Uniform d-emulations of rings, with an application to distributed virtual ring construction

E.M. Bakker, J. van Leeuwen

RUU-CS-91-21
June 1991

Utrecht University

Department of Computer Science

Padualaan 14, P.O. Box 80.089,
3508 TB Utrecht, The Netherlands,
Tel. : ... + 31 - 30 - 531454

Uniform d-emulations of rings, with an application to distributed virtual ring construction

E.M. Bakker, J. van Leeuwen

Technical Report RUU-CS-91-21
June 1991

Department of Computer Science
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
The Netherlands

ISSN: 0924-3275

Uniform d-Emulations of Rings, with an Application to Distributed Virtual Ring Construction *

Erwin M. Bakker, Jan van Leeuwen
Department of Computer Science, Utrecht University
P.O.Box 80.089, 3508 TB Utrecht, the Netherlands

Abstract

Emulations are a special kind of structure-preserving mappings between processor interconnection networks (i.e., graphs). In this paper the notion of (uniform) emulation is generalized to the notion of (uniform) d-emulation. Several problems concerning the complexity of (uniform) d-emulations are studied. It is shown that the problem of deciding for arbitrary graphs H and G whether H can be uniformly d-emulated on G is NP-complete for every fixed $d \in \mathbb{N}$. Also it is shown that the problem of deciding for arbitrary rings R and planar graphs G whether R can be uniformly 2-emulated on G is NP-complete. Further, a new constructive proof is given of the fact that there always exists a uniform 3-emulation of a ring R on a graph $G = (V, E)$, if $|V_R| = k.|V_G|$ and $k \in \mathbb{N}$. This uniform 3-emulation of R on G is used as a basis for a Distributed Virtual Ring Construction algorithm that uses $2|E|$ messages of $O(1)$ bits and has time complexity $\leq 2(|V| - 1)$. A traversal of the constructed virtual ring costs $2(|V| - 1)$ messages.

1 Introduction

Many distributed algorithms assume a special underlying structure of the communication network on which they are implemented. For example, it is desirable for many token-based distributed algorithms such as leader finding (see e.g. [B86]) that the underlying network has a ring structure. These algorithms demand a sequential traversal of all the nodes (processes) by a token. The implementation of such a traversal is easy in the case of a ring structure, whereas it may cost many extra messages to traverse all the nodes of a network that does not have this topology. To

*This work was partially supported by the ESPRIT Basic Research Actions of the EC under contract no. 3075 (project ALCOM).

match the demand for a special topology of the network there are two approaches. One approach is to physically create this special structure in the network. Hence it must be possible to rearrange the links, or the network must already have the desired topology. In general this is not the case. Although this leads to interesting problems, we will not study this approach here. A second approach is to build a *virtual* network with the desired special structure on the given (arbitrary but fixed) network. This method is followed in [A85], [HR87] and [HR88] and is well-known in networking. In these papers a *virtual ring* is created in an arbitrary communication network, i.e., a ring network is "simulated" on the given network. A useful notion of efficient simulation of a network on another network was introduced by Fishburn and Finkel in [FF82] and is called *emulation*. Extensive results on emulations can be found in [B86] and [BL86].

Definition 1.1 *Let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ be graphs. G can be* emulated *on H iff there exists a function $f : V_G \to V_H$ such that for every edge $(v, w) \in E_G$: $f(v) = f(w)$ or $(f(v), f(w)) \in E_H$. The function f is called an* emulation *of G on H. The emulation f is called a* uniform emulation *of G on H iff for all $v, w \in V_H$: $|f^{-1}(v)| = |f^{-1}(w)|$.*

Let the graphs G and H model two different communication networks. According to the definition, a function that maps all the nodes of G onto one node of H is an emulation of G on H. In this case the network modeled by G is simulated on exactly one node of the network modeled by H. Obviously this is not a desirable situation. In general we want to distribute the work-load over all nodes of the network modeled by H. Therefore in the following we will require that emulations f are "onto".

If G is emulated on H, then by definition for every edge $(v, w) \in E_G$: $f(v) = f(w)$ or $((f(v), f(w)) \in E_H$. Thus if a ring $R = (V_R, E_R)$ is emulated on H, and $|V_R| = |V_H|$, then H must be Hamiltonian. In this case the notion of emulation is rather restrictive, as many graphs are not Hamiltonian. Therefore we introduce a generalization of this notion called: *distance-d emulation* ($d \in \mathbb{N}$), or *d-emulation* for short. In the following $d_G : V_G \times V_G \to \mathbb{N}$ denotes the usual distance function on a graph $G = (V_G, E_G)$, i.e., $d_G(v, w)$ is the length of the shortest path between v and w in G (undefined if v and w are not connected). The subscript G is omitted if there is no chance of confusion.

Definition 1.2 *Let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ be graphs, and $d \in \mathbb{N}$. G can be* d-emulated *on H iff there exists a function $f : V_G \to V_H$ such that for every edge $(v, w) \in E_G$: $d_H(f(v), f(w)) \leq d$. The function f is called a* d-emulation *of G on H. The d-emulation f is called a* uniform d-emulation *of G on H iff for all $v, w \in V_G$: $|f^{-1}(v)| = |f^{-1}(w)|$.*

As before we will require that d-emulations be "onto". If f is a d-emulation of G on H, then for every edge $(v, w) \in E_G$: $d_H(f(v), f(w)) \leq d$. The idea is that

G is the *virtual network* and H is the *physical network*. Thus a virtual link, i.e., a link of the network modeled by G, consists of $\leq d$ physical links, i.e., links of the network modeled by H. The d could be called the *dilation* of the mapping. Note that 1-emulations are precisely the emulations as defined in Definition 1.1. It is clear that for larger d, larger classes of graphs can be (uniformly) d-emulated on a graph H. In fact, for $d = \text{diam}(H)$, every graph G with $|V_G| = k \cdot |V_H|$ (k a positive integer) can be uniformly d-emulated on H. In this paper we will study the notion of d-emulation in more detail and especially consider d-emulations of rings. d-Emulations of rings turn out to be the key-notion needed for the design of virtual rings in communication networks. We will indicate how this is used in the context of distributed computing.

The paper is organized as follows. In the remainder of this section some basic results concerning (uniform) d-emulations are given. In Section 2 the complexity of finding uniform d-emulations is studied. It is shown that it is NP-hard to decide whether there exists a uniform d-emulation of a graph G on an arbitrary graph H for every fixed $d \in \mathbf{N}$. Furthermore, it is shown that it is NP-hard to decide whether there exists a uniform d-emulation of a ring on an arbitrary planar graph for $d = 2$. It is however always possible to find a uniform 3-emulation of a ring on an arbitrary graph (this follows from [K68]). We give a new constructive proof of this fact. Next we discuss the application to distributed virtual ring construction. The distributed virtual ring construction algorithm as described in [HR87] and [HR88] constructs a virtual ring on a graph H, but virtual links may consist of up to $2(|V_H| - 1)$ physical links. In Section 3 a new Distributed Virtual Ring Construction algorithm is described in which each virtual link consists of at most 3 physical links. In the same section some conclusions for distributed computing and suggestions for further research are given.

