Optimal Algorithms for Packet Routing on Trees

Antonios Symvonis
Basser Department of Computer Science
University of Sydney
Sydney, N.S.W. 2006
Australia
symvonis@cs.su.oz.au

Abstract

In this paper, we study the permutation packet routing problem on trees. We show that every permutation can be routed on a tree of n vertices in $n-1$ routing steps. We provide an algorithm which produces in $O(n^2)$ time a schedule that needs $O(n^2)$ bits for its description. Moreover, we describe an on-line algorithm that completes the routing of any permutation in $n-1$ routing steps by using at each vertex v buffering area of size at most $2d(v)$, where $d(v)$ is the degree of vertex v. Our results provide upper bounds on the number of routing steps required to route a permutation on an arbitrary connected graph G since the routing can be done by using only the edges of a spanning tree of G.

Proc. ICCI'94, 144-161 © 1994 Int. Conf. on Computing and Information

1 Introduction

The *permutation packet routing problem* on a connected undirected graph is the following: We are given a graph $G = (V, E)$ and a permutation π of the vertices of G. Every vertex v of G contains a packet destined for $\pi(v)$. Our task is to route all packets to their destinations.

During the routing, the movement of the packets follows a set of rules. These rules specify the *routing model*. Routing models might differ on the way edges are treated (unidirectional, bidirectional), the number of packets a vertex can receive or transmit in a single step, the number of packets allowed to queue in a vertex (queuesize), etc.

Let $rt_M(G, \pi)$ be the number of steps required to route permutation π on graph G using routing model M. The routing number of graph G with respect to routing model M, $rt_M(G)$, is defined to be the

$$rt_M(G) = \max_{\pi} rt_M(G, \pi)$$

over all permutations π of the vertex set V of G.

The routing number of a graph was first defined by Alon, Chung and Graham in [1]. In their routing model, the only operation allowed during the routing is the exchange of the packets at the endpoints of an edge of graph G. The exchange of the packets at the endpoints of a set of disjoint edges (a matching on G) can occur in one routing step. We refer to this model as the *matching routing model* and, for a graph G, we refer to the routing number of G with respect to the matching routing model, simply as the routing number of G, denoted by $rt(G)$. In [1], it was shown that $rt(T) < 3n$ for any tree T of n vertices. As a consequence, $rt(G) < 3n$ for any graph G of n vertices.

A lot of work has been devoted to the study of packet routing problems. As it is natural, several routing models have been considered. However, most of the papers

in the literature [2, 5, 6, 7, 8, 9, 11, 12, 13, 15, 17] consider the model in which, at any time step, all edges can carry a packet (bidirectional edges can carry one packet for each direction). Upfal [16] considered the model in which, at each step, each processor can either send or receive and only along one communication link. Meyer auf der Heide, Oesterdiekhoff and Wanka [10] considered the model in which each processor can receive at most one packet per step. Note that, the above list of references is in no way complete.

In this paper, we consider the commonly used routing model in which, at any time step, all edges can carry at most one packet in each direction. Moreover, at each time step, at most $d(v)$ packets can be found in any vertex v of degree $d(v)$ and no pair of packets competes for the same communication link. A different way to describe this restriction is to say that with each communication link there is associated a buffer that can hold at most one packet. The routing schedule has to assure that this buffer is never overloaded. Since there is no chance that two packets will compete for the same edge, it is fair to say that during the routing queues are not created. We will refer to it as the *simplified routing model* and we will denote the routing number of graph G with respect to the simplified routing model by $rt'(G)$.

The rest of the paper is organized as follows: In Section 2, we give definitions for terms we use in the paper. In Section 3, we show that $rt'(T) < n$ for any tree $T = (V, E)$ of n vertices. We achieve this upper bound by demonstrating an algorithm that computes a routing schedule of at most $n - 1$ steps for any permutation over the vertex set of T. The routing schedule requires $O(n^2)$ bits for its description and is computed in $O(n^2)$ time. In Section 4, an on-line algorithm that achieves the same number of routing steps is derived. The on-line algorithm uses at each vertex v buffering area of size at most $2d(v)$ where $d(v)$ is the degree of vertex v. We conclude in Section 5 with further research that has to be done in this area. Because of space limitations we have omitted all proofs. These proofs can be found in [14].

