority of our new algorithm over the standard Bellman–Ford algorithm, we show that for a large subclass of BHC topologies the $O(\cdot)$ operator can be replaced with $\Theta(\cdot)$ for the Bellman–Ford time complexities mentioned earlier. So, we show that the time complexities of our new algorithm are in fact *generically* better than those associated with the standard Bellman–Ford algorithm.

In order to illustrate the spirit of our new algorithm, consider the problem of solving a SDSP problem for the simple 2-level hierarchically clustered topology shown in Fig. 1. By blindly employing the distributed Bellman–Ford algorithm, the problem can be solved with a computational time complexity of $O(\bar{d}n)$, where $\bar{d}$ denotes the maximum node degree, and n is the total number of nodes in the network. Now, if we assume that each cluster contains $O(n/b)$ nodes, and no more than say $\bar{g}$ gates, i.e., the nodes connecting each cluster to the rest of the network, then our proposed algorithm operates as follows. First, shortest path subproblems are solved—in parallel at each cluster—from the nodes within each cluster to each of their associated gates; then the solutions from all the clusters are merged together to obtain the complete solution for the entire network. The parallel solving of the cluster subproblems requires at most $O((\bar{d}\bar{g}/b)n)$ time, while the merging takes no more than $O[\bar{g}(\bar{d}b + 1)]$ time. So, the overall time complexity of our new algorithm is $O[(\bar{d}\bar{g}/b)n + \bar{g}(\bar{d}b + 1)]$. Now, if we assume that b, $\bar{g}$, and $\bar{d}$ are all independent of n, which is often a reasonable assumption, then we note that both the distributed Bellman–Ford algorithm and our new algorithm have computational time complexities of $O(n)$. However, if $\bar{g}/b < 1$ is small enough, then the actual amount of time required for our algorithm is less than that associated with the standard Bellman–Ford algorithm. The savings in computation time for this simple 2-level example is in the order of $b/\bar{g}$ folds. This basic idea is applied recursively to general multilevel hierarchically clustered networks to achieve a time complexity of $O(\bar{g}^3 \log n)$, see Section III. Thus, if the value of $\bar{g}$ is independent of the size of the network, then we see that a time complexity of $O(\log n)$ is achievable. Actually, even if the value of $\bar{g}$ increases with a sufficiently slow function of n, then the new algorithm can still out-perform the standard Bellman–Ford algorithm.

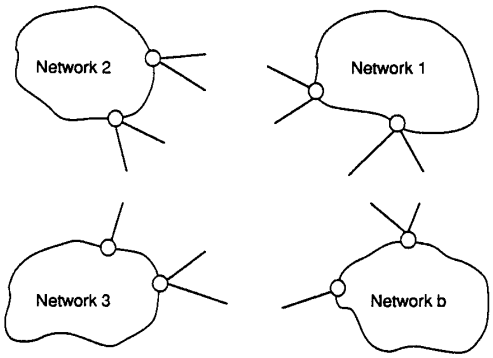

Fig. 1. An example 2-level hierarchically clustered network which can achieve $b/\bar{g}$ speedup.

The remainder of the paper is organized as follows. In Section II we define the BHC topology and point out that many large network topologies belong to this class. In Section III we develop the proposed distributed algorithm, which is designed specifically for data networks possessing a BHC topology. In Section IV we show that in general, the assumption of a BHC topology alone does not improve the time complexities associated with the standard distributed Bellman–Ford algorithm. Section V gives a modification for our new algorithm and proves that the modified version will converge asynchronously.

II. The Balanced Hierarchically Clustered Topology

In extremely large networks, the topology often possesses a natural hierarchically clustered structure. Our goal here is to exploit this structure as a means of improving the time complexity associated with solving shortest path problems.

In studying network problems/algorithms, most researchers assume a general network topology. As a result, the estimates on the complexities of their algorithms tend to be pessimistic—usually there exists a worst case topology for which the complexity bound is attained. However, the worst case topology may rarely occur in practice, therefore, more optimistic complexity bounds may be achievable for realistic networks. One straightforward way to obtain better complexities is to design algorithms around a restricted class of network topologies. We next characterize such a class, which we call the BHC topology.

[2]The concept of a balanced-tree itself was introduced originally in [17] and [18].

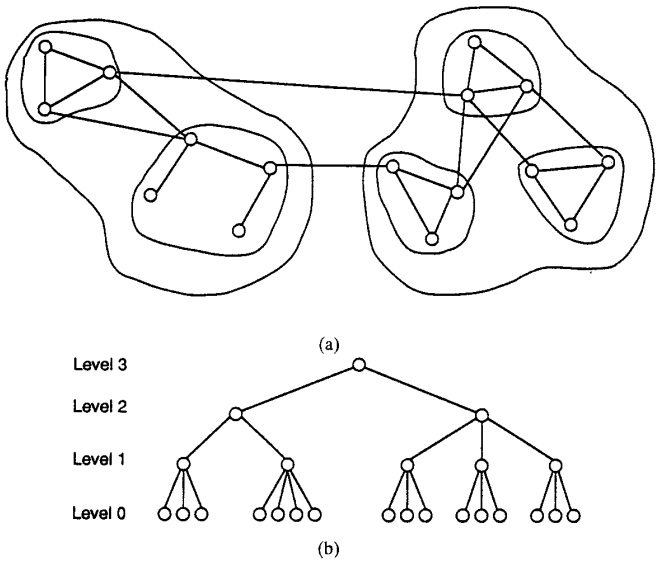

Fig. 2. (a) An example of a three-level BHC topology. (b) The balanced-tree which conceptualizes the hierarchically clustered nature of the network in (a).

A. Defining The BHC Topology

The organizational structure of the BHC topology can be characterized conceptually with a balanced-tree [1] hierarchy.[3] Thus, to assist us in describing the BHC topology, we first give a definition for a generalized $(\underline{b}, \bar{b})$ balanced-tree.

Definition 2.1: A $(\underline{b}, \bar{b})$ balanced-tree is defined as a complete tree[4] in which the number of children for each parent is lower bounded by $\underline{b}$ and upper bounded by $\bar{b}$, where $\underline{b}$ and $\bar{b}$ are independent of n, i.e., the total number leaves on the tree, and $\underline{b} \geq 2$.

The nodes of a BHC topology can be hierarchically clustered according to a $(\underline{b}, \bar{b})$ balanced-tree in which each leaf on the tree represents a node, each parent on the tree represents a cluster which contains a group of connected children, where the children may represent either clusters or nodes.

The BHC topology characterizes networks that are formed by recursively interconnecting existing smaller networks. Fig. 2(a) shows an example of a network that is clustered as a three-level BHC topology. The clusters have been circled for clarity. In Fig. 2(b), the hierarchically clustered nature of the network is conceptualized with an associated complete balanced-tree.

In a general k-level BHC topology, the clusters that have $(j-1)$-level clusters as members (i.e., children) are referred to as j-level clusters, for each $j \in \{1, 2, \cdots, k\}$. The actual network nodes are therefore defined as 0-level clusters. So in a k-level BHC topology the total number of nodes in the network n is bounded below by $(\underline{b})^k$ and above by $(\bar{b})^k$, thus we have that $k \leq \log_{\underline{b}} n$. Note that each j-level cluster (contained in a given k-level BHC topology) is itself a j-level BHC topology, for all $j \in \{1, 2, \cdots, k\}$.

[3]The balanced-tree was originally introduced to describe a class of hierarchical data structures, see for example [1].

[4]By a complete tree, we mean a rooted tree in which all leaves are of the same depth.

In terms of the above terminology, a k-level BHC topology can be created by interconnecting an $O(1)$ collection of distinct $(k-1)$-level BHC topologies so as to create a connected graph. By recursively applying this rule, a k-level BHC topology can be constructed for any integer $k \geq 1$.

Next we introduce precise notation for characterizing the member nodes of a generic j-level BHC topology.

Definition 2.2: Given a collection of j-level BHC topologies, let N_j^i denote the node set of the ith j-level BHC topology.

A key concept in establishing the complexity result of the new algorithm is based on the fact that all inter-cluster traffic must pass through gates. Next, we define the notion of gate sets and gates for a BHC topology.

Definition 2.3: The ith j-level gate set, denoted G_j^i, is defined as the subset of nodes taken from N_j^i which are directly connected to nodes in some other j-level node set N_j^p, where N_j^i and N_j^p belong to a common $(j+1)$-level node set, and $p \neq i$. The elements of G_j^i are called j-level gates.

In order to clearly illustrate the definitions of node sets, gate sets, and gates; consider the example three-level BHC topology given in Fig. 3. The node sets for this example are: $N_1^1 = \{1, \cdots, 7\}$, $N_1^2 = \{8, \cdots, 13\}$, $N_1^3 = \{14, \cdots, 17\}$, $N_1^4 = \{18, \cdots, 24\}$, $N_1^5 = \{25, \cdots, 33\}$, $N_1^6 = \{34, \cdots, 41\}$, $N_1^7 = \{42, \cdots, 47\}$, $N_2^1 = N_1^1 \cup N_1^2$, $N_2^2 = N_1^3 \cup N_1^4$, $N_2^3 = N_1^5 \cup N_1^6 \cup N_1^7$, and $N_3^1 = N_2^1 \cup N_2^2 \cup N_2^3$. The gate sets are given by: $G_1^1 = \{2, 3, 7\}$, $G_1^2 = \{8, 12, 13\}$, $G_1^3 = \{16, 17\}$, $G_1^4 = \{18, 23\}$, $G_1^5 = \{27, 31, 32, 33\}$, $G_1^6 = \{34, 35\}$, $G_1^7 = \{42, 44\}$, $G_2^1 = \{7, 13\}$, $G_2^2 = \{16, 23\}$, $G_2^3 = \{27, 34, 44\}$.