We end this section with some basic properties of uniform d-emulations that will be useful in the sequel.

Notation 1.3 *Let $G = (V, E)$ be a graph. For every $v \in V$ and $d \in \mathbf{N}$ the set of nodes at distance $\leq d$ of v is called the d-neighborhood of v and denoted as $N_{d,G}[v] = \{w \mid w \in V \text{ and } d(v,w) \leq d\}$. We let $N_{d,G}(v) = N_{d,G}[v] \setminus \{v\}$. We ommit the subscript G if there is no chance of confussion.*

Let $H = (V_H, E_H)$ be a graph. Recall that the kth power of H, denoted as H^k, is the graph with the same node-set as H and $(v,w) \in E_{H^k}$ iff $d_H(v,w) \leq k$. We have the following lemma.

Lemma 1.4 *Let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ be graphs. Let $k, d \in \mathbf{N}$ be such that $k \mid d$ (k divides d). There exists a (uniform) d-emulation of G on H iff there exists a (uniform) (d/k)-emulation of G on H^k.*

Proof. For all $(v,w) \in E_H$: $d_H(v,w) \leq d$ iff $d_{H^k}(v,w) \leq (d/k)$. Hence a function $f : V_G \to V_H$ is a (uniform) d-emulation of G on H iff f is a (uniform) (d/k)-emulation of G on H^k. □

From this lemma it follows that there exists a (uniform) d-emulation of G on H if and only if there exists a (uniform) emulation of G on H^d.

Lemma 1.5 *Let f be a uniform d-emulation of G on H, and $|V_G| = |V_H|$. If $v \in V_G$ and $N_{1,G}(v) \geq k$ ($k \in \mathbf{N}$), then $N_{d,H}(f(v)) \geq k$.*

Proof. f maps all the nodes that are element of $N_{1,G}(v)$ within distance d of $f(v)$. □

For all standard notions from graph theory the reader is referred to [H69]. Also some familiarity with distributed algorithms and networking is assumed (see e.g. [M89], [SK87]).

2 The Complexity of Finding Uniform d-Emulations

In the following it is assumed that the reader is familiar with the theory of NP-completeness, cf. [GJ79]. The problems studied in this section will be stated in the same format as used in this book. The first problem of which the complexity is studied is UNIFORM d-EMULATION.

Problem: UNIFORM d-EMULATION. (Fixed $d \in \mathbf{N}$.)
Instance: Connected graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$.
Question: Is there a uniform d-emulation of G on H?

In [B86] it is shown that this problem is NP-complete for $d = 1$. Here it will be shown that UNIFORM d-EMULATION is NP-complete for every fixed $d \in \mathbf{N}$. An interesting restricted version of UNIFORM d-EMULATION is the following problem.

Problem: UNIFORM d-EMULATION OF RINGS. (Fixed $d \in \mathbf{N}$.)
Instance: A ring $R = (V_R, E_R)$ and a connected graph $H = (V_H, E_H)$.
Question: Is there a uniform d-emulation of R on H?

Recall that a simple circuit in $G = (V, E)$ is a sequence $[v_1, ..., v_t]$ of distinct nodes $v_i \in V$ such that $(v_i, v_{i+1}) \in E$ for $i \in \{1, ..., (t-1)\}$ and such that $(v_t, v_1) \in E$. A Hamiltonian circuit in G is a simple circuit that includes all the nodes of G. (Hamiltonian paths are defined the same way, without the requirement that $(v_t, v_1) \in E$.) The HAMILTONIAN CIRCUIT problem is defined as follows.

Problem: HAMILTONIAN CIRCUIT.
Instance: A graph $G = (V, E)$.
Question: Does G contain a Hamiltonian circuit?

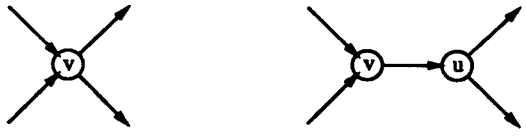
By Lemma 1.4 it is clear that if $|V_R| = |V_H|$, then UNIFORM d-EMULATION OF RINGS becomes equivalent to the question of deciding whether H^d is Hamiltonian. It is well-known that HAMILTONIAN CIRCUIT is NP-complete (see e.g. [GJT76]). Hence UNIFORM 1-EMULATION OF RINGS is NP-complete. Chvátal proved in [C76] that HAMILTONIAN CIRCUIT restricted to graphs that are the square of a graph is NP-complete. It follows that UNIFORM 2-EMULATION OF RINGS is NP-complete as well. In Section 2.2 it will be shown that UNIFORM 2-EMULATION OF RINGS is NP-complete even when restricted to graphs that are planar. From this it follows that HAMILTONIAN CIRCUIT restricted to graphs that are the square of a planar graph is NP-complete. Karaganis proved in [K68] that the cube of a nontrivial connected graph G is Hamiltonian connected, i.e., there exists a Hamiltonian path between any two nodes of G^3. From this it easily follows that G^3 is Hamiltonian. In Section 2.2 an alternative, constructive proof of this fact is given. This proof will be used as a basis for the Distributed Virtual Ring Construction Algorithm in Section 3. Section 2.3 contains some remarks on polynomial-time algorithms for constructing uniform d-emulations of rings. These polynomial-time algorithms merely exist as a consequence of the existence of polynomial-time algorithms for the HAMILTONIAN CIRCUIT problem restricted to special classes of graphs.

2.1 The Complexity of UNIFORM d-EMULATION

In order to prove that UNIFORM d-EMULATION is NP-complete the following lemma is useful.

Lemma 2.1 *HAMILTONIAN CIRCUIT is NP-Complete for undirected bipartite cubic graphs.*

Proof In [P79] it is proven that HAMILTONIAN CIRCUIT is NP-complete for directed graphs (*digraphs*) where every node v has indegree(v) = outdegree(v) = 2. Such a digraph G can easily be transformed into an undirected bipartite cubic graph G' that has a Hamiltonian circuit if and only if the original digraph has. To see this, first transform G into the digraph G_1 by replacing every node $v \in V$ and its adjacent in- and out-going edges (see Figure 2.1a) by the subgraph shown in Figure 2.1b.



a) b)

Figure 2.1

It is easy to verify that G has a Hamiltonian circuit if and only if G_1 has. Next transform G_1 into the undirected graph G_2 by replacing every subgraph of G_1 as shown in Figure 2.1b by a subgraph as shown in Figure 2.2.