2 Preliminaries

A *finite directed graph* $G = (V, E)$ is a structure which consists of a finite set of vertices V and a finite set of edges $E = \{e_1, e_2, \ldots, e_{|E|}\}$. Each edge is incident to the elements of an ordered pair of vertices (u, v). u is the *start-vertex* of the edge and v is its *end-vertex*. Occasionally, we refer to the vertex and the edge sets of graph G by $V(G)$ and $E(G)$, respectively.

Edges with the same start and end-vertices are called *self-loops*. We define the directed *self-loop augmented graph* $G^{SL} = (V, E')$ of $G = (V, E)$ to be the graph with $E' = E \cup \{e^v = (v, v) | v \in V\}$ (one self loop is added for each vertex in G provided that it does not already exist).

The set $Neighbors(v, G)$ is defined to be the set of vertices in G that can be reached from v by crossing just one edge. Formally, $Neighbors(v, G) = \{w |\ (v, w) \in E \text{ of } G\}$.

If we ignore the direction of the edges of a finite directed graph, we get a *finite undirected graph*. A *tree* is an undirected graph with no cycles and exactly $|V| - 1$ edges.

Given an undirected graph G, we can transform it to a directed one by substituting each edge (u, v) in $E(G)$ by the pair of *anti-parallel* edges (u, v) and (v, u). We denote the graph produced by the above transformation by G^D.

An *permutation packet routing problem* R is defined to be the pair (G, π) where $G = (V, E)$ is the directed graph that represents the network in which the routing will take place (vertices in V represent processors and edges in E represent unidirectional communication links) and π is the permutation to be routed. Formally, the set P of $|V|$ packets to be routed is defined by $P = \{p_1, p_2, \ldots, p_{|V|} |\ p_i = (i, \pi(i)),\ i, \pi(i) \in V,\ 1 \leq i \leq |V|\}$. A more general definition that incorporates the maximum allowed queuesize was given in [15]. Note that, even though the informal definition of most routing models involves an undirected graph G with bidirectional communication

links, the corresponding directed graph G^D can be used in the formal definition of the routing problem.

An *off-line solution* (or *routing schedule*) of *length* L for the off-line packet routing problem $R = (G, \pi)$ is a set of directed paths $SOLUTION(R) = \{d_1, d_2, \ldots, d_{|V|}\}$ where d_i is the directed path corresponding to packet p_i. The paths are taken on graph G^{SL}, the self-loop augmented graph of G, instead of G. We do that in order to make it possible to incorporate self loops in the directed paths. A self loop from vertex v in the path of some packet indicates that the packet was not advanced at the corresponding routing step. Each directed path contains at most $L + 1$ vertices. For $i = 1 \ldots |V|$ we have that

$$d_i = v_i^0 v_i^1 \ldots v_i^l,\ 0 \leq l \leq L$$

where, $v_i^0 = i$ and $v_i^l = \pi(i)$.

In order to have a valid solution for our routing problem, the directed paths must satisfy the condition: *"At any routing step, each edge that corresponds to an unidirected communication link appears in at most one directed path."*

In order to describe the solution of a permutation routing problem we need to specify for each packet the path it follows during the routing. At the worst case, a solution of length L can be represented by a two dimensional matrix $SOLUTION$ of $|V|$ rows (one for each packet) and $L + 1$ columns (one for each vertex in the path). $SOLUTION[p, t]$, $0, \leq t \leq L$, is the vertex in which packet p is after t routing steps. The space needed for reporting the solution is $O(|V|L \log |V|)$ bits. However, in cases in which there is a unique path between any pair of vertices of the underlying graph and derouting is not allowed, a path can be determined simply by knowing if at a given step the packet is advanced towards its destination. In this case, each entry of matrix $SOLUTION$ consists of a single bit and thus, space of $O(|V|L)$ bits is sufficient. Furthermore, if it also holds that the movement of the packets is uninterupted, then

only the step in which each packet starts its routing needs to be stored. In that case, $O(|V| \log L)$ bits are sufficient for reporting the routing schedule.