We now introduce more definitions that further parameterize the BHC topology. The next two definitions are counts for the number of children and grandchildren belonging to a given cluster.

Definition 2.4: Given a collection of j-level BHC topolo-

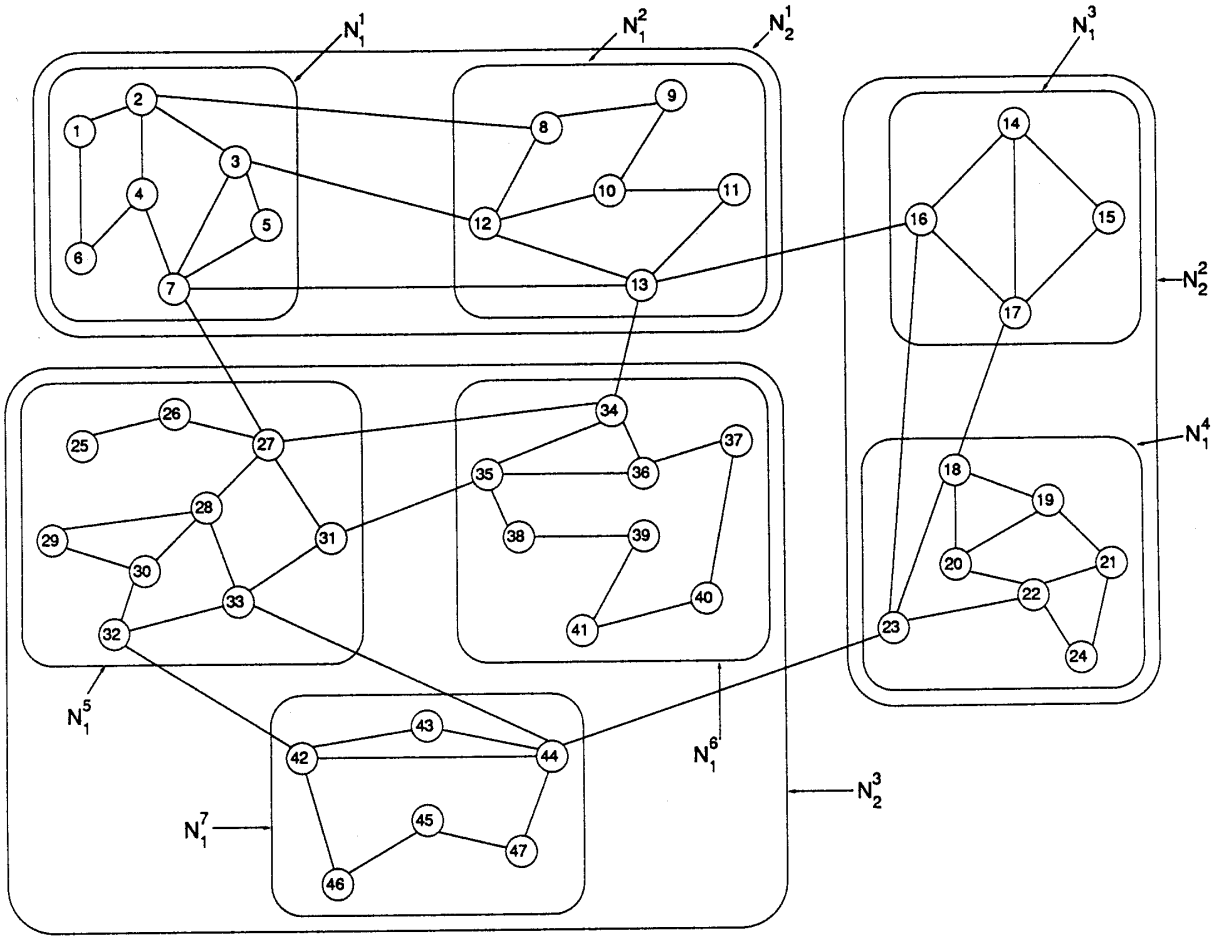

Fig. 3. An example of a 3-level BHC topology.

gies, let b_j^i denote the number of $(j-1)$-level BHC topologies contained in the ith j-level BHC topology.

Definition 2.5: Let b_j denote the total number of $(j-1)$-level BHC topologies contained in a given k-level BHC topology. So, by the previous definition,

$$b_j = \sum_{i=1}^{b_{j+1}} b_j^i, \quad \text{for all } j \in \{1, \cdots, k\}$$

where

$$b_{k+1} \triangleq 1$$

and b_1 equals n (the total number of nodes in the network).

Next, we define sets which index the set of the $(j-1)$-level clusters that belong to a common parent j-level cluster.

Definition 2.6: Let M_j^q denote the qth j-level cluster member set, which is defined to be the index set which satisfies

$$N_j^q = \cup_{i \in M_j^q} N_{j-1}^i, \quad \text{for all } j \in \{1, \cdots, k\},\ q \in \{1, \cdots, b_{j+1}\}$$

where

$$N_0^i \triangleq \{i\}, \quad \text{for all } i \in \{1, \cdots, b_1\}.$$

The next parameter denotes the maximum number of gates contained in any single gate set. As we shall see later, the time complexity of our algorithm depends directly on the value of this important parameter.

Definition 2.7: For a given k-level BHC topology, let $\bar{g}$ denote the maximum number of elements contained in all sets G_j^i. That is, $\bar{g}$ is the integer which satisfies the following:

$$\bar{g} = \max\{|G_j^i|\}, \quad \text{for all } j \in \{1, \cdots, k-1\},\ i \in \{1, \cdots, b_{j+1}\}$$

where $|G_j^i|$ denotes the number of elements in G_j^i. (Note that it is necessary to have $\bar{g} \geq 1$ in order for a BHC topology to be connected.)

To illustrate Definitions 2.4 through 2.7, we again refer to the three-level BHC topology depicted in Fig. 3. (The node and gate sets were given previously.) By Definition 2.4, we have the following values: $b_1^1 = 7$, $b_1^2 = 6$, $b_1^3 = 4$, $b_1^4 = 7$, $b_1^5 = 9$, $b_1^6 = 8$, $b_1^7 = 6$, $b_2^1 = 2$, $b_2^2 = 2$, $b_2^3 = 3$, and $b_3^1 = 3$. From Definition 2.5 we have: $b_1 = 47$, $b_2 = 7$, $b_3 = 3$, and $b_4 = 1$. From Definition 2.6, we get the following cluster member sets: $M_1^i = N_1^i$, for all $i \in \{1, \cdots, 7\}$, $M_2^1 = \{1, 2\}$, $M_2^2 = \{3, 4\}$,

$M_2^3 = \{5, 6, 7\}$, and $M_3^1 = \{1, 2, 3\}$. By using the gate sets defined earlier, we have by Definition 2.7 that $\bar{g} = 4$.

B. The Diameter of the BHC Topology

Definition 2.8: Given a connected network with a node set $\mathcal{N}$, let $MHD(u, v)$ denote the minimum hop distance between nodes u and v, for any $u, v \in \mathcal{N}$.

Definition 2.9: Given a connected network with a node set $\mathcal{N}$, the diameter of the network, denoted as D, is defined by

$$D = \max_{u,v \in \mathcal{N}} \{MHD(u, v)\}.$$

During the operation of the new algorithm, control messages must be broadcast from each j-level gate to the member nodes of the associated j-level cluster. The maximum time required for such a broadcast is clearly dependent on the diameter of the network. If the diameter is too large, then this communication overhead begins to increase the overall time complexity of the algorithm. The following gate set conditions characterize a subclass of BHC topologies for which the diameter is small enough to avoid excessive communication complexity.

Gate Set Conditions:

C1) $G_j^q \subset \cup_{i \in M_j^q} G_{j-1}^i$, for all $j \in \{1, \cdots, k-1\}, q \in \{1, \cdots, b_{j+1}\}$

C2) The subgraph consisting only of nodes contained in G_j^i is connected, for each $j \in \{1, \cdots, k-1\}, i \in \{1, \cdots, b_{j+1}\}$.

Simply put, condition C1 requires each j-level gate to also serve as a $(j-1)$-level gate. Condition C2 states that the subgraph of gates contained in each gate set be connected. These two gate set conditions greatly simplify the algorithm description and the analysis of the algorithm's time complexity. We should note, however, that our algorithm may still out-perform the Bellman–Ford algorithm even without these conditions being satisfied—more on this in the next section.

We are now ready to state a theorem that bounds the diameter of a BHC topology (assuming the gates set conditions are satisfied). First a preliminary definition.

Definition 2.10: Consider two nodes u and v, which belong to a common k-level BHC topology. The least common level between u and v, denoted $LCL(u, v)$, is defined to be the level number of the lowest level BHC topology (within the common k-level BHC topology) to which both u and v belong.

Now we state the main theorem.

Theorem 2.1: If gate set conditions C1 and C2 are satisfied in a k-level BHC topology, then the diameter of the network is $O(\bar{g} \log n)$.

Proof: Note that in a k-level BHC topology, $LCL(u, v) \in \{1, 2, \cdots, k\}$, for all nodes u and v. First show that if $LCL(u, v) = k$, then $MHD(u, v) = O(\bar{g} \log n)$. Second, show that for any other pair of nodes u' and v', if $LCL(u', v') < k$, then $MHD(u', v') = O(\bar{g} \log n)$.