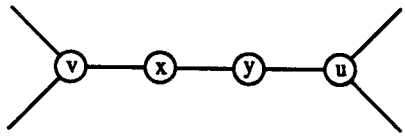


Figure 2.2

Again it is easy to see that G_1 has a Hamiltonian circuit iff G_2 has. Furthermore, G_2 is an undirected bipartite graph. Finally we replace every subgraph as shown in Figure 2.2 by a subgraph as shown in Figure 2.3 to obtain an undirected bipartite cubic graph G' that has a Hamiltonian circuit iff G_2, and hence, iff the original digraph G has a Hamiltonian circuit. Clearly the transformation is polynomial-time computable.

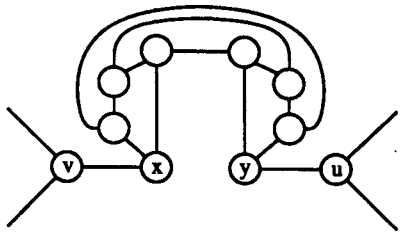


Figure 2.3

□

Theorem 2.2 *UNIFORM d-EMULATION is NP-complete for every fixed* $d \geq 1$.

Proof. From [B86] we know that the theorem is true for $d = 1$. Now assume that a fixed $d \in \mathbf{N}, d \geq 2$ is given. Clearly the problem is in NP, since a nondeterministic algorithm can always guess a function $f : V_G \rightarrow V_H$ and check whether it is a uniform d-emulation in polynomial time.
To prove NP-completeness, we transform HAMILTONIAN CIRCUIT for undirected bipartite cubic graphs (cf. Lemma 2.1) to this problem. Let $G = (V_G, E_G)$ be an arbitrary connected bipartite cubic graph with $V_G = \{v_0, ..., v_{(n-1)}\}$. Note that $|V_G|$ is even and $|E_G| = 3\,|V_G|\,/2$. We will construct a graph G_1 and a graph T such that G has a Hamiltonian circuit iff there is a uniform d-emulation of T on G_1. Transform G into a graph $G_1 = (V_{G_1}, E_{G_1})$ by connecting a structure as pictured in Figure 2.4 to every $v_i \in V_G$. Some nodes are given special names for later reference. (The nodes are named from v_i^0 to v_i^r, passing over ϕ_i which is not indexed as a v_i-node. Node v_i^0 is identified with v_i.)

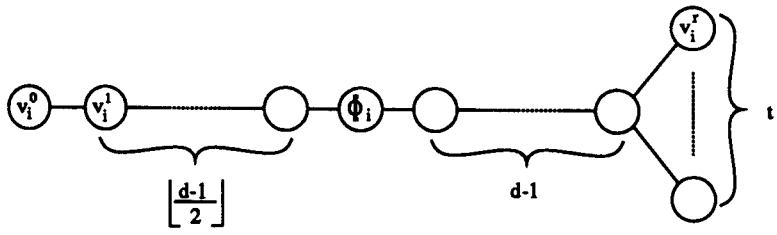


Figure 2.4

We take $t \geq 6d.(2^d - 1)$. Furthermore, if d is even we introduce a new node u for every edge $(v, w) \in E_G$ and replace the edge (v, w) by two new edges (v, u) and (u, w) in G_1 (see Figure 2.5). Define U_G as the set of all the new nodes u that are introduced in this way.

Figure 2.5

Note that if $(v_i, v_j) \in E_G$, then $d_{G_1}(\phi_i, \phi_j) = d$ for d odd and d even. Next we construct the graph $T = (V_T, E_T)$ as pictured in Figure 2.6.

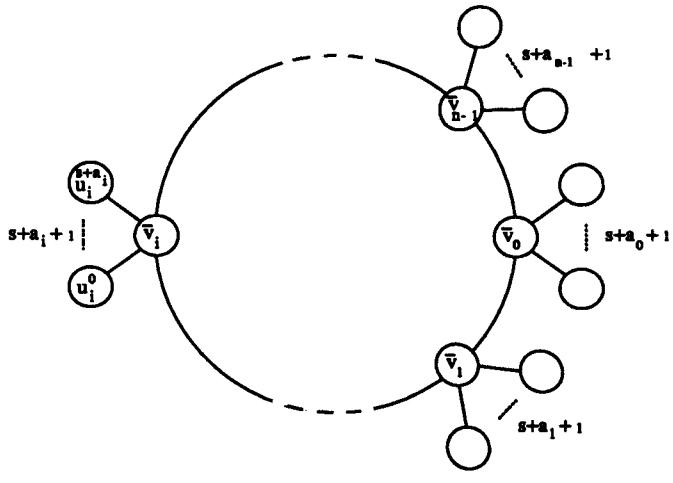


Figure 2.6

In Figure 2.6 we take

$$a_i = \begin{cases} 0 & \text{if } d \text{ is odd or } i \text{ is odd,} \\ 3 & \text{otherwise,} \end{cases}$$

and $s = \lfloor \frac{d-1}{2} \rfloor + d + t - 2$. Let V denote the set $\{0, ..., (n-1)\}$.

Claim 2.3 *G contains a Hamiltonian circuit if and only if there exists a uniform d-emulation of T on G_1.*

Proof. [⇒] Let $H = (V_H, E_H)$ be a Hamiltonian circuit in G. W.l.o.g. we may assume that $E_H = \{(v_0, v_1), (v_1, v_2), ..., (v_{n-1}, v_0)\}$. Define the function $f : V_T \to V_{G_1}$ by:

1. $f(\overline{v_i}) = \phi_i$ and $f(u_i^0) = v_i$, for all $i \in V$.

2. for all $i \in V$ and for all $j \in \{1, ..., s\}$: $f(u_i^j) = v_i^j$.

3. if d and i are even, i.e., if $a_i = 3$, then for all $j \in \{1, 2, 3\}$: $f(u_i^{s+j}) = w_j$, where w_1, w_2 and w_3 are the three nodes from the set U adjacent to v_i in G_1.