One important property that trees possess is that there is a unique simple path between any pair of vertices in the tree. Given a tree T, we denote the unique path in T from vertex u to vertex v by $path(u, v)$. The number of edges in $path(u, v)$ is denoted by $path\_size(u, v)$. We assume that, if w is a vertex in $path(u, v)$, we can determine the vertex that is immediately after w in the path from u to v in constant time. It is easy to do so by using a $|V| \times |V|$ matrix N such that $N[u, v]$ contains the first vertex (not including u) of $path(u, v)$. Of course, some preprocessing is necessary to initialize matrix N and several algorithms for doing so are available. Obvious choises include shortest path algorithms and tree traversal techniques [3] which can optimally initialize matrix N in $O(|V|^2)$ time. In the rest of the paper, we assume that the information of matrix N is available to us.

3 The routing number of trees with respect to the simplified routing model

In this section, we show that the routing number of a tree T of n vertices with respect to the simplified routing model is bounded from above by n, i.e., $rt'(T) < n$. We prove this bound by excibiting an algorithm that, given a tree T of n vertices and a permutation π on T's vertex set, produces a routing schedule for the permutation problem (T^D, π) of length at most $n - 1$. The routing schedule is produced in $O(n^2)$ time and requires $O(n^2)$ bits for its description.

3.1 The routing graph

Before we proceed with the description of the algorithm, we need to define some notation. Let $T = (V, E)$ be the tree in which the routing will take place. V is the vertex set of T and E its edge set. It holds that $|E| = |V| - 1$.

For each vertex $v \in V$ we construct the set

$$S_v = \{v_u \mid u \in Neighbors(v, T)\} \cup v_{\text{con}}$$

("con" stands for "consume"). The set $V^R = \bigcup_{v \in V} S_v$ will be the vertex set of an auxiliary directed graph $T^R = (V^R, E^R)$ which we will use in the algorithm. We call T^R the *routing graph*. The edge set of the routing graph will be different at each stage of the off-line routing algorithm. We denote by $T_i^R = (V^R, E_i^R)$ the routing graph at stage i. E_i^R is the edge set of T_i^R.

During the course of the routing, we denote by $current(p)$ the current position of packet p while, by $orig(p)$ we denote its origin and by $dest(p)$ we denote its destination. Since the routing is happening on a tree, for each packet p, there is a unique simple (with no repeating vertices) path $path(current(p), dest(p))$ from $current(p)$ to $dest(p)$. We denote by $f(p)$ the first vertex on this path (not including $current(p)$) and by $s(p)$ the second one. In the case that $s(p)$ and/or $f(p)$ are not well defined ($path(current(p), dest(p))$ is too short for $s(p)$ and/or $f(p)$ to have meaning), we assume that they return the special value "con" (for "consumed").

To define graph $T_i^R = (V^R, E_i^R)$ we simply have to specify E_i^R since V^R is fixed. E_i^R contains at most $|V|$ edges, one for each packet that hasn't reached its destination after i routing steps (stages). The edge that corresponds to packet p, denoted by $edge(p)$, is defined as follows:

$$edge(p) = \begin{cases} (current(p)_{f(p)}, f(p)_{s(p)}) & \text{if } f(p) \neq dest(p) \\ (current(p)_{f(p)}, f(p)_{\text{con}}) & \text{otherwise} \end{cases}$$

What we want to represent with each edge is the information that, if in this routing step packet p travels through edge $(current(p), f(p))$ of T, then in the next step it will compete for edge $(f(p), s(p))$ of T. An example of a routing graph is given in Figure 1.