Assume $LCL(u, v) = k$. The minimum hop distance path from u to v must generically go from node u to a 1-level gate to a 2-level gate $\cdots$ to a k-level gate to a $(k-1)$-level gate to a $(k-2)$-level gate $\cdots$ to a 1-level gate to node v. The minimum hop distance from node u to each of its parent 1-level gates is obviously bounded by $\bar{b}$. Also, by using conditions C1 and C2 it is clear that the minimum hop distance from a j-level gate to a parent $(j+1)$-level gate is bounded by $\bar{g}\bar{b}$. Using these bounds recursively, we can therefore bound the $MHD(u, v)$ as follows:

$$\begin{aligned} MHD(u, v) &\le \underbrace{\bar{b} + \bar{g}\bar{b} + \cdots + \bar{g}\bar{b}}_{k \text{ terms}} + \underbrace{\bar{g}\bar{b} + \cdots + \bar{g}\bar{b} + \bar{b}}_{k \text{ terms}} \\ &\le 2\bar{g}\bar{b}k \quad (\text{since } \bar{g} \ge 1) \\ &= 2\bar{g}\bar{b} \log_{\underline{b}}(\underline{b}^k) \\ &\le 2\bar{g}\bar{b} \log_{\underline{b}}(n) \quad (\text{since } \underline{b}^k \le n) \\ &= O(\bar{g} \log n) \quad (\text{since } \bar{b} \text{ and } \underline{b} \text{ are independent of } n). \end{aligned}$$

Now for $LCL(u', v') = m < k$, we have

$$MHD(u', v') \le \underbrace{\bar{b} + \bar{g}\bar{b} + \cdots + \bar{g}\bar{b}}_{m \text{ terms}} + \underbrace{\bar{g}\bar{b} + \cdots + \bar{g}\bar{b} + \bar{b}}_{m \text{ terms}},$$

which is less than the bound for $MHD(u, v)$. Q.E.D.

Note from the above theorem that if we can assume $\bar{g} = O(1)$, then the diameter of the BHC topology becomes $O(\log n)$.

C. Characterizing Realistic Data Networks as BHC Topologies

We now turn our attention toward justifying the assumption that many existing data networks can be characterized as having a BHC topology. Perhaps the most straightforward way to show that a given graph has a BHC topology is to exhaustively check many different node set descriptions until one is found that satisfies the definition of a BHC topology. However, such an approach is attempting to solve an NP-complete problem [17]; therefore, heuristic clustering algorithms are a viable alternative. We should stress that a heuristic multilevel clustering procedure does not have to be "completed" in order to reap the benefit of the proposed shortest path algorithm. For instance, even if such an algorithm stops with a 2-level clusterization, the speedup can be substantial, as shown by the example provided in the introductory section.

In many cases the natural hierarchically clustered structure of a large data network is obvious by simply looking at a logical layout of the network graph. The basic premise that large data networks exhibit some sort of hierarchical structure has been confirmed by other researchers. For example, in [8] a method is introduced for interconnecting existing networks, via gateways, to form larger networks. Thus, nodes belonging to a geographical area or a particular existing subnetwork form natural clusters. The concept of area routing is also discussed, whereby routing within an area is managed separately from routing between areas. (We utilize this same basic idea in the development of the proposed shortest path algorithm presented in the next section.) Also, in [2], the use of multilevel gateways is shown to be an effective way of interconnecting similar (or dissimilar) networks. In [10], several approaches to interconnecting local networks to long-distance networks, are discussed. Overall, the network descriptions assumed or developed in [2], [8], and [10] have obvious hierarchical

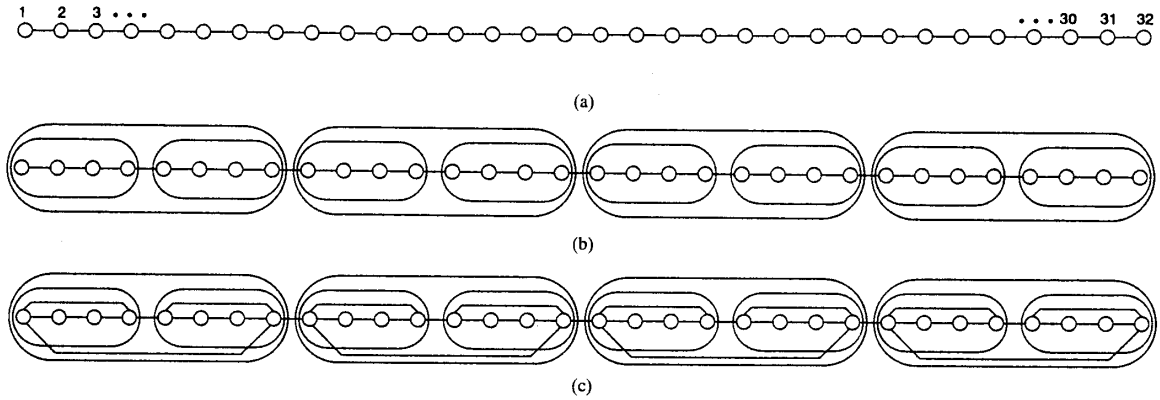

Fig. 4. (a) A 32 node "linear" network. (b) Viewing the linear network as a 3-level BHC topology. (c) By adding some links to the linear network, a 3-level BHC topology is produced which satisfies the gate set conditions.

structures which coincide with the structure of the BHC topology.

In certain (existing) networks, the addition and/or deletion of a few links may result in a network that has a BHC topology. To illustrate this point, consider the network shown in Fig. 4(a), which depicts a simple "linear" network. Define the 1-level node and gate sets for this network as follows: $N_1^1 = \{1,2,3,4\}$, $N_1^2 = \{5,6,7,8\}$, $N_1^3 = \{9,10,11,12\}$, $N_1^4 = \{13,14,15,16\}$, $N_1^5 = \{17,18,19,20\}$, $N_1^6 = \{21,22,23,24\}$, $N_1^7 = \{25,26,27,28\}$, $N_1^8 = \{29,30,31,32\}$, $G_1^1 = \{4\}$, $G_1^2 = \{5\}$, $G_1^3 = \{12\}$, $G_1^4 = \{13\}$, $G_1^5 = \{20\}$, $G_1^6 = \{21\}$, $G_1^7 = \{28\}$, $G_1^8 = \{29\}$. The 2-level sets are then given by $N_2^1 = N_1^1 \cup N_1^2$, $N_2^2 = N_1^3 \cup N_1^4$, $N_2^3 = N_1^5 \cup N_1^6$, $N_2^4 = N_1^7 \cup N_1^8$, $G_2^1 = \{8\}$, $G_2^2 = \{9,16\}$, $G_2^3 = \{17,24\}$, $G_2^4 = \{25\}$. Finally, we have $N_3^1 = \cup_{i=1}^4 N_2^i$. In Fig. 4(b) the clusters of the linear network are circled to emphasize its "hierarchical structure" described above. It is easy to verify that condition C2 of the previously stated gate set conditions is *not* satisfied for the clusterization of the linear network. In Fig. 4(c), we show that by adding some links to the linear network, a new network is formed in which the gate set conditions are satisfied. For this modified network, we assume the same node sets as used for the original linear network. The new gate sets for the network of Fig. 4(c) become: $G_1^1 = \{1,4\}$, $G_1^2 = \{5,8\}$, $G_1^3 = \{9,12\}$, $G_1^4 = \{13,16\}$, $G_1^5 = \{17,20\}$, $G_1^6 = \{21,24\}$, $G_1^7 = \{25,28\}$, $G_1^8 = \{29,32\}$, $G_2^1 = \{8\}$, $G_2^2 = \{9,16\}$, $G_2^3 = \{17,24\}$, $G_2^4 = \{25\}$. By using the above gate set description, it is easy to verify that both gate set conditions C1 and C2 are satisfied.

It should be noted that there are actually many valid choices for the node and gate sets of the network shown in Fig. 4(a). For example, by using the same N_1^i's and G_1^i's as defined for Fig. 4(b), we could consider the entire network to be only a 2-level BHC topology, where we define $N_2^1 = \cup_{i=1}^8 N_1^i$. This clearly implies that, in general, a given network may have many different possible BHC topology descriptions.

The clustering problem has been studied extensively in the area of VLSI layout. The classic results in divide and conquer layout algorithms are due independently to Valiant [19], Leiserson [20], and Floyd and Ullman [21]. A nice overview of this material is found in Ullman [22, pp. 91–102]. The classical divide and conquer layout algorithms for VLSI are based on the concept of graph separators. Roughly speaking, an n-node graph is said to be $S(n)$-*separable* if a set of at most $S(n)$ edges can be found whose removal disconnects the graph into two subgraphs, neither having more than twice the number of nodes of the other. So, combining this terminology with our definition of $\overline{g}$ (see Definition 2.7), we note that the class of BHC topologies are essentially $\overline{g}$-separable graphs. We should note that most of the "efficient" algorithms for finding graph separators assume that the graph is planar and has a maximum node degree of four. While such assumptions are certainly realistic in many VLSI applications, they clearly may not be valid when considering large and complicated data networks. For a more complete description of general graph separators, the interested reader is referred to Leighton [23].

III. A New Distributed Synchronous Shortest Path Algorithm for BHC Topologies

The main idea of the proposed algorithm is twofold. First, shortest path problems are solved for a collection of small disjoint clusters. Because there are no common nodes between these clusters, the entire collection of problems are solved in parallel. Second, the solution of the clusters are recursively merged together until the solution for the entire network is acquired.

We should point out that the algorithm described in this section requires a certain amount of synchronicity for its operation. In particular, it is assumed that the iterations of the main step are done in a particular order, i.e., the jth iteration must be completed before the $(j+1)$th iteration may begin. By knowing the bounds for communication and computation delays, an upper bound can be found for the maximum amount of time required for any iteration. Thus, a straightforward synchronization scheme can be used. (A uniform bound is assumed for computation and communication delays for the synchronous algorithms presented in this and the next section.)