For every $i \in V$, $\overline{v_i}$ and $\overline{v_{(i+1) \bmod n}}$ are mapped to ϕ_i and $\phi_{(i+1) \bmod n}$, respectively. Note that $d_{G_1}(\phi_i, \phi_{(i+1) \bmod n}) = d$. Furthermore, $d_{G_1}(v_i^j, \phi_i) \leq d$ for every $i \in V$ and every $j \in \{0, ..., (s + a_i)\}$. Hence all the nodes u_i^j are mapped to nodes within distance d of $f(\overline{v_i}) = \phi_i$. As G is a bipartite graph we know that if i and j are even, then $(v_i, v_j) \notin E_G$. Therefore point 3) of the definition of f translates to: for every $u \in U$ there exists exactly one u_i^j such that $f(u_i^j) = u$. Hence f defines a uniform d-emulation of T on G_1.
[⇐] Let $f : V_T \to V_{G_1}$ be a uniform d-emulation of T on G_1. We claim that there is a permutation π of V such that for all $i \in V$, $f(\overline{v_i})$ must be equal to $\phi_{\pi(i)}$. f must map all the $(s + a_i + 3)$ neighbors of $\overline{v_i}$ to different nodes within distance d of $f(\overline{v_i})$. G is a cubic graph. It follows that for all $v \in V_G : |N_{d,G}(v)| \leq 3(2^d - 1)$ by the Moore bound (cf. [B85]). Hence for all $w \in W = V_G \cup U \cup \{v_i^j \mid j \in \{1, ..., \lfloor \frac{d-1}{2} \rfloor - 1\}$ and $i \in V\}$: $|N_{d,T}[w]| \leq 3d(2^d - 1)$, whereas there are $(s + a_i + 3) > 3d(2^d - 1)$ neighbors of $\overline{v_i}$. Therefore it is impossible that $f(\overline{v_i})$ is an element of W. Now assume that there exists an $i \in V$ such that $f(\overline{v_i}) \in V_{G_1} - W - \{\phi_j \mid j \in V\}$. Because f is a uniform d-emulation it follows that $|N_{d,G_1}[f(\overline{v_i})]| + \left|N_{d,G_1}[f(\overline{v_{(i+1) \bmod n}})]\right| \leq 6d(2^d - 1) + t$. But there are more than $2t > 6d(2^d - 1) + t$ neighbors of $\overline{v_i}$ and $\overline{v_{(i+1) \bmod n}}$. Hence it is impossible to map all these neighbors to the nodes within distance d of $f(\overline{v_i})$ and $f(\overline{v_{(i+1) \bmod n}})$. Thus we conclude that $f(\overline{v_i})$ must be equal to some ϕ-node, for every i. Thus a permutation π as claimed must exist. Furthermore, it is clear that if $(\overline{v_i}, \overline{v_j}) \in E_T$ and $f(\overline{v_i}) = \phi_{i_1}$ and $f(\overline{v_j}) = \phi_{i_2}$, then $(v_{i_1}, v_{i_2}) \in E_G$. Now it is easy to verify that $[(v_{\pi(0)}, v_{\pi(1)}), ..., (v_{\pi(n-1)}, v_{\pi(0)})]$ defines a Hamiltonian circuit in G. □

The entire construction is clearly polynomial-time computable. From Claim 2.3 it follows that UNIFORM d-EMULATION is NP-complete for every fixed $d \geq 2$.
□

2.2 The Complexity of UNIFORM d-EMULATIONS OF RINGS

Theorem 2.4 *There is a polynomial transformation from HAMILTONIAN CIRCUIT to UNIFORM 1-EMULATION OF RINGS.*

Proof. If R is a ring and H a graph such that $|V_R| = |V_H|$, then the problem of deciding whether there exists a uniform 1-emulation of R on H is equivalent to the problem of deciding whether H has a Hamiltonian circuit. □

Chvátal proved in [C76] that the HAMILTONIAN CIRCUIT problem is NP-complete for graphs that are the square of a graph. With Lemma 1.4 this implies that UNIFORM 2-EMULATION OF RINGS is NP-complete. The next theorem improves this result.

Theorem 2.5 *UNIFORM 2-EMULATION OF RINGS is NP-complete for planar graphs.*

Proof. Clearly the problem is in NP, since a nondeterministic algorithm can guess a function $f : V_R \to V_H$ and check whether it is a uniform 2-emulation in polynomial time. To prove NP-completeness, we use a transformation from HAMILTONIAN CIRCUIT for undirected planar graphs with nodes of degree ≤ 3 only. In [IPS82] this problem is proven to be NP-complete. Let $G = (V_G, E_G)$ be a connected planar graph such that for all $v \in V_G$: degree$(v) \leq 3$. We construct a graph $G_1 = (V_{G_1}, E_{G_1})$ by replacing each edge $(v, w) \in E_G$ (see Figure 2.7a) by a subgraph as pictured in Figure 2.7b.

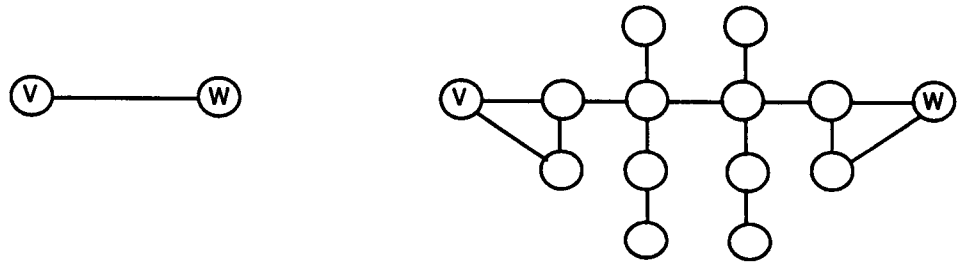


Figure 2.7)

Clearly G_1 is planar and the transformation is polynomial-time computable. Now the following is claimed.

Claim 2.6 *G contains a Hamiltonian circuit if and only if there exists a uniform 2-emulation of the ring $R = (V_R, E_R)$ on $G_1 = (V_{G_1}, E_{G_1})$, where R is such that $|V_R| = |V_{G_1}|$.*

Proof. [⇒] Let $H = (V_H, E_H)$ be a Hamiltonian circuit in G. Let $V_H = V_G = \{v_0, ..., v_{n-1}\}$. W.l.o.g. we may assume that $E_H = \{(v_0, v_1), (v_1, v_2), ..., (v_{n-1}, v_0)\}$. Every $v_i \in V_G$ has degree 2 or 3 in G. Let the nodes of the ring R be numbered, say clockwise in n consecutive parts, for $i \in \{0, ..., (n-1)\}$ as follows: $v_i^0, ..., v_i^{12}$ if v_i has degree 2 in G, and $v_i^0, ..., v_i^{18}$ if v_i has degree 3 in G. A uniform 2-emulation $f : V_R \to V_{G_1}$ can be defined as follows. Note that $(v_i, v_{(i+1) \bmod n}) \in E_H$. Define $f(v_i^0) = v_i$ and for all $j \in \{1, ..., 12\}$: $f(v_i^j) = w_j$. See Figure 2.8. (The dotted lines stand for edges in R, whereas the continuous lines represent edges in G_1.)

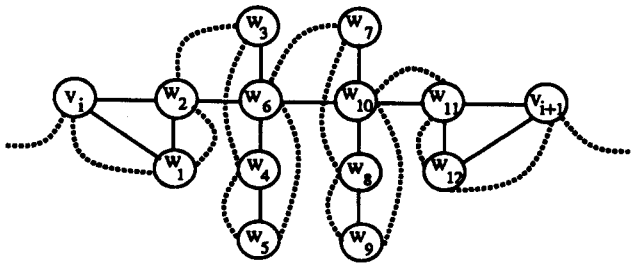


Figure 2.8

If v_i has degree 3 in G, then there exists a $k \in \{0, ..., (n-1)\}$ such that $(v_i, v_k) \in E_G$ but $(v_i, v_k) \notin E_H$. Define for all $j \in \{1, ..., 6\}$: $f(v_i^{j+12}) = x_j$. See Figure 2.9.