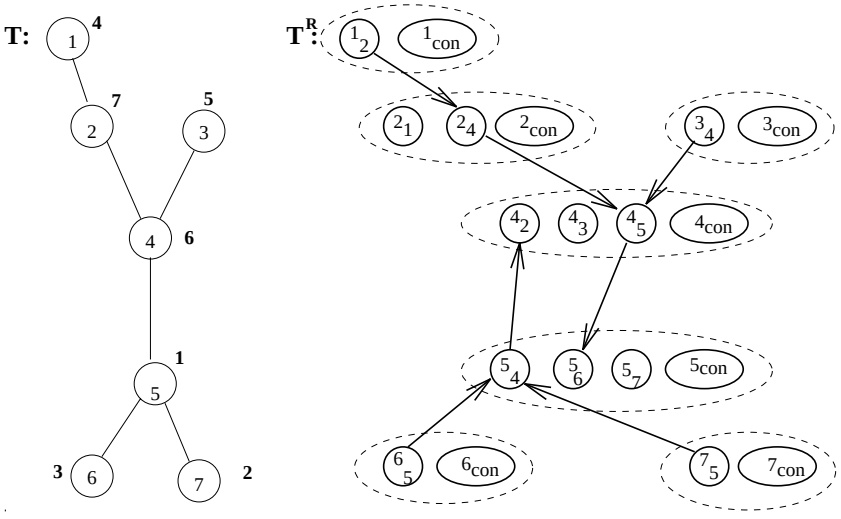

Figure 1: The numbers next to the vertices of the tree T represent the destination of the packet located in that vertex. Graph T^R is the routing graph which corresponds to tree T and the permutation to be routed.

The following lemmata regarding the routing graph are useful for the design and the time analysis of the off-line routing algorithm.

Lemma 1 *Let V^R be the vertex set of the routing graph constructed from tree $T = (V, E)$. Then, $|V^R| = 3|V| - 2$.* □

Lemma 2 *Assume a distribution of $|V|$ packets at the vertices of a tree $T = (V, E)$ which satisfies the requirement that no two packets compete for the same edge in a given direction i.e., there doesn't exist a pair of packets p and q such that $current(p) = current(q)$ and $f(p) = f(q)$. Then, the corresponding routing graph T^R consists of a collection of directed (toward the root) trees and a set of isolated vertices.* □

3.2 An $O(n^2)$-time off-line routing algorithm for trees

Consider any rooted at vertex v tree T_v which is a (not strongly connected) component of a routing graph T^R (see for example the tree rooted at vertex 5_6 in Figure 1). If at the first routing step we advance all the packets that correspond to the edges of the tree, then, at the second step, the edges of the original tree T that correspond to vertices of in-degree greater than 1, will be requested by more than one packet. To avoid this situation, we will not advance all packets. We will advance the packets that corresponds to only one path (arbitrary chosen) connecting v (the root of T_v) with one of the leaves of T_v. We can choose the packets which will move during the next step (and also notify the one that will not) by a simple traversal of the tree (in the opposite direction from that indicated by the edges) in $O(|V(T_v)|)$ time. We are now ready to present a high level description of the off-line routing algorithm.

Algorithm *Off_line_tree_routing(T, π)*
/* π is the permutation to be routed on tree T */

1. $i = 0$ /* i denotes the number of routing steps (stages) completed so far */
2. **While** there are still packets that haven't reach their destination **do**
   (a) Based on the current position of the packets (after i steps of routing) construct the routing graph T_i^R.

(b) Choose, based on the trees that form T_i^R, the packets that will move in step $i+1$.

(c) Move the packets, i.e., update the data structure that keeps track of the current position and the journey of each packet.

(d) $i = i + 1$

Since at each iteration of the while-loop of Algorithm *Off_line_tree_routing*(T, π) at least one packet is advanced towards its destination and no packet is routed away from its destination, the total distance that all packet have to travel always reduces. This implies that after at most $O(n^2)$ steps the routing is completed. In the following lemma, we provide a better upper bound on the number of times the while-loop in algorithm *Off_line_tree_routing*(T, π) is executed.

Lemma 3 *Assume a tree T of n vertices and a permutation π on its vertex set that has to be routed. Then, Algorithm* Off_line_tree_routing(T, π) *produces an off-line routing solution of at most $n-1$ routing steps.* □

Corollary 1 *For any tree T of n vertices, $rt'(T) < n$.* □

When we have to solve a routing problem on an arbitrary graph G instead of a tree, we can ignore all edges of G but those that form a spanning tree of G. We conclude that:

Corollary 2 *For any graph G of n vertices, $rt'(G) < n$.* □

From Lemma 1 we know that the number of vertices of the routing graph T^R that corresponds to a tree of n vertices is $3n-2$. Based on this, it is not difficult to implement each iteration of the while-loop in $O(n)$ time. Since Lemma 3 ensures that there

will be at most $n-1$ iterations we conclude that Algorithm *Off_line_tree_routing*(T, π) produces an off-line solution in $O(n^2)$ time.