A mechanism is, of course, needed for getting all the nodes to agree to start the algorithm. However, in this section we

will not discuss this initialization problem for several reasons. First, a straightforward initialization algorithm is possible due to our assumption on uniformly bounded delays. Also, the fact that the minimum hop distance between any pair of nodes is $O(\bar{g} \log n)$ implies that such a procedure should not increase the overall time complexity. Finally, the start-up problem can be eliminated by using the asynchronous version of the algorithm derived in Section V.

A. Network Assumptions

A1) The network has a k-level BHC topology (with $n \leq (\bar{b})^k$ nodes) and gate set conditions C1 and C2 are satisfied.

A2) Each node has at least one processor.

A3) Each node has a unique identification (ID) number taken from the set of integers $\{1, \cdots, n\}$. (For convenience, we assume that the given destination node is number 1.)

A4) Each node u has available the link lengths ℓ_{uw}, for all $w \in N(u)$, which are assumed positive and constant after some initial time t_o, where $N(u)$ denotes the neighbors of node u.

A5) Each node u has sufficient memory to store two $(1+\bar{g}k)$-length vectors d_u and e_u. The first component of d_u, denoted d_{u1}, is the current estimate of the shortest distance from node u to the given destination node. The remaining gk components of d_u, denoted d_{uv} for the vth component, are used to store estimates of the distances from node u to each of its (at most) $\bar{g}k$ logical parent gates (a physical node can be the site of multiple logical gates). The vth component of e_u, denoted e_{uv}, is the ID of the next node (from node u) along the current estimate of the shortest path associated with d_{uv}.

A6) Besides the memory required by A5, each logical gate and each regular node has extra memory allocated to store an additional $\bar{g}$ d_u vectors.

A7) The processing, transmission, queueing, and propagation delays, associated with sending a single $O(1)$ length control message across any link in the network, are assumed independent of n.

Assumptions A1 through A6 will prove useful primarily in deriving the computational time complexity component of the proposed algorithm. With the exception of A1, these first seven assumptions are fairly standard for most distributed shortest path algorithms.

Assumption A7 is related to the communication complexity involved with sending control messages in the network. It is reasonable to assume that the processing and transmission delay (associated with sending an $O(1)$ length control message) is independent of n, because the length of each message is independent of n. Also in A7, the assumption of the queueing delay being independent of n is justified by simply allowing the control messages to have the highest possible priority at all queues. Finally in A7, note that the propagation delay associated with most data network links is considered negligible and therefore independent of n.

B. Description of the New Distributed Shortest Path Algorithm

In this subsection, we first give a general description of the new shortest path algorithm, which has the intent of showing only the most fundamental tasks performed. Later, a detailed description is given which shows how to implement each step. Of prime importance is step 2. In particular, we will show how to "merge" shortest path information. The merging procedure is not obvious and our implementation of this task is the foundation of the entire algorithm.

In the following general description of the algorithm, note that steps 0 and 1 are executed once and step 2 is done recursively $k-1$ times.

General Description:

0) Each component[5] of d_u is set to ∞ and each component of e_u is set to zero for all $u = 1, \cdots, n$.

1) Each of the b_2 1-level BHC topologies solves at most $\bar{g}+1$ versions of the single destination shortest path problem—one version for the given destination node (if applicable) and at most $\bar{g}$ versions for each of the 1-level gates for each 1-level BHC topology (i.e., cluster). So after this step, each node has an estimate (possibly noninfinite) of the shortest distance to the given destination node and an estimate of the shortest distance to each associated 1-level gate.

DO $j = 1, \ k-1$.

2) Parallely merge shortest path information of the b_{j+1} j-level BHC topologies into shortest path information for b_{j+2} $(j+1)$-level BHC topologies.

END DO

Next, we give the detailed description of each step in the algorithm, along with the time complexities associated with these steps. Time complexities are measured by using two separate time units—one for computation time complexity, the other for communication time complexity. In the detailed description, step 2 is divided into four "substeps." Furthermore, some of these substeps rely on a standard distributed version of the Bellman–Ford algorithm to solve shortest path problems for small clusters or groups of connected nodes. We use the fact that the distributed Bellman–Ford algorithm has generic computation and communication time complexities of n^2 and n, respectively, to estimate the time complexities for the applicable substeps.

Detailed Description (with complexity computation in each step):

0) Initialize the vectors d_u and e_u. This can be done in $1+\bar{g}k \leq 1+\bar{g}\log_{\bar{b}} n$ computation time units.

1) At this point, the network is divided into b_2 1-level BHC topologies (or clusters) containing at most $\bar{b}$ nodes each. Every cluster solves the single destination shortest path problem for the given destination node (if the destination node happens to belong to their cluster). Next, each cluster solves the single destination shortest path problem for each gate in G_1^i, for all $i \in \{1, \cdots, b_2\}$. These tasks are accomplished using the standard dis-

[5] Except for the first components of d_1 and e_1 which are set to 0 and 1, respectively.

tributed Bellman–Ford algorithm. Because each node is a member of only one cluster, the b_2 clusters can solve their shortest path problems in parallel. Because each cluster contains at most $\bar{b}$ nodes, this step will require at most $\bar{b}^2(\bar{g}+1)$ computation time units and $\bar{b}(\bar{g}+1)$ communication time units.

DO $j = 1,\ k-1$.

2.1) All pairs of gates $g_a, g_b \in G_j^i$, replace the value of $\ell_{g_a g_b}$ with the current estimate of the shortest distance between g_a and g_b, for each $i \in \{1, \cdots, b_{j+1}\}$. (Note: for those pairs of gates in G_j^i which are not directly connected, a *virtual* link length is stored.) Thus, each gate must update at most $\bar{g}$ values of $\ell_{g_a g_b}$, which requires no more than $\bar{g}$ computation time units.

2.2) The gates in $\cup_{i \in M_{j+1}^q} G_j^i$ are considered as connected subgraphs, for each $q \in \{1, \cdots, b_{j+2}\}$. The single destination problem is solved from every gate in $\cup_{i \in M_{j+1}^q} G_j^i$ to the given destination node by using a slightly modified distributed Bellman–Ford algorithm.[6] The current estimates from each gate to the destination node (from the previous iteration) are used as a virtual link length to the given destination node. Next, the single destination problem is solved for each $(j+1)$-level gate in $\cup_{i \in M_{j+1}^q} G_j^i$, again using a modified distributed Bellman–Ford algorithm. Note that there are b_{j+2} of these subgraphs and since they are disjoint, the computation is done in parallel. Also, since there are at most $\bar{b}\bar{g}$ nodes in each $\cup_{i \in M_{j+1}^q} G_j^i$, the shortest path computation for the given destination node and all parent $(j+1)$-level gates requires at most $(\bar{g}+1)(\bar{b}\bar{g})^2$ computation time units and $(\bar{g}+1)\bar{b}\bar{g}$ communication time units. So, at this point we have the following situation: i) Every regular node has an estimate of the shortest distance to the given destination node and to each associated j-level gate. ii) Every gate in the set $\cup_{i \in M_{j+1}^q} G_j^i$ has a new estimate of the shortest path distance to the given destination node and to every other $(j+1)$-level gate belonging to $\cup_{i \in M_{j+1}^q} G_j^i$, for all $q \in \{1, \cdots, b_{j+2}\}$.

2.3) Every gate in G_j^i transmits its new estimates (of the distance from itself to the given destination node and the distance from itself to each $(j+1)$-level parent gate), to every other node belonging to the same j-level BHC topology, for all $i \in \{1, \cdots, b_{j+1}\}$. This broadcast can be done in parallel for each i, with at most $2\bar{b}\bar{g}\log_{\underline{b}} n$ hops, see proof of Theorem 2.1. Thus, the broadcast of at most $\bar{g}+1$ such messages requires no more than $2\bar{b}\bar{g}(\bar{g}+1)\log_{\underline{b}} n$ hops. Since transmitting $O(1)$ length control messages across any link in the network requires $O(1)$ communication time units, this step will require no more than $2\bar{b}\bar{g}(\bar{g}+1)\log_{\underline{b}} n$ communication time units. Note that these messages are stored in the extra memory allotted to each regular node described by A6.

2.4) Finally, every node within each $(j+1)$-level BHC topology can compute a new estimate of the shortest path distance to the given destination node, and a new estimate of the shortest path distance to each parent $(j+1)$-level gate, by adding its distance to each of its j-level gates to the estimates from each of these gates to the destination node (i.e., the given destination node or one of the $(j+1)$-level gates). Because there are at most $\bar{g}$ gates per j-level BHC topology, this requires at most $\bar{g}(\bar{g}+1)$ additions and $(\bar{g}+1)^2$ comparisons or $(\bar{g}+1)(2\bar{g}+1)$ computation time units. Note that the one extra comparison comes from the fact that a comparison must also be made with the current estimate stored in the distance vector at each origin node.

END DO

Proof of Correctness: To prove correctness of the algorithm, it suffices to show that after the jth iteration of step 2, the value of the first component of the d_u vectors at each node represent the shortest path distance to the destination node (assuming that it belongings to the same $(j+1)$-level BHC topology), with the constraint that the path includes only nodes that are members of these $(j+1)$-level BHC topologies. Thus, when $j = k-1$, the first component of these distance vectors at each node are the shortest path distances to the destination node assuming it belongs to the same k-level BHC topology, which is assumed to be the entire network.

We will now show the above assertion is true. Consider any node $u \neq 1$ with $LCL(u,1) = m$. Note that the possible values of m are contained in the set $\{1, \cdots, k\}$. Also, note that all possible paths connecting u to 1 can be classified into $k+1-m$ categories. That is, all possible paths must be contained within either a m-level BHC topology, a $(m+1)$-level BHC topology, $\cdots$, or a k-level BHC topology. Clearly, it is impossible for all the nodes associated with such paths to be contained in a BHC topology having less than m levels, since it is assumed that $LCL(u,1) = m$.