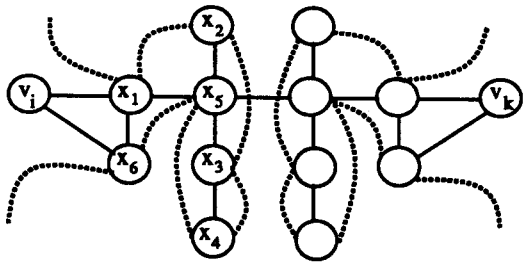


Figure 2.9

It is straightforward to verify that this defines a uniform 2-emulation of R on G_1.

[⇐] Assume there exists a uniform 2-emulation f of the ring $R = (V_R, E_R)$ on $G_1 = (V_{G_1}, E_{G_1})$, with $|V_R| = |V_{G_1}|$. Let $u_0, ..., u_{|V_R|-1}$ be a numbering of the nodes of the ring R, say in clockwise order. Consider Figure 2.10.

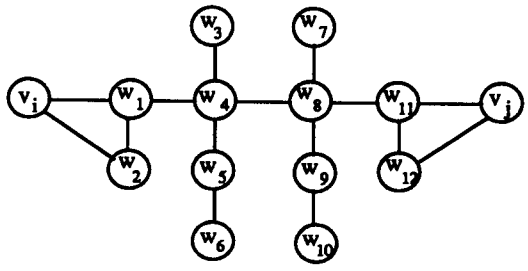


Figure 2.10

If $f(u_i) = w_6$, then either $f(u_{i-1}) = w_4$ and $f(u_{i+1}) = w_5$, or $f(u_{i-1}) = w_5$ and $f(u_{i+1}) = w_4$. Something similar holds for w_8, w_9 and w_{10}.
Assume that we have a consecutive part $u_i, ..., u_{i+l}$ of the ring R such that $f(u_i) = w_1$ and $f(u_{i+l}) = w_2$, and there exists a $t \in \{(i+1), ..., (i+l-1)\}$ such that $f(u_t) = w_3, w_4$ or w_5. It follows that $f(u_{i+l-1}) = w_4$, $f(u_{i+l-2}) = w_6$ and $f(u_{i+l-3}) = w_5$. Let I denote the set $\{(i+2), ..., (i+l-2)\}$. It is easy to see that $f(u_{i+1}) \neq w_8$ because this would imply that there does not exist a $t \in I$ such that $f(u_t) = w_3$. (In fact it would imply that there does not exist a $u \in V_R$ such that $f(u) = w_3$.) Similarly $f(u_{i+1}) \neq w_5$. It follows that $f(u_{i+1}) = w_3$. Thus it is clear that there does not exist a $t \in I$ such that $f(u_t) = w_8$. Because this would imply that u_t must have 3 neighbors in the ring R, say u'_1, u'_2 and u'_3 such that $f(u'_1) = w_3, f(u'_2) = w_5$ and $f(u'_3) = w_{10}$, which is impossible. Thus, either a consecutive part $u_i, ..., u_{i+5}$ is mapped to the nodes $w_1, ..., w_6$, where $f(u_i) = w_1$ and $f(u_{i+5}) = w_2$, in which case we take $(v_i, v_j) \notin E_H$, or the consecutive part $u_i, ..., u_{i+11}$ is mapped to the nodes $w_1, ..., w_{12}$, where $f(u_i) = w_1$ and $f(u_{i+11}) = w_{12}$, in which case we take $(v_i, v_j) \in E_H$. This defines a Hamiltonian circuit in G. □

From this claim it follows that UNIFORM 2-EMULATION of rings is NP-complete. □

In [K68] Karaganis proved that the cube of a nontrivial connected graph is Hamiltonian connected, and hence Hamiltonian. Here we give an alternative, constructive proof of this result. The main reason for this alternative proof is that it can be used as a basis for a Distributed Virtual Ring Construction Algorithm (see Section 3).

Theorem 2.7 *Let $R = (V_R, E_R)$ be a ring and $H = (V_H, E_H)$ a connected graph, then it is always possible to find a uniform 3-emulation of R on H.*

Proof. Let $T = (V_T, E_T)$ be a rooted spanning tree of H. We will show that it is always possible to construct a uniform 3-emulation of the ring R on the tree T, and hence on H. Consider the procedure *BuildPath(v, endpath$_v$)* given below. The procedure constructs a path in T^3 starting in node v and visiting all the nodes of T_v (the subtree of the tree T with v as root), and ending in a son of v in T or in v itself if v has no son. This end of the path in T_v is kept in the variable *endpath$_v$*.

```
procedure BuildPath(v, endpath_v)
  begin
    if v has sons
    then for all sons v_i of v in T (i ∈ {1,..., deg(v)})
         do
           BuildPath(v_i, endpath_{v_i}) ;
           if i < deg(v)
           then add (endpath_{v_i}, v_{i+1}) to E_R
           endif
         od ;
         add (v, endpath_{v_deg(v)}) to E_R ;
         endpath_v := v_1
    else {v has no sons}
         endpath_v := v
    endif
  end {BuildPath};

begin {Main}
  initialize E_R to φ ;
  BuildPath(root(T), endpath_{root(T)}) ;
  add (root(T), endpath_{root(T)}) to E_R
  {E_R is the set of edges of a virtual ring in T}
end {Main}.
```

In the figure below an example of a virtual ring constructed by the previous procedure is shown. The continuous lines represent the edges in the given tree, whereas the dotted lines represent the virtual edges, i.e., edges of the ring.

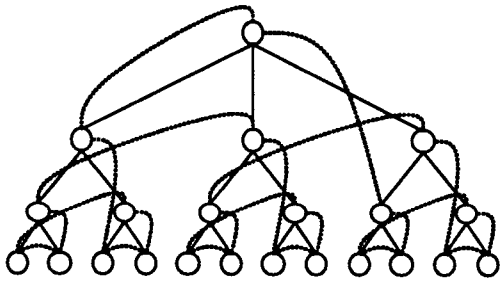


Figure 2.11

Consider the procedure BuildPath. Observe that once endpath_v has gotten a value, the node it denotes lies in T_v and has distance ≤ 1 from v. Thus it is clear that if this procedure adds (v, w) to E_R, then $d_H(v, w) \leq 3$. The only thing that remains to prove is that the graph R that has been constructed is indeed a ring that spans H. This is done by induction on the number of levels of the tree T. We claim that the procedure constructs a Hamiltonian path on T that starts in $\text{root}(T)$ and ends in a

son of root(T), or in root(T) itself if root(T) has no sons. If the tree consists of one or two levels it is easy to check that the claim is correct. Now assume it is correct for trees T with $\leq (L-1)$ levels. Let T be a tree with L levels with root r. Let T_i be the subtree of T with the son v_i of r as root, for every $i \in \{1,...,\deg(v)\}$. BuildPath(r, endpath$_r$) first executes BuildPath(v_i, endpath$_{v_i}$) for every i. Every tree T_i has $\leq (L-1)$ levels. Thus by induction BuildPath(v_i, endpath$_{v_i}$) builds a Hamiltonian path in T_i starting in v_i and ending in endpath$_{v_i}$. For every i the Hamiltonian path of T_i is appended to the Hamiltonian path of T_{i-1}. Hence we get a Hamiltonian path through the subtrees of r starting in v_1 end ending in endpath$_{v_{deg(v)}}$. Thus by connecting r to endpath$_{v_{deg(v)}}$ we obtain a Hamiltonian path of T starting in r, and ending at r. This proves the claim. The argument also shows that the 3-emulation is uniform. □