At each routing step, each packet either advances towards its destination or waits at some vertex. So, the journey of each packet can be described by an array of 0/1 entries where, "0" denotes that the packet waits in a vertex while, "1" denotes that it moves. So, to report the solution of the routing problem we need space of size $O(n^2)$ bits.

The following theorem summarizes the results of this section.

Theorem 1 *Assume a tree T of n vertices and a permutation π on its vertex set that has to be routed. Then, Algorithm* Off_line_tree_routing(T, π) *produces an off-line routing solution of at most $n-1$ steps which can be described with $O(n^2)$ bits, in $O(n^2)$ time.* □

3.3 A more compact routing schedule

It is possible to produce a more compact routing schedule, i.e., one that requires $n \log n$ bits for its description. The algorithm that is used in deriving this compact solution is based on the *multistage off-line routing method* introduced by Symvonis and Tidswell in [15]. A detailed description of the algorithm can be found in [14]. The result is sumarized in the following theorem:

Theorem 2 *Assume a tree T of n vertices and a permutation π on its vertex set that has to be routed. Then, it is possible to produce an off-line routing solution of at most $n-1$ steps which can be described with $O(n \log n)$ bits, in $O(n^3)$ time. The routing model used assumes that each processor can store the packet that originates at it at no extra cost.*

4 An on-line routing algorithm

In this section, we present an on-line algorithm that completes the routing of a permutation on a tree of n vertices in at most $n-1$ steps. Denote by $d(v)$ the degree of vertex v. The algorithm requires a buffering area of size $2d(v)$ packets in each vertex v of the tree.

Consider first the trivial greedy on-line algorithm in which, at each time step, each processor tries to advance as many packets as possible towards their destination while the remaining packets are queued in the processor. It is not difficult to see that this simple algorithm will complete the routing in $n-1$ steps. At the same time, the queue at some processor might become as large as $(n/2)-2$. This can happen in the tree of Figure 2 when all the packets initially in the set (of processors) A are destined for the set (of processors) B and vice versa.

We first modify the above greedy algorithm and make it use queues of size at most $d^2(v)$ packets in any vertex v. For trees of maximum degree bounded by a constant, this yields an algorithm that uses constant size queues. A similar method was used by Symvonis and Makedon in [9] where an asymptotically optimal routing algorithm was developed for the *many-to-one* packet routing problem on 2-dimensional meshes. In each vertex v, we associate with each of the $d(v)$ outgoing communication links (edges) a queue of size $d(v)$. Consider any edge (v,u). At most $d(v)-1$ neighbors of v can send packets that want to cross edge (v,u). So, in order to avoid overloading the queue associated with edge (v,u) we allow the neighbors of v to send packets that want to cross that edge only when the associated queue contains at most 1 packet. We must inform the neighbors of v when the queue associated with (v,u) can receive packets. We do this by having, at each step, neighboring vertices exchanging information about the status of their queues. Vertex v has to transmit $d(v)$ bits of information to each of its neighbors. A "0" ("1") bit disables (enables) the transmission of packets

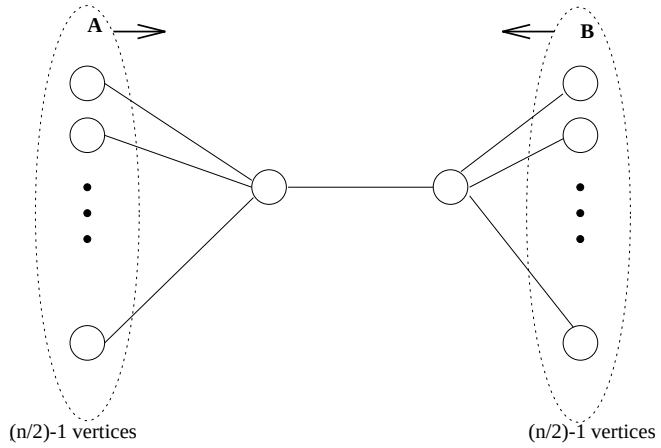

Figure 2: An example in which the queues created during the execution of the trivial greedy on-line routing algorithm can grow as large as $(n/2) - 2$ packets.

that want to cross a specific edge. This information has to be transmitted even when no packet has to cross the edge that connects two neighbors. In general, the messages exchanged between vertices consist of two fields; the original packet and the information regarding the status of the queues.