Next, we show by induction that after the jth iteration of step 2, the shortest distance between any node u and node 1 having $LCL(u,1) \leq j+1$ is obtained, with the constraint that such a path does not leave the common $(j+1)$-level BHC topology. Note that step 1 can be considered to be the result of step 2 with $j = 0$. Therefore, we can now consider any arbitrary value of j and proceed with the proof.

At substep 2.1 we have, by the induction hypothesis, that the value of d_{u1} is the shortest distance between u and 1 for all paths contained in the common j-level BHC topology. In substep 2.2, we see that every gate in $\cup_{i \in M_{j+1}^q} G_j^i$ does indeed obtain the shortest distance to the given destination node and to every other $(j+1)$-level gate in $\cup_{i \in M_{j+1}^q} G_j^i$, with paths restricted to the appropriate $(j+1)$-level BHC topology. This

[6] By modified distributed Bellman–Ford algorithm, we mean the following. The gates in each j-level gate set (i.e., G_j^i) act as if they are directly connected to every other gate in G_j^i, by using the virtual link lengths determined in substep 2.1. Clearly, this modified Bellman–Ford algorithm requires some extra communication complexity in order to simulate the gates in each G_j^i as being completely connected. Due to the fact that each pair of gates in G_j^i are at most $\bar{g}$ hops apart, it can be shown that the communication complexity associated with the modified Bellman–Ford is of the same order as the standard Bellman–Ford algorithm. Thus, we will use the standard Bellman–Ford time complexities.

is true by the fact that the *virtual* link lengths of all pairs of nodes in G_j^i were updated in 2.1. Substep 2.3 involves communication only. Substep 2.4 is obviously correct for all origin nodes having a least common level of $j+1$ between itself and the destination node. Also, for those origins having a least common level less than $j+1$, this step compares the distance of the shortest path which utilizes a $(j+1)$-level BHC topology to the current estimate of distance stored in the first component of the distance vector. Q.E.D.

Overall Time Complexity: Since step 2 is done $k-1 \leq (\log_{\underline{b}} n) - 1$ times, the overall computation and communication time complexities, denoted $T_{\text{comp}}(n)$ and $T_{\text{comm}}(n)$, respectively, are given by

$$T_{\text{comp}}(n) \leq [2\bar{g} + (\bar{g}+1)(\bar{b}^2 + \bar{b}^2\bar{g}^2 + 2\bar{g}+1)][\log_{\underline{b}} n] \quad (3.1)$$

and

$$T_{\text{comm}}(n) \leq [(\bar{g}+1)(\bar{b} + \bar{b}\bar{g} + 2\bar{b}\log_{\underline{b}} n)][\log_{\underline{b}} n]. \quad (3.2)$$

Note that because $\underline{b}$ and $\bar{b}$ are independent of n, we have

$$T_{\text{comp}}(n) = O(\bar{g}^3 \log n) \quad (3.3)$$

and

$$T_{\text{comm}}(n) = O[\bar{g}^2 (\log n)^2]. \quad (3.4)$$

In the best case, i.e., assuming $\bar{g} = O(1)$, the complexities become

$$T_{\text{comp}}(n) = O(\log n) \quad (3.5)$$

and

$$T_{\text{comm}}(n) = O[(\log n)^2]. \quad (3.6)$$

Actually, it is *not necessary* to assume $\bar{g} = O(1)$ in order to guarantee that the complexities of our algorithm are asymptotically better than those of the standard Bellman–Ford algorithm. In the next section it is shown that the complexities of the Bellman–Ford Algorithm are generically $\Theta(n)$, even when the underlying network is a BHC topology. Thus, as long as $\bar{g} = O(n^c)$, for some constant $c < 1/3$, our complexities will be better. Intuitively, this means that the structure of the network must be "more clustered," for example, than a planar square mesh, because the clustering of a mesh obviously yields $\bar{g} = O(\sqrt{n})$.

The communication complexity derived above is actually quite pessimistic. It turns out that by using a slightly more sophisticated communication scheme, we can achieve communication complexity of

$$T_{\text{comm}}(n) = O(\log n). \quad (3.7)$$

To achieve this better complexity, first note that substep 2.4–iteration j, does not need to be completed before substep 2.1–iteration $j+1$ can begin. In fact, 2.1–iteration $j+1$ may begin just as soon as 2.2–iteration j is complete. So, the modified communication scheme is summarized as follows: As soon as 2.2–iteration j is complete, then both 2.3–iteration j and 2.1–iteration $j+1$ begin, i.e., 2.1–iteration $j+1$ does not wait for the completion of 2.3– and 2.4–iteration j. Now, as soon as the data which is broadcast at 2.3–iteration j reach all the appropriate member nodes, then 2.4–iteration j begins.

Fig. 5(b) shows how the modified communication scheme achieves an overall complexity of $O(\log n)$, assuming $\bar{g} = O(1)$. (For generic values of $\bar{g}$, a similar argument yields a complexity of $\bar{g}^2 \log n$.) The original straightforward communication scheme is depicted in Fig. 5(a). Due to the fact that $1 + 2 + \cdots + k = O(k^2)$, it is easy to see why the original communication scheme requires a complexity of $O[(\log n)^2]$. On the other hand, the modified scheme achieves a communication complexity of $O(\log n)$ since the last iteration, which requires $O(\log n)$ time for completion, begins after only $O(\log n)$ time has elapsed. Note that each "block of time" in Fig. 5 represents an $O(1)$ amount of time. So, the first iteration requires four $O(1)$ blocks of time, the second iteration requires five $O(1)$ blocks of time, and so on.

C. Operation of the Algorithm Under More Relaxed Gate Set Conditions

One of the main reasons for assuming the original gate set conditions was to simplify the description of the algorithm. In this subsection we show that more relaxed gate set conditions can be accommodated by making some minor modifications to the current algorithm description.

Condition C1 may be relaxed by introducing the concept of a singleton cluster. A j-level singleton cluster is a cluster having only a single j-level gate member. We require that a j-level singleton cluster only be directly connected to related (i.e., child, sibling, or parent) $(j-1)$-, j-, or $(j+1)$-level gates. Fig. 6 shows how the BHC topology of Fig. 3 can be modified by adding singleton clusters. The operation of the algorithm on this type of modified network is virtually the same; the exception is that one extra link must be used for communication between $(j-1)$-level and parent j-level gates. One advantage of this modified network is that the j-level gates do not have to reside at the same physical node as the $(j-1)$-level gates. As a result, the high level gates need not have a degree of $O(\log n)$, as would be the case by strictly requiring C1 be satisfied.

Condition C2 may also be relaxed. For instance, assuming $O(\bar{g})$ minimum hop distances between gates within each gate set is sufficient. The modified algorithm operates by using nodes (which are not members of the gate set) and virtual links to simulate the gate set subgraph as being connected. Actually, this type of modification will work regardless of the minimum hop distance between gates (within each gate set). However, if this minimum hop distance parameter, say $\bar{h}$, becomes too large, then the communication overhead begins to slow down the algorithm. By assuming condition C2, we ensured that $\bar{h}$ is bounded by $\bar{g}$. It is straightforward to re-derive the time complexity of the algorithm based on a generic value of $\bar{h}$.

In addition to the aforementioned distributed versions of the algorithm, a parallel algorithm is also possible feasible. Such an algorithm could, for example, be used at a centralized parallel computing site having n processors that operate in a shared memory environment. By assuming shared memory,

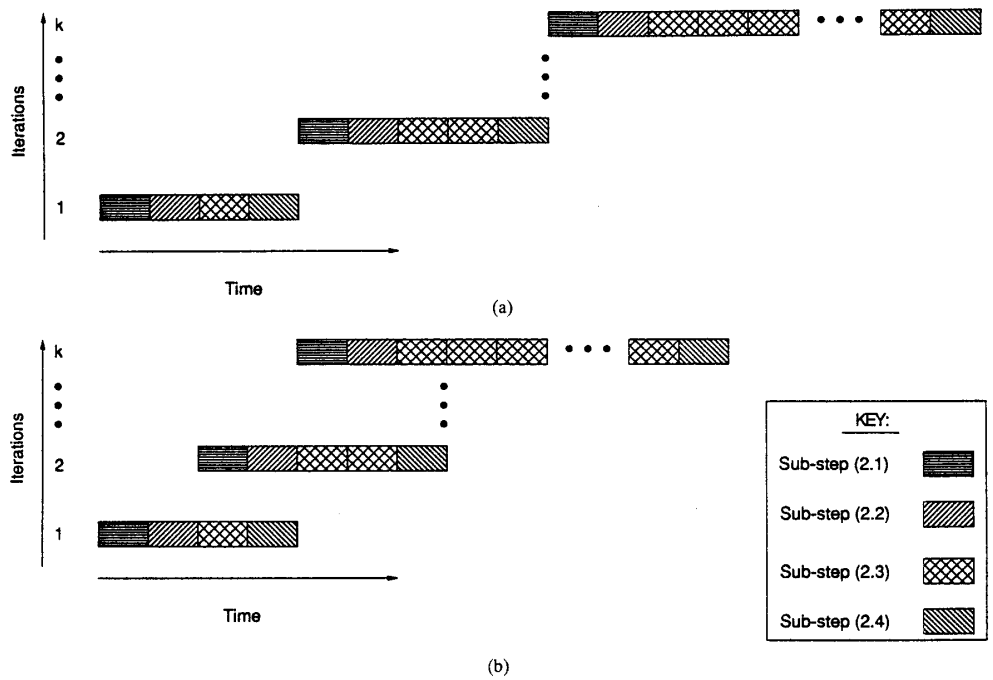

Fig. 5. (a) The straightforward communication scheme that yields an overall communication complexity of $O[(\log n)^2]$. (b) The modified communication scheme that achieves a communication complexity of $O(\log n)$.

the communication complexity associated with sending control messages and the initiation procedure can be reduced. Thus, a centralized version of the algorithm would have an overall time complexity roughly the same as the computation time complexity associated with the distributed algorithm.