2.3 Polynomial Algorithms for Constructing Uniform d-Emulations of Rings (Remarks)

In [F74] it is shown that the square of every 2-connected graph is Hamiltonian. Hence for every 2-connected graph there exists a uniform 2-emulation of a ring of the same size. To our knowledge a polynomial-time algorithm that constructs this uniform 2-emulation is not known.
In [FH76] characterizations of cacti and vertex-unicyclic graphs of which the squares are Hamiltonian are given. Therefrom it is easy to obtain polynomial-time algorithms that determine whether a uniform 2-emulation of a ring on a cactus or a vertex-unicyclic graph exists and, if this is the case, construct this uniform 2-emulation. In [T56] it is shown that every 4-connected planar graph is Hamiltonian. In [G82] a polynomial-time algorithm is given that constructs a Hamiltonian circuit for every given 4-connected planar graph. For graphs with a given tree-decomposition with bounded tree-width HAMILTONIAN CIRCUIT is solvable in linear time [AP89]. If a graph with a given tree-decomposition with bounded tree-width is Hamiltonian, then it is possible to construct the Hamiltonian circuit in linear time [B91].

3 Distributed Sequential Traversal Algorithms and Virtual Ring Construction

In this section we focus attention on communication networks and distributed algorithms. Distributed sequential traversal algorithms are very suitable as a basis for a virtual ring construction algorithm. They define a total ordering on the nodes of the communication network, which can be used to construct a virtual ring. Thus, for example, a distributed depth-first search algorithm (DDFS algorithm for short) [SIM89] can be used as a basis for a distributed virtual ring construction algorithm

(DVRC algorithm for short.). This was done in [HR87]. The algorithm presented in that paper uses $2(|V|-1)$ messages of $O(|V|)$ bits, where V is the set of nodes of the communication network. The virtual ring is established by computing the appropriate routing information at the nodes of the network. Also an improved version of this algorithm was presented that allowed the construction of a virtual ring whose traversal requires only p messages, where $|V| \leq p \leq 2(|V|-1)$. However, the virtual ring R constructed by the algorithms in [HR87] is not necessarily a uniform 3-emulation of the graph representing the given network, and a virtual link may span as many as $(|V|-1)$ physical links. The purpose of building the virtual ring is to be able to *simulate* distributed algorithms that are designed for rings on arbitrary communication networks. If the DVRC algorithm of [HR87] is used, the simulation may cost up to $(|V|-1)$ times as many messages, if the original algorithm concentrates message traffic on virtual links which are mapped on $(|V|-1)$ physical links. In contrast with this, the simulation on a virtual ring that is uniformly 3-emulated on the network will cost only up to three times as many messages regardless of the distribution of the messages over the links. Furthermore, it can be noted that it is not necessary to use an exact DFS traversal. In fact, any enumeration of the nodes of the network can be used to define the virtual ring. Therefore distributed breadth first search (DBFS) algorithms [ZC87] and even distributed spanning tree (DST) algorithms can be used as a bases for the DVRC algorithm as well. In this section a DVRC algorithm is described that constructs a uniform 3-emulation of a ring R on the graph G representing the underlying network. It uses a distributed spanning tree algorithm of Segall [S83] as a basis. Segall's algorithm is a single-initiator DST algorithm that does not necessarily construct a DFS or BFS spanning tree. It is as time-efficient as the DDFS- and DBFS-algorithms but it uses less messages. The DST algorithm is an optimized version of Chang's Echo algorithm (see [C82]), which is a centralized (i.e., single-initiator) total algorithm for bidirectional networks. A *total algorithm* is an algorithm where all nodes of the network are required to participate before a decision can be taken. (For a more formal definition of total algorithms the reader is referred to [T91].) In [T91] it is shown that total algorithms are rather universal, in the sense that by adding extra control information to the messages of a total algorithm various other network problems can be solved (basically by the same flow of control). The DVRC algorithm that we develop is another demonstration of this fact and shows that total algorithms can effectively be used as a building block in the design of a large number of distributed algorithms (cf. [T91]).

3.1 The Network Model

We consider communication networks consisting of n nodes. It is assumed that the nodes are interconnected via bidirectional communication links. Every two nodes connected by a link will be able to communicate directly in a fault-free manner, i.e., messages will arrive unaltered after an arbitrary but finite delay. It is not necessary in our model that links obey the FIFO rule. It is assumed that each node

of a communication network can only distinguish its incident communication links and does not know, at the start of the DVRC algorithm for sure, the identities of its neighboring nodes. Furthermore it is assumed that there exists a leader of the network. The communication network is modeled by the graph $G = (V, E)$ where V models the set of nodes and E the set of communication links.

The complexity of a distributed algorithm is measured by its communication complexity and its time complexity. The *communication complexity* of a distributed algorithm is the total number of messages sent during the execution of the algorithm. The *time complexity* is the maximum time elapsed from the beginning to the termination of the algorithm, assuming that delivering a message over a link requires at most one unit of time and that receiving a message, local processing, and sending it over a link require negligible time. Without this assumption the algorithm must still operate correctly.

3.2 A Distributed Virtual Ring Construction Algorithm

In this section we will describe a *Distributed Virtual Ring Construction Algorithm.* The algorithm consists of two phases. In the first phase a spanning tree is built such that each node in the tree knows the parity of its level in the tree. In fact this is Segall's DST algorithm [S83], with some minor extensions. In the second phase the parity information is used to establish the necessary routing information at each node of the network to implement the virtual ring. We now present the DVRC algorithm, called *Algorithm VR* (the first and second phase of Algorithm VR are called *Algorithm* VR_1 and *Algorithm* VR_2, respectively). We first describe the messages used by Algorithm VR_1 and the variables kept at every node of the network before we give routines for the nodes. Algorithm VR is message-driven, i.e., the actions at the various nodes are initiated by received messages.

Messages used by Algorithm VR_1:

- <flip>,<flop>: messages sent by a node to a possible son conveying information to establish the routing information that implements the virtual ring.
- <ack>: message sent to acknowledge a received <flip> or <flop> message, thereby claiming the sender of this <flip>/<flop> message as the receiver's father in the spanning tree.