We now give a high level description of the on-line algorithm which uses queues of size $d^2(v)$.

Algorithm *On_line_tree_routing*(T, π)
/* π is the permutation to be routed on tree T */
For each processor, repeat until the routing of all packets is completed

1. From the information received in the previous step regarding the queues of its

neighbors, the processor decides which packets to transmit.

2. By checking the number of the remaining packets in each of its queues, the processor determines which of its queues can receive packets during the next step. Based on that information, a string of d bits is created[1].

3. The processor transmits the packets it selected in Step 1 concatenated with the binary string created in Step 2.

Lemma 4 *Assume a tree T of n vertices and a permutation π on its vertex set that has to be routed. Then, Algorithm* On_line_tree_routing(T, π) *completes the routing in at most $n-1$ steps and by using queues of size $d^2(v)$ at each vertex v.* □

Another possible method we can use for routing on trees is that of *hot-potato routing*. Assume that, at some time step, vertex v contains $l \leq d(v)$ packets. Then, using the hot-potato method, at the next routing step it sends as many as possible packets towards their destination while the rest of the packets are derouted (in an arbitrary manner) through the remaining communication links (since $l \leq d(v)$ there are enough communication links). The hot-potato method can be simply described with the phrase *"any packet entering a processor must get out or be consummed"*. Implementing the hot-potato method is easy. In contrast with Algorithm *On_line_tree_routing(T, π)* no additional information needs to be attached to transmitted packets. It is also easy to see that there exists a class of problems which require about $2n$ steps for their routing when the hot-potato method is used. The

[1] d is the maximum degree of the tree. Since the processor might have less than d neighbors, some of the bits might carry useless information.

routing problem of Figure 2 belongs in this class. On the other hand, one can show, in a similar way to that in Section 3.2, that $2n$ steps are enough for the routing.

The disturbing fact in Algorithm *On_line_tree_routing*(T, π) is that queues of size $d^2(v)$ are too large while, in the hot-potato method, we would like to reduce the number of routing steps. The idea of the hot-potato routing method, i.e., allowing derouting of packets, can be incorporated into Algorithm *On_line_tree_routing*(T, π) to reduce the queuesize to $2d(v)$ packets per processor. Now, in the bad event that all the packets which entered a processor want to exit from the same communication link, one packet will do so, one will be queued and the rest will be derouted. One packet is queued to ensure that derouting does not prolong the routing time. Thus, we have:

Theorem 3 *Assume a tree T of n vertices and a permutation π on its vertex set that has to be routed. Then, a modification of Algorithm* On_line_tree_routing(T, π) *(based on the idea of the hot-potato routing method) completes the routing in at most $n-1$ steps and by using queues of size $2d(v)$ at each vertex v.*

By performing the routing along the edges of a spanning tree of a graph G, we prove that:

Corollary 3 *Assume a graph G of n vertices and maximum degree d, and a permutation π on its vertex set that has to be routed. Then, π can be on-line routed in at most $n-1$ steps and by using queue of size at most $2d$ packets.*

The fact that any spanning tree of graph G can be used for the routing of a permutation on G, suggests the problem of finding a spanning tree in which the maximum degree is minimized. Unfortunately, this problem is NP-complete as it can be shown by an elementary reduction from the *hamiltonian path* problem [4].

5 Conclusions - further work

In this paper, motivated by the work of Alon, Chung and Graham [1], we considered the routing nuber of a tree T of n vertices with respect to the simplified routing model, denoted by $rt'(T)$. We proved that $rt'(T) \leq n-1$ by developing algorithms that compute routing schedules of different quality (with respect to the space required for the description of the routing schedule). This result is asymptotically optimal since for a chain of n vertices there exist a class of permutations that require $n-1$ steps for their routing.