D. Solving the APSP Problem

The algorithm description given previously is based on solving the SDSP problem. However, due to the fact that the algorithm consists of a collection of Bellman–Ford-type segments, solving the APSP problem is a straightforward extension. Specifically, solving the APSP problem involves solving n SDSP problems in parallel at each node. If $p \leq n$ processors are available at each node, the computation complexity becomes $O((\bar{g}^3/p)n \log n)$, while the communication complexity is given by $O(\bar{g}^2 n \log n)$. The increase in the communication complexity by a factor of n is due to the fact that n-length control messages must now be broadcast.

IV. Time Complexity of the Distributed Synchronous Bellman–Ford Algorithm—Assuming a BHC Topology

In this section we show that the time complexities associated with the standard distributed Bellman–Ford algorithm do not improve (generically) by assuming the network topology is a BHC topology.

As in the previous section, we denote the length of link (u, w) as ℓ_{uw}. The current estimate of the shortest distance from node u to the given destination node is stored at node u as d_{u1}. The Bellman–Ford algorithm is then given by

0) $d_{u1}^{(0)} = \infty, \quad u \neq 1,$
$d_{11}^{(0)} = 0.$
1) $d_{u1}^{(j+1)} = \min_{w \in N(u)} [\ell_{uw} + d_{w1}^{(j)}], \quad u \neq 1,$
$d_{11}^{(j+1)} = 0,$

where $N(u)$ denotes the set of current neighbors of node u, and j is the iteration count.

The time complexities for the distributed Bellman–Ford algorithm to solve the SDSP problem, assuming a general n-node network topology[7] with maximum node degree of $\bar{d}$, can be simply derived as

$$T_{\text{comp}}(n) = \Theta(\bar{d}\bar{n}_{\text{hop}}) \tag{4.1}$$

and

$$T_{\text{comm}}(n) = \Theta(\bar{n}_{\text{hop}}) \tag{4.2}$$

where $\bar{n}_{\text{hop}}$ denotes the maximum number of hops along any of the $n-1$ shortest paths. It is obvious that in general $1 \leq \bar{n}_{\text{hop}} \leq n-1$.

We are able to construct a class of example BHC topologies in which $\bar{n}_{\text{hop}} = n-1$. This result implies that the generic computation and communication time complexities for the distributed Bellman–Ford algorithm remain $\Theta(\bar{d}n)$ and $\Theta(n)$, even when the underlying network is a BHC topology. Such a network is shown in Fig. 7, which depicts a simple 2-level BHC topology. The links are assumed bidirectional and

[7]We assume that network assumptions A2 through A7 of the previous section are satisfied for a general network containing n nodes.

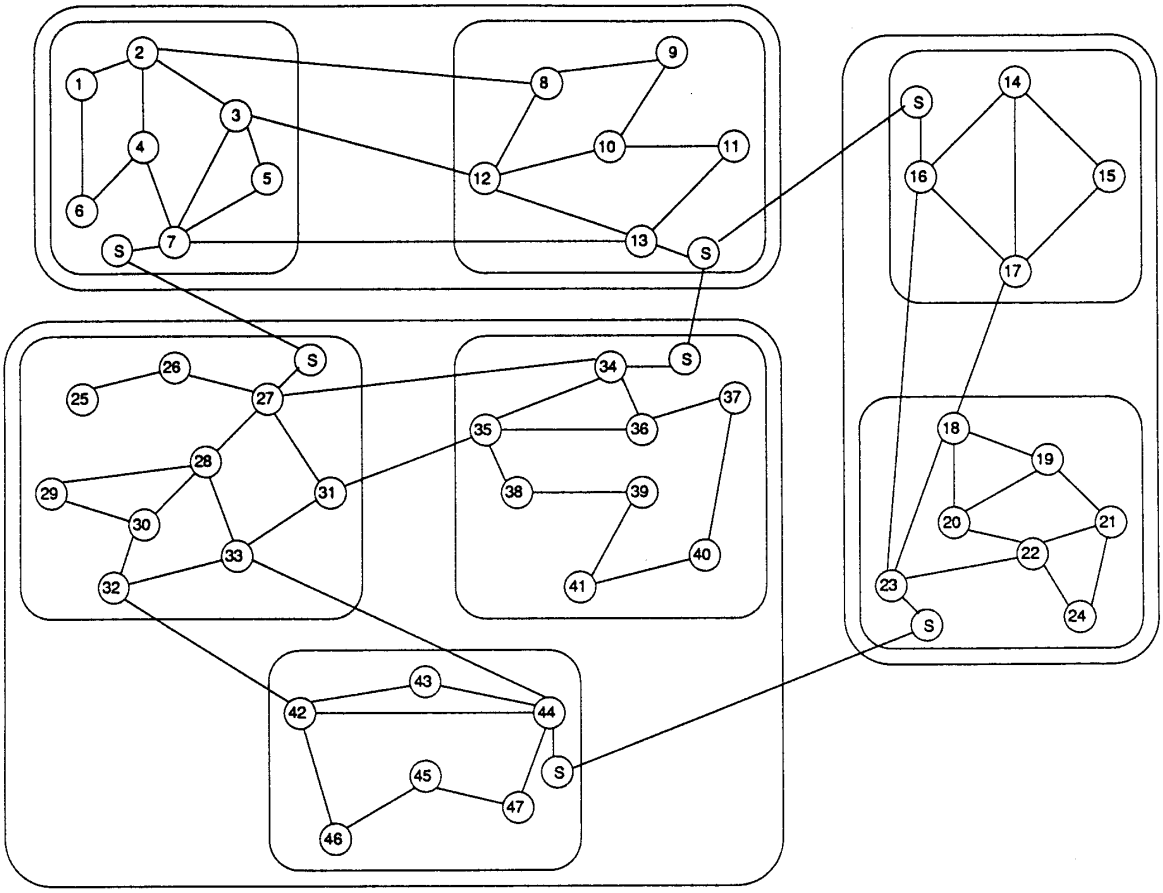

Fig. 6. The addition of the singleton clusters to the networkof Fig. 3.

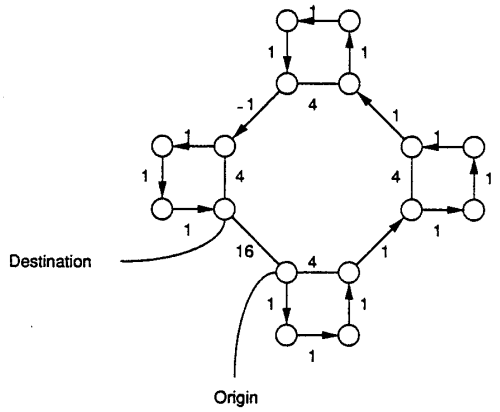

Fig. 7. A 2-level BHC topology with 16 nodes and 15 links in a shortest path.

their lengths are labeled on the figure. It is easy to verify that the shortest path (between the nodes labeled origin and destination) is the path denoted by the "arrowed" links. Note that there are 16 nodes in this 2-level BHC topology network and 15 $(= n - 1)$ links in the aforementioned shortest path. It is easy to see that 4 copies of this 2-level BHC topology can be connected together to form a 64 node 3-level BHC topology network, as shown in Fig. 8, with a shortest path containing 63 $(= n - 1)$ links. Inductively, this procedure can be repeated to produce similar such k-level BHC topologies, where k can be made arbitrarily large. So, we have a construction procedure which produces a class of BHC topologies for which the distributed Bellman–Ford algorithm has $\Theta(n)$ computation and communication time complexities. Obviously there exist a multitude of such examples where a shortest path in a BHC topology contains $\Theta(n)$ hops. The specific topologies presented here are included to verify the existence of a class of such examples.

V. Distributed Asynchronous Operation of Shortest Path Algorithms

In this section, we show that our new algorithm can actually operate in an asynchronous manner. By asynchronous, we mean that the iterations do not have to be executed in any particular order, even if there is no uniform bound for computing and communication delays. We note that the algorithm

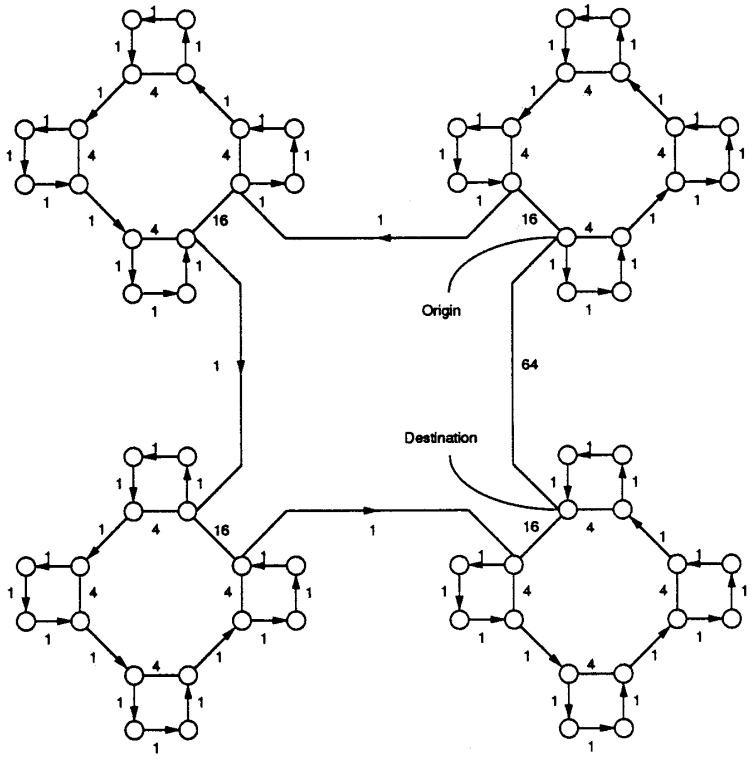

Fig. 8. A 3-level BHC topology with 64 nodes and 63 links in a shortest path.

of Section III does require some slight modifications in order to achieve asynchronous convergence.