Variables kept at every node v:

- adj(v): the set of links adjacent to v. (In the *special* node v_0 that initiates the algorithm a special internal link e_{init} to itself is added too.)
- status(v, e): the status of the adjacent link e. Status values belong to the set $\{\textbf{nil}, \textbf{father}, \textbf{son}, \textbf{reject}, \textbf{sentf}\}$ and are initially **nil** for all $e \in$ adj(v).

- lfather(v): the adjacent link leading to the father of v in the spanning tree, initially **nil**.
- level(v): gives the parity of the level of v in the spanning tree. Level values belong to the set {**odd**, **even**}.

Initialization of Algorithm VR_1:

A special node v_0 triggers the algorithm by sending a <flip> message to itself over an internal link e_{init}.

Atomic actions at node v:

Upon receipt of <flip> or <flop> **over link** e
do
if lfather(v) = **nil**
then status(v, e) := **father** ;
lfather(v) := e ;
if <flip> received
then level(v) := **odd** ;
for all links e' with status(v, e') = **nil**
do
send <flop> over link e' ;
status(v, e') := **sentf**
od
else level(v) := **even** ;
for all links e' with status(v, e') = **nil**
do
send <flip> over link e' ;
status(v, e') := **sentf**
od
endif ;
elif status(v, e) = **sentf**
then status(v, e) := **reject**
endif ;
if for every $e \in$ adj(v): status(v, e) $\in$ {**reject**, **father**, **son**}
then send <ack> over link lfather(v)
endif
od

Upon receipt of <ack> **over link** e
do
status(v, e) := **son** ;

```
    if for every e ∈ adj(v): status(v, e) ∈ {reject, father, son}
    then if e ≠ e_init
            then send <ack> over link lfather(v)
            else finished
            endif
    endif
od
```

Description of Algorithm VR_1:
Initially every node of the network is inactive. A node v becomes active when it receives a <flip> or <flop> message for the first time. As father(v) is **nil**, the link over which the message is received will be marked as father link. An <ack> message is sent over this link if v has received a message of all its neighbors. Furthermore, v determines the parity of its level in the spanning tree and sends a <flip> or <flop> message to all its other neighbors if its level is even or odd, respectively. These other neighbors are potential sons in the spanning tree. The status of the links to these neighbors is set to **sentf**. If v receives another <flip> or <flop> message of a neighbor, then it knows this neighbor cannot be a son in the spanning tree. The status is suitably adapted, i.e., set to **reject**. If v receives an <ack> message of a neighbor, then this neighbor can be marked as son. If v has received a message from all its neighbors and has sent an <ack> message to its father, it can resume the second phase of Algorithm VR_1.

The number of messages needed in the first phase of Algorithm VR_1 is equal to $2|E|$. The number of time units used by Algorithm VR_1 is less or equal than 2.depth(constructed spanning tree) $\leq 2(|V|-1)$.

3.3 Routing Information

In the first phase of Algorithm VR_1 a spanning tree is constructed in which every node knows the parity of its level. We will show that this information suffices to route messages in one direction of the virtual ring, provided that every message contains a bit that indicates the direction in which the message should travel. After a node finishes its role in the Algorithm VR_1, i.e., has received a message of all its neighbours and sent an <ack> message to its father, it executes the following algorithm (Algorithm VR_2) to determine its predecessor and successor in the virtual ring.

The variables used by Algorithm VR_2:

Every node $v \in V$ holds the following variables:

- succ: will contain the name of the link over which a message is sent to /received from the successor of v in R.
- pred: will contain the link over which a message is sent to /received from the predecessor of v in R.
- predlink(e): the link over which a message will be forwarded if it was received over link e and travels the ring in reverse-order.
- succlink(e): the link over which a message will be forwarded if it was received over link e and travels the ring in-order.

Algorithm VR_2 at node v:

order all links e of node $v \neq$ lfather(v) with status(v, e) = **son** ;
let $e_1, e_2, ..., e_k$ be these links in order.
if level(v) = **odd**
then if lfather(v) = v {v is root}
then succ := e_1 ; pred := e_k ;
else succ := e_1 ; pred := lfather(v) ;
succlink(e_k) := lfather(v) ;
predlink(lfather(v)) := e_k
endif
else succ := lfather(v) ; pred := e_k ;
succlink(lfather(v)) := e_1 ;
predlink(e_1) := lfather(v)
endif ;
forall $i \in \{1, .., k-1\}$
do
succlink(e_i) := e_{i+1} ;
predlink(e_{i+1}) := e_i
od

Description of Algorithm VR_2:
Note that after a node v finishes its role in Algorithm VR_1 it knows the parity of its level in the constructed spanning tree. This information is used to create an oriented virtual ring in the network. The parity of its level determines over which links it can reach its successor and predecessor in the virtual ring, this information will be stored in the variables *succ* and *pred*, respectively. Also it determines when to forward messages not addressed to it. This information will be stored in the variables *predlink()* and *succlink()*. If a message not addressed to v, is received over link e, then *predlink*(e) and *succlink*(e) are equal to the links over which the message must be forwarded in order to reach its destination if the message travels the ring in reverse-order and in-order, respectively.

If level(v) is **odd**, then the part of the virtual ring that is created in the subtree T_v rooted by v starts in v, proceeds via link e_1 through the various subtrees rooted by the sons of v and ends in the son of v incident to link e_k. If level(v) is **even**, then the part of the virtual ring that is created in T_v starts in the son of v incident to link e_1, proceeds through the various subtrees rooted by the sons of v and ends in v. It is easy to verify that the variables kept at a node are set to the appropriate values. At the root of the tree T the virtual ring is closed, i.e., $\text{succ}_{root(T)} = e_1$ and $\text{pred}_{root(T)} = e_k$.

Many distributed algorithms on a ring network demand a sequential traversal of all the nodes by a token. To implement a ring traversal every node executes the following "routing" algorithm. A message is routed over the virtual ring through physical links. It is assumed that the message which traverses the virtual ring holds an extra bit *direction* that indicates in which direction the ring is traversed. If *direction* is equal to *in-order*, then the virtual ring is traversed in, say, clockwise order. If *direction* is equal to *reverse-order*, then the virtual ring is traversed in counterclockwise order.

Routing algorithm at node v:
{In order to route a message over the virtual link through physical links.}

```
Upon receipt of a message <direction,"other info"> over link e
 do
   if direction = in-order
   then
        if e = pred
        then {v is end-node of the virtual link over which the message is sent}
             message is addressed to v ;
             process the message and continue with further actions
             {the message can be sent to the succesor of v over link succ}
        else {v is an intermediate node of the virtual link over which the message is sent}
             send message over link succlink(e)
        endif
   else {direction = reverse-order}
        if e = succ
        then {v is end-node of the virtual link over which the message is sent}
             message is addressed to v ;
             process the message and continue with further actions
             {the message can be sent to the succesor of v over link pred}
        else {v is an intermediate node of the virtual link over which the message is sent}
             send message over link predlink(e)
        endif
   endif
 od
```

Description of the routing algorithm:
In Algorithm VR_2 the links over which v can reach its successor and predecessor in the virtual ring are determined and stored in the variables *succ* and *pred*, respectively. If a message is received over the link equal to *succ* and the message travels the ring in-order, then v knows the message is addressed to it. Similarly, messages traveling the ring in reverse-order received over the link equal to *pred* are addressed to v. In all other cases the received message is not addressed to v and must be forwarded. If a message is received over link e that is not addressed to v, then the message is forwarded over predlink(e) and succlink(e) if the message travels the ring in reverse-order and in-order, respectively.