An interesting problem we are currently working on is to design algorithms with performance close to the *actual lower bound* for the permutation on hands. Such lower bounds can be obtained based on combinations of distance and bisection arguments or, by using the multistage routing method introduced in [15] and the relation between the routing problem and the multicommodity flow problem.

Acknowledgement We thank Jop Sibeyen for suggesting the use of the hot-potato routing method to reduce the queuesize of the on-line routing algorithm.

References

[1] A. Alon, F.R.K. Chung and R.L. Graham, "Routing Permutations on Graphs via Matchings", in the Proceedings of the 25^{th} Annual ACM Symposium on the Theory of Computing, San Diego, California, May 1993, pp. 583-591. To appear in the SIAM Journal on Discrete Mathematics.

[2] A. Borodin and J.E. Hopcroft, "Routing, Merging , and Sorting on Parallel Models of Computation", *Journal of Computer and System Sciences*, Vol. 30, 1985, pp. 130-145.

[3] T.H Cormen, C.E. Leiserson and R.L Rivest, *Introduction to Algorithms*, The MIT Press, Cambridge, Massachusetts, 1990.

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

[5] C. Kaklamanis, Danny Krinzanc and Satish Rao, "Simple Path Selection for Optimal Routing on Processor Arrays", in the proceedings of the 4^{th} Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA), San Diego, California, June 1992, pp. 23-30.

[6] M. Kaufmann, S. Rajasekaran and J.F. Sibeyn, "Matching the Bisection Bound for Routing and Sorting on the Mesh", in the proceedings of the 4^{th} Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA), San Diego, California, June 1992, pp. 31-40.

[7] F.T. Leithon, B. Maggs and S. Rao, "Universal Packet Routing Algorithms", in the proceedings of the 29^{th} Annual Symposium of Foundations of Computer Science, October 1988, pp. 256-271.

[8] F.T. Leighton, F. Makedon and I.G. Tollis, "A $2n-2$ Algorithm for Routing in an $n \times n$ Array With Constant Size Queues", in the proceedings of the 1^{st} Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA), June 1989, pp. 328-335.

[9] F. Makedon and A. Symvonis, "Optimal Algorithms for the Many-to-One Routing Problem on 2-Dimensional Meshes", to appear in the *Journal of Microprocessors and Microsystems*. An extended abstract appeared in the Proceedings of the 19^{th} Annual ACM Computer Science Conference, San Antonio, Texas, March 1991, pp. 280-288.

[10] F. Meyer auf de Heide, B. Oesterdiekhoff and R. Wanka, "Strongly Adaptive Token Distribution" in the proceedings of ICALP93, LNCS 700, pp. 398-409.

[11] I. Parberry, "An Optimal Time Bound for Oblivious Routing", *Algorithmica*, 1990, 5, pp. 243-250.

[12] D. Peleg and E. Upfal, "The Generalized Packet Routing Problem", *Theoretical Computer Science*, 53, 1987, pp. 281-293.

[13] D. Peleg and E. Upfal, "The Token Distribution Problem", *SIAM Journal on Computing,* Vol. 18, No. 2, 1989, pp. 229-243.

[14] A. Symvonis, "Routing on Trees", Technical Report 471, Basser Dept. of Computer Science, University of Sydney, January 1994. Available by anonymous ftp from ftp.cs.su.oz.au (directory pub/tr).

[15] A. Symvonis and J. Tidswell, "A New approach to Off-Line Packet Routing. Case Study: 2-Dimensional Meshes", Proceedings of the 1992 DAGS/PC Symposium, Dartmouth Institute for Advanced Graduate Studies in Parallel Computation, June 1992, Hanover, NH, USA, pp. 84-93.

[16] E. Upfal, "Efficient Schemes for Parallel Communication", in ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, August 1982, pp. 55-59.

[17] L.G. Valiant, "A Scheme for Fast Parallel Communication", *SIAM Journal on Computing,* Vol. 11, No. 2, 1982, pp. 350-361.