In general it is difficult to estimate time complexities for asynchronous algorithms. Specifically, if a node is permitted to start executing a new iteration, regardless of the status of the other nodes in the network, it is not clear how many "distributed" iterations are needed for convergence. In addition, because each distributed iteration can take an arbitrarily large amount of time to complete, the algorithm may only converge at infinity (even for a finite size network). Because of these difficulties, we will not derive time complexity results for the asynchronous algorithm. However, in many realistic situations, the synchronous time complexities derived in previous sections should serve also as an optimistic, i.e., lower bound, asynchronous time complexity.

In [3] and [4], an asynchronous distributed computation model is introduced. This model has very weak assumptions on the ordering of computations, the timing of information exchange, the amount of information needed at each node, and the initial conditions for the algorithm. The distributed Bellman–Ford shortest path algorithm is given as an example algorithm which converges in this computation model. We should mention that the convergence of distributed asynchronous Bellman–Ford algorithm has been known and discussed by McQuillan *et al.* [9] and Schwartz [12], and a rigorous proof of convergence was provided by Tajibnapis [13] and Bertsekas [3], [4].

We will now show that by making some slight modifications on our new algorithm, it too converges asynchronously. The essential element involved in proving this type of convergence is centered around the fact that our new algorithm consists of solving an ordered series of dynamic programming problems. Furthermore, because each problem in this series depends only on the solution of the previous problem, it can be shown that under some mild assumptions, the overall solution is eventually achieved. Due to the space limit, only a sketch of the proof for convergence will be given. First we give the description of the new asynchronous algorithm.

Description of the New Asynchronous Algorithm: For convenience, we will describe the new asynchronous version of the algorithm by simply stating necessary changes to the original algorithm (presented in Section III). Only one modification is necessary in the existing network assumptions. Specifically, A5 is changed to the following.

A5) Each node u has stored in its memory k versions of the $(1+\bar{g}k)$-vectors d_u and e_u, denoted as $d_u^1, \cdots, d_u^k$ and $e_u^1, \cdots, e_u^k$. The first component of d_u^j, denoted d_{u1}^j, is the current estimate of the shortest distance from node u to the destination node with the constraint that the path is contained in a j-level BHC topology. The vth components of d_u^j have a similar interpretation with the gates being the destinations. The vth component of e_u^j, denoted e_{uv}^j, is the ID of the next node (from node u) along the current estimate of the shortest path associated with d_{uv}^j.

Also, the following new assumptions are added. We note that assumptions A9 and A10 were originally outlined in [3] and [4].

A8) Each node u has available the value of $LCL(u,1)$.

A9) Nodes never stop updating their own estimates and receiving messages from all their neighbors.

A10) All the initial estimates of d_{uv}^j and d_{wv}^j, $w \in N(u)$, $j \in \{1,\cdots,k\}$, are nonnegative. Furthermore, all estimates communicated to nodes by neighbors before the initial time t_o and received after time t_o are nonnegative.

A11) Each of the $k+1$ iterations of the algorithm are executed infinitely often. Note that by "$k+1$ iterations," we mean: step 1, the $k-1$ iterations of step 2, and step 3.

The asynchronous version of the algorithm is basically the same as the original algorithm with the exception that each of the k iterations, i.e., step 1 and $k-1$ iterations of step 2, uses its own versions of the distance vectors. Also, an additional step 3 is added. The modifications for the asynchronous algorithm are as follows.

0) This step is no longer needed. That is, any nonnegative initial values for the vectors d_u^j and e_u^j for all $j \in \{1,\cdots,k\}$, will suffice.

1) Use the vector d_u^1 instead of d_u to store the solution of the b_2 1-level BHC topologies.

DO $j = 1,\ k-1$.

2.1) Use d_{uv}^j instead of d_{uv} as a virtual value for ℓ_{uv}.

2.2) Use d_u^{j+1} to store the shortest distances obtained in this step.

2.3) Here, we have every gate $g_o \in G_j^i$ transmit the appropriate distance values stored in the vector $d_{g_o}^{j+1}$ to every other node in the same j-level BHC topology, for all $i \in \{1,\cdots,b^{k-j}\}$.

2.4) The new step here is as follows: Every node u computes a new $(j+1)$th version estimate of the shortest path to the given destination node (and each $(j+1)$-level parent gate), assuming it is in the same $(j+1)$-level BHC topology, by adding the jth version of its distance to each j-level gate $g_a \in G_j^i$, to the value of $d_{g_a 1}^{j+1}$ (computed in 2.2). Note that the smallest of these values is stored as d_{u1}^{j+1} and is not compared with say d_{u1}^j, as was essentially done in the synchronous version.

END DO

3) The shortest distance from all nodes u to the given destination node (and to the other $(j+1)$-level gates), denoted d_{u1}, is determined at node u by

$$d_{u1} = \min\{d_{u1}^j\} \quad \text{over all } j \in \{LCL(u,1),\cdots,k\}.$$

Next, we give a sketch of the proof for asynchronous convergence of the above algorithm. A rigorous proof is not included because the main concepts involved are essentially the same as the ones used in deriving the model algorithm of [3] and [4].

The Proof for Asynchronous Convergence: (A Sketch) First, observe that the asynchronous algorithm will converge *synchronously* to the correct solution, since the asynchronous algorithm is a generalization of the original synchronous version. In order to prove that it will also converge asynchronously to the correct solution, we need only show that each of the steps, when considered as an ordered sequence of events, depends only on the correct solution of the previous step in the sequence eventually being achieved. That is, step $2(j=1)$ requires the correct solution of step 1, before it can achieve a correct solution. Similarly, step $2(j=2)$ requires the correct solution of step $2(j=1)$, before it can compute its correct solution and so on down the line until we get to step 3 which requires the solution of step $2(j=k-1)$, before it can determine the overall correct solution. So, by the assumption that each of these steps is executed infinitely often, the overall solution will be eventually obtained.

Notice that the solution of step 1 does not depend on the solution of any other step. That is, the solution of the b_2 independent shortest path problems solved via b_2 versions of the distributed Bellman–Ford algorithm only require assumptions A9 and A10, see [4]. Thus, at some finite time say t_1, the correct solution of step 1 is guaranteed. After time t_1, step $2(j=1)$ can then achieve its correct solution by some time t_2, since step $2(j=1)$ only depends on the correct values of d_u^1 and the assumptions A9 and A10. Inductively, similar statements can be made up to step $2(j=k-1)$. That is, after some time t_k, step $2(j=k-1)$ will have produced its correct solution. So finally after time t_k, step 3 can then determine the correct overall solution. Note in step 3, the reason for minimizing over all $j \in \{LCL(u,1),\cdots,k\}$ instead of all $j \in \{1,\cdots,k\}$, is due to the fact that if $LCL(u,1) = m$, then it does not make sense to consider paths which do not encompass the nodes of a BHC topology with less than m levels. That is, due to the fact that the initial conditions on all versions of the distance vectors are assumed arbitrary, minimizing over all possible versions could lead to incorrect results. Q.E.D.

Our asynchronous algorithm has an extra advantage over the asynchronous Bellman–Ford algorithm, in that our algorithm eliminates some types of "looping problems" whereby the algorithm requires many iterations before it realizes a shortest path should include a congested link. Looping can occur if there is a sudden increase in one or more link lengths. An example of looping is depicted in Fig. 9. (See also [11].) Consider, for example, the situation whereby a link which connects gates of different clusters has a sudden increase in length. We can show that our new algorithm does not get caught in a looping problem for this type of situation. Specifically, let nodes 4 and 5 be the gates of some j-level cluster and let nodes 1 and 2 be the gates of another j-level cluster. Clearly, since our algorithm "merges" the solution of each j-level cluster together via links (4,1) and (5,2), no looping occurs. We should give warning that even though the aforementioned type of looping may be avoided in our algorithm, looping may still be a problem within any substep of the new algorithm that uses Bellman–Ford to solve the shortest path problem for a subgraph or cluster.

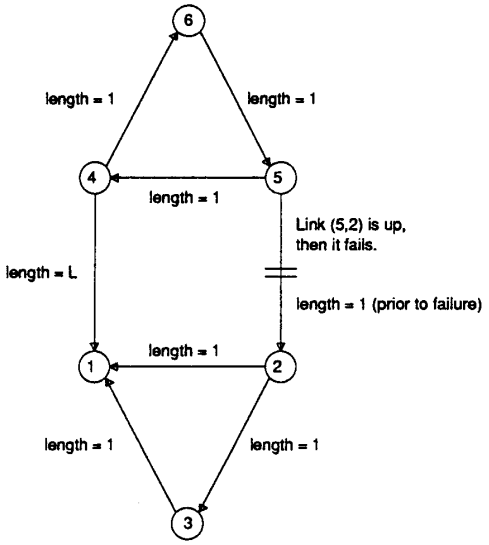

Fig. 9. Example of the looping problem associated with the Bellman–Ford algorithm. Suppose the shortest distance estimates from nodes 4, 5 and 6 to node 1 (before link (5,2) fails) are: $d_{41}^{(0)} = 4$, $d_{51}^{(0)} = 2$, $d_{61}^{(0)} = 3$. Then, after link (5,2) fails, approximately L iterations are required before node 4 realizes that its shortest path to node 1 is the direct link (4,1).