3.4 Correctness of the DVRC Algorithm

As mentioned before, Algorithm VR is an adapted version of Segall's centralized DST-Algorithm. With some minor changes to Segall's correctness proof, a correctness proof can be obtained for the first phase of the DVRC Algorithm (Algorithm VR_1). With the proof of Theorem 2.7 it can easily be verified that the routing information obtained in the second phase of the algorithm (Algorithm VR_2) is correct as well.

3.5 Results

With the previous results the next theorem should be clear.

Theorem 3.1 *Given a communication network modelled by a graph $G = (V, E)$ with properties as described in Subsection 3.1, we can construct a virtual ring on G and determine the appropriate routing information per node by a distributed single-initiator algorithm that uses O(depth(constructed spanning tree)) time-units and $2|E|$ messages of O(1) bits.*

Corollary 3.2 *Every distributed algorithm on a ring network with properties as described in Subsection 3.1 can be simulated on an arbitrary communication network with the same properties and the same number of nodes at the expense of three times as many messages, and $2|E|$ messages of O(1) bits and O(depth(constructed spanning tree)) time for preprocessing to construct the virtual ring.*

Finally, we remark that in [RFH72] a VRCA is formulated as a network of finite automata. Here the constructed virtual ring is also a uniform 3-emulation. However, the algorithm works sequentially and requires $O(E)$ time.

4 Acknowledgements

We thank Hans Bodlaender and Gerard Tel for useful comments.

References

[A85] S.A. Andreasson. *Minimizing a Virtual Control Token Ring.* In: E. Gafni, N. Santoro (Eds.), Distributed Algorithms on Graphs, Proc. 1st International Workshop on Distributed Algorithms on graphs (1985), Carleton University Press, Ottawa, 1986.

[AP89] S. Arnborg, A. Proskurowski. *Linear Time Algorithms for NP-Hard Problems on Graphs Embedded in k-Trees.* Discrete Applied Math., Vol. 23, 1989, pp. 11-24.

[B86] H.L. Bodlaender. *Distributed Computing-Structure and Complexity.* Ph.D. Thesis, Dept. of Computer Science, Utrecht University, November 1986.

[B91] H.L. Bodlaender. *Private communication*, 1991.

[BL86] H.L. Bodlaender, J. van Leeuwen. *Simulation of Large Networks on Smaller Networks.* Information and Control, Vol. 71, December 1986, pp. 143-180.

[C82] E.J.H. Chang. *Echo Algorithms: Depth Parallel Operations on General Graphs.* IEEE Trans. Software Eng., SE-8, 1982, pp. 391-401.

[C76] V. Chvátal, private communication, cited in [GJ79], 1976.

[FF82] J.P. Fishburn, R.A. Finkel. *Quotient Networks.* IEEE Transactions on Computers, C-31, 1982, pp. 288-295.

[F74] H. Fleischner. *The Square of Every Two-Connected Graph is Hamiltonian.* Journal of Combinatorial Theory (B), Vol. 16, 1974, pp. 29-34.

[FH76] H. Fleischner, A.M. Hobbs. *Hamiltonian Cycles in Squares of Vertex-Unicyclic Graphs.* Canadian Math. Bull., Vol. 19, 1976, pp. 169-172.

[GJ79] M.R. Garey, D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman, San Francisco, California, 1979.

[GJT76] M.R. Garey, D.S. Johnson, R.E. Tarjan. *The Planar Hamiltonian Circuit Problem is NP-Complete.* SIAM J. Comput., Vol. 5, 1976, pp. 704-714.

[G82] D. Gouyou-Beauchamps. *The Hamiltonian Circuit Problem is Polynomial for 4-Connected Planar Graphs.* SIAM J. Comput., Vol. 11, August 1982, pp. 529-539.

[H69] F. Harary. *Graph Theory.* Addison-Wesley Publ. Comp., Reading, Mass., 1969.

[HR87] J. Helary, M. Raynal. *Depth-First Traversal and Virtual Ring Construction in Distributed Systems.* Technical Report No. 396, IRISA, Rennes, June 1987.

[HR88] J. Helary, M. Raynal. *Virtual Ring Construction in Parallel Distributed Systems.* Proceedings of the IFIP WG 10.3 Working Conference on Parallel Processing, Pisa, Italy, 25-27 April 1988, pp. 333-345.

[IPS82] A. Itai, C.H. Papadimitriou, J.L. Szwarcfiter. *Hamiltonian Paths in Grid Graphs.* SIAM J. Comput., Vol. 11, November 1982, pp. 676-686.

[K68] J.J. Karaganis. *On the Cube of a Graph.* Canad. Math. Bull, Vol. 11, 1968, pp. 295-296.

[M89] F. Mattern. *Verteilte Basisalgorithmen.* Informatik-Fachberichte, Vol. 226, Springer-Verlag, Berlin, 1989.

[P79] J. Plesnik. *The NP-Completeness of the Hamiltonian Cycle Problem in Planar Digraphs with Degree Bound Two.* Information Processing Letters, Vol. 8, 1979, pp. 199-201.

[RFH72] P. Rosenstiehl, J.R. Fiksel, A. Holloger. *Intelligent Graphs: Networks of Finite Automata Capable of Solving Graph Problems.* In: R.C. Read (Ed.), Graph Theory and Computing, Academic Press, New York, 1972.

[S83] A. Segall. *Distributed Network Protocols.* IEEE Trans. Information Theory, IT-29, 1983, pp. 23-35.

[SIM89] M.B. Sharma, S.S. Iyengar, N.K. Mandyam. *An Efficient Distributed Depth-First Search Algorithm.* Information Processing Letters, Vol. 32, 1989, pp. 183-186.

[SK87] M. Sloman, J. Kramer. *Distributed Systems and Computer Networks.* Prentice-Hall Inc., Englewood Cliffs, N.J., 1987.

[T91] G. Tel. *Topics in Distributed Algorithms.* Cambridge University Press, 1991.

[T56] W.T. Tutte. *A Theorem on Planar Graphs.* Trans. Amer. Math. Soc., Vol. 82, 1956, pp. 99-116.

[ZC87] Y. Zhu, T. Cheung. *A New Distributed Breadth-First-Search Algorithm.* Information Processing letters, Vol. 25, 1987, pp. 329-333.