VI. Summary and Conclusions

In conclusion, we have demonstrated that a properly chosen (natural as well as artificial) organization of the node sets for a graph can significantly improve the time complexity of solving the shortest path problem in either a distributed or parallel environment. The key concept has been that inter-cluster shortest paths must contain gates (nodes through which a cluster is connected to other clusters). If the number of gates of each cluster is independent of n at all levels of the hierarchy, then the properly chosen hierarchy (which we call the BHC topology) can be exploited to obtain an $O(\log n)$ time complexity. The proposed algorithm is also shown to converge in a distributed asynchronous environment. We have also shown a somewhat surprising result that even if the minimum hop distance between all pairs of nodes in a BHC topology is $O(\log n)$, the Bellman–Ford algorithm still generically requires $\Theta(n)$ iterations to converge.

The problem of clustering a network graph into a $O(\log n)$-level BHC topology turns out to be an NP-complete problem in general, [17]. However, we have pointed out that for many large wire data networks, the hierarchical structure is obvious by inspection or is not difficult to obtain if one is satisfied with only a few levels of hierarchy. We provided a simple example showing that even if two levels are involved, the speedup of our algorithm over the existing algorithm can still be very significant. Also, under certain conditions on the graph, efficient divide and conquer algorithms from the VLSI literature are available [19]–[23].

As a final note, we mention two closely related works. A hierarchical routing protocol and hierarchical topology update and maintenance protocols have been designed and shown to perform efficiently in [17] and [18]. Following this work, our algorithm can be readily adapted into efficient protocols, to be used in a realistic data network control scheme. The hierarchical concept has also been adapted by us [14], [15] to improve the convergence rate of optimal routing algorithms. Because most optimal routing algorithms (including the one in [14] and [15]) require shortest path solutions at each iteration, our proposed algorithm is readily applicable.

References

[1] A. V. Aho, J. E. Hoptcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Reading, MA: Addison-Wesley, 1974.
[2] E. Benhamou and J. Estrin, "Multilevel internetworking gateways: Architecture and applications," *IEEE Comput Mag.*, vol. 16, no. 9, pp. 27–34, Sept. 1983.
[3] D. Bertsekas, "Distributed dynamic programming," *IEEE Trans. Automat. Contr.*, vol. AC-26, pp. 610–616, 1982.
[4] D. Bertsekas, "Distributed asynchronous computation of fixed points," *Math Prog.*, vol. 27, pp. 107–120, 1983.
[5] N. Deo, C. Pang, and R. E. Lord, "Two parallel algorithms for shortest path problems," in *Proc. 1980 Int. Conf. Parallel Processing*, IEEE, New York, Aug. 1980, pp. 244–253.
[6] E. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.
[7] L. R. Ford, Jr. and D. R. Fulkerson, *Flows in Networks*. Princeton, NJ: Princeton University Press, 1962.
[8] R. Hinden, J. Haverty, and A. Sheltzer, "The DARPA Internet: Interconnecting heterogeneous computer networks with gateways," *IEEE Comput. Mag.*, vol. 16, no. 9, pp. 38–48, Sept. 1983.
[9] J. M. McQuillan, G. Falk, and I. Richer, "A review of the development and performance of the ARPANET routing algorithm," *IEEE Trans. Commun.*, vol. COM-26, no. 12, pp. 1802–1811, Dec. 1978.
[10] N. F. Schneidewind, "Interconnecting local networks to long-distance networks," *IEEE Comput. Mag.*, vol. 16, no. 9, pp. 15–24, Sept. 1983.
[11] M. Schwartz and T. E. Stern, "Routing techniques used in computer communication networks," *IEEE Trans. Commun.*, vol. COM-28, no. 4, pp. 539–552, Apr. 1980.
[12] M. Schwartz, "Analysis of congestion control techniques in computer routing and flow control in data networks," *NATO Advanced Study Inst.: New Concepts in Multi-User Communications*, Norwich, U.K., Aug. 4–16, 1980; Sijthoof and Nordhoof, The Netherlands.
[13] W. D. Tajibnapis, "A correctness proof of a topology information maintenance protocol for distributed computer networks," *Commun. ACM*, vol. 10, pp. 477–485, July 1977.
[14] W. K. Tsai, G. M. Huang, J. K. Antonio, and W. T. Tsai, "Distributed iterative aggregation algorithms for box-constrained minimization problems and optimal routing in data networks," *IEEE Trans. Automat. Contr.*, Jan. 1989.
[15] ——, "Distributed aggregation/disaggregation algorithms for optimal routing in data networks," in *Proc. Automat. Contr. Conf.*, Atlanta, GA, June 1988.
[16] W. K. Tsai, J. K. Antonio, and G. M. Huang, "Fast parallel hierarchical aggregation and disaggregation algorithms for multistage optimization problems and the shortest path problems," under preparation.
[17] W. T. Tsai, "Control and management of large and dynamic networks," Ph.D. dissertation, Dep. EECS, UC Berkeley, 1985.
[18] W. T. Tsai, C. V. Ramamoorthy, W. K. Tsai, and O. Nishiguchi, "An adaptive hierarchical routing protocol," *IEEE Trans. Comput.*, vol. 38, no. 8, pp. 1059–1075, Aug. 1989.
[19] L. G. Valiant, "Universality considerations in VLSI circuits," *IEEE Trans. Comput.*, vol. 30, no. 2, pp. 135–140, 1981.
[20] C. E. Leiserson, "Area efficient graph algorithms (for VLSI)," in *Proc. Twenty-First Annu. IEEE Symp. Foundations Comput. Sci.*, 1980, pp. 270–281.
[21] R. W. Floyd and J. D. Ullman, "The compilation of regular expressions into integrated circuits," *J. ACM*, vol. 29, no. 2, pp. 603–622, 1982.
[22] J. D. Ullman, *Computational Aspects of VLSI*. Rockville, MD: Computer Science Press, 1984.
[23] F. T. Leighton, "A layout strategy for VLSI which is provably good," in *Proc. Fourteenth Annu. ACM Symp. Theory of Comput.*, 1982, pp. 1–12.

John K. Antonio (S'85–M'89) was born in Fort Worth, TX, in 1961. He received the B.S. degree (with honors), the M.S. degree, and the Ph.D degree, all in electrical engineering from Texas A&M University, College Station, in 1984, 1986, and 1989, respectively.

He worked with the Digital Flight Control Group, General Dynamics/Fort Worth Division, during the summers of 1983 and 1984. From 1984 to 1986, he was both a teaching assistant and research assistant, while working on his Master's degree. During the summer of 1986, he was employed by General Motors Research Laboratories, Warren, MI, as a Research Associate. From 1986 until 1989, he held the position of lecturer in the Department of Electrical Engineering at Texas A&M University, while pursuing the Ph.D. After graduation, he accepted employment at Purdue University, where he currently holds the position of Assistant Professor of electrical engineering. During the summer of 1991, he participated in the summer faculty research program at Rome Laboratory, Rome, NY, where he conducted research in the area of high-performance computing. His research interests include analysis and design of fast distributed algorithms for high-speed computer networks, iterative methods for solving large scale computational problems, optimal routing in data networks, computational aspects of control and optimization, and control theory.

Dr. Antonio is a member of the IEEE Computer Society, Control Systems Society, and Communications Society. Also, he is a member of Tau Beta Pi, Eta Kappa Nu, and Phi Kappa Phi. In 1989 he received the Ruth and Joel Spira "Outstanding Teacher Award" from the School of Electrical Engineering at Purdue University. He is currently developing a graduate course entitled "Parallel and Distributed Computation for Optimization and Control."

Garng M. Huang (S'76–M'80–SM'85) received the B.S. and M.S. degrees in electrical engineering from National Chiao Tung University, Hsinchu, Taiwan, Republic of China, in 1975 and 1977, respectively. He received his doctorate degree in systems science and mathematics from Washington University, St. Louis, MO, in 1980, and taught there until 1984.

In 1984 he joined Texas A&M University, Department of Electrical Engineering, where he is currently an Associate Professor. He has worked on many funded research projects, such as Emergency Control of Large Interconnected Power System, HVDC Systems, Restoration of Large Scale Power Systems, Online Detection of System Instabilities and Online Stabilization of Large Power System, fast parallel/distributed textured algorithms, and fast parallel textured algorithms for large power systems. His current interest is in large-scale systems theory, large-scale and distributed computing and control, and their applications.

Dr. Huang is a Registered Professional Engineer of Texas, the Committee Chairman of the Energy System Control Committee of the IEEE Transactions on Automatic Control Society, and a member of the Systems Engineering Committee of the IEEE PAS Society. He has published more than 80 papers and reports in the areas of nonlinear, distributed control systems and parallel computing and their applications to power systems, data networks, and flexible structures.

Wei K. Tsai (S'79–M'86) received the B.Sc., M.Sc., and the Ph.D. degrees, all in electrical engineering, from the Massachusetts Institute of Technology, Cambridge, in 1979, 1982, and 1986, respectively.

In the summer of 1982, he was on the engineering staff of the New England Electric Power Service Company, Westboro. From September 1986 through October 1989, he was an assistant professor in the Department of Electrical Engineering at Texas A&M University, College Station. In September 1989, he joined the University of California at Irvine, Department of Electrical and Computer Engineering, where he is now an Associate Professor. His research interests include data communication networks, artificial neural networks, parallel and distributed algorithms, parallel computer architectures, and control systems.

Dr. Tsai is a member SIAM, Sigma Xi, and Eta Kappa Nu.