Dynamic Tree Routing under the "Matching with Consumption" Model*

Grammati E. Pantziou[1], Alan Roberts[2] and Antonis Symvonis[2]

[1] Computer Technology Institute, P.O. Box 1122, 26110 Patras, Greece
[2] Department of Computer Science, University of Sydney, N.S.W. 2006, Australia

Abstract. In this paper we present an extensive study of dynamic routing on trees under the "matching with consumption" routing model. We present an asymptotically optimal on-line algorithm which routes k packets to their destination within $d(k-1) + d \cdot dist$ routing steps, where d is the degree of tree T on which the routing takes place and $dist$ is the maximum distance any packet has to travel. We also present an off-line algorithm that solves the same problem within $2(k-1) + dist$ steps. The analysis of our algorithms is based on the establishment of a close relationship between the matching and the hot-potato routing models.

1 Introduction

In a *packet routing problem* on a connected undirected graph G we are given a collection of packets, each packet having an origin and a destination node, and we are asked to route them to their destinations as fast as possible. 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, the number of packets each node can receive/transmit/hold in a single step, the number of packets that are allowed to queue in a node (queue-size), etc.

When all packets are available at the beginning of the routing, we have a *static* routing problem, while, when it is possible to generate packets during the course of the routing we have a *dynamic* routing problem. When each node is the origin of at most h_1 packets and the destination of at most h_2 packets, we have an (h_1, h_2)*-routing* (or *many-to-many*) problem. In the case where $h_1 = 1$ and $h_2 > 1$ we have a *many-to-one* routing problem (*many* nodes send packets to *one* node); when $h_1 = h_2 = 1$ and the number of packets is (less than or) equal to the number of nodes of the graph we have a *(partial) permutation.*

The *matching model* was defined by Alon, Chung and Graham when they studied the routing of permutations [1]. In the matching model, each node initially holds exactly one packet and the only operation allowed during the routing is the exchange of the packets at the endpoints of an edge. The exchange of the packets at the endpoints of a set of disjoint edges can occur in a single routing step. These edges are said to be *active* during the routing step. When a packet reaches its destination node it is not consumed. Instead, it continues to participate in the routing until the time all the packets in the graph simultaneously reach their destination nodes.

* The work of Dr Pantziou was partly supported by the EEC ESPRIT Projects GEPPCOM (contract No. 9072) and ALCOM IT. The work of Dr Symvonis is supported by an ARC Institutional Grant. Email: pantziou@cti.gr, {alanr,symvonis}@cs.su.oz.au.

Alon, Chung and Graham [1] showed that any permutation on a tree of n nodes can be routed in at most $3n$ steps. Roberts, Symvonis and Zhang [13] reduced the number of steps to at most $2.3n$. Furthermore, for the special cases of bounded degree trees and complete d-ary trees of n nodes, they showed that routing terminates after $2n + o(n)$ and $n + o(n)$ steps, respectively. Zhang [14] and Høyer and Larsen [8] subsequently reduced the number of steps required to route a permutation on an arbitrary tree to $2n$. The only work related to on-line routing on trees consists of the study of sorting on linear arrays based on the odd-even transposition method [6].

In this paper, we consider the natural extension of the original model which allows for the consumption of packets. We refer to this routing model as the *matching with consumption model.* Krizanc and Zhang [10] independently considered many-to-one routing under the same model. For n-node trees, they showed that any many-to-one routing pattern can be routed in at most $9n$ steps and posed the question whether it is possible to complete the routing for that type of pattern in less than $4n$ steps. In this paper we answer their question to the affirmative.

Consider any $(h_1 - h_2)$-routing problem which has to be routed under the matching model. Even though at most h_1 packets originate from any given node v, initially at most one of them participates in the routing. The remaining packets which originate at node v are *injected* into the routing at times where v holds no other packet, i.e., at times when either no packet entered v or the packet which did so was consumed at v.

Another commonly used routing model is the *hot-potato* (or *deflection*) routing model in which packets continuously move between nodes from the time they are injected into the graph until they are consumed at their destination. This implies that i) at any time instance the number of packets present at any node is bounded by the out-degree of the node, and ii) at any routing step each node must transmit the packets it received during the previous step (unless they were destined for it). Because packets always move, it is not possible to always route all packets to nodes closer to their destination. At any given routing step several packets might be derouted away from their destination. This makes the analysis extremely difficult. Consequently, even though hot-potato routing algorithms have been around for several years [2], no detailed and non-trivial analysis of their routing time was available until recently.

The work of Feige and Raghavan[5] which provided analysis for hot-potato routing algorithms for the torus and the hypercube renewed the interest in hot-potato routing. As a result, several papers appeared with hot-potato routing as their main theme (see [9, 11] and the references therein). Borodin et al [3] formalised the notion of the *deflection sequence*, a nice way to charge each deflection of an individual packet to distinct packets participating in the routing. Among other results, they show that routing k packets in a hot-potato manner can be completed within $2(k-1) + dist$ steps for trees where $dist$ is the initial maximum distance a packet has to travel. A similar result was proven earlier by Hajek [7] and Brassil and Cruz [4] for hypercubes.

Due to space limitations, it is not possible to provide complete proofs for most of our results. Details can be found in [12].

2 Preliminaries

A *tree* $T = (V, E)$ is an undirected acyclic graph with node set V and edge set E. The nodes of V are supposed to be ordered according to some ordering criteria. Throughout the paper we assume n-node trees, i.e., $|V| = n$. An undirected edge connecting nodes

u and v is denoted by $\{u, v\}$, while a directed edge from node u to node v is denoted (u, v). The set of neighbours of node u is defined as $Neighbours(u) = \{v \mid \{u, v\} \in E\}$. The degree of node u is defined as $degree(u) = |Neighbours(u)|$. In a similar way we define the *in-degree* and the *out-degree* of a directed graph. For a graph $G = (V, E)$ and two nodes u, $v \in V$, we denote by $dist_T(u, v)$ the distance (i.e., the length of the shortest path) from u to v on G.

A static routing problem $\mathcal{R}$ can be defined to be a tuple $\mathcal{R} = (G, S)$ where G is the graph on which the routing takes place and S is the set of packets to be routed. Each packet $p \in S$ can be described by the tuple $p = (orig, dest)$ where $orig$ and $dest$ denote the origin and the destination of packet p, respectively. The notation $orig(p)$ and $dest(p)$ is also used to denote the origin and the destination of packet p. For simplicity, we assume that for every packet $p \in S$ it holds that $orig(p) \neq dest(p)$.

In the analysis of our algorithms for the matching model we are going to use the *"charging argument"* formulated by Borodin, Rabani, and Schieber [3] for the hot-potato routing model. Consider an arbitrary packet p which, at time t, is located at node v and, during the next routing step, moves away from its destination because all edges incident to node v which lead to nodes closer to the destination of p are used for the routing of other packets. In this case, we say that packet p suffers a *deflection* at time t and that any of the packets which move closer to the destination of p is responsible for (or *caused*) that deflection.

Borodin et al [3] defined the notions of the *deflection sequence* and the *deflection path* for a particular deflection as follows: Consider a deflection of a packet p at time t_1 and let p_1 be the packet which caused the deflection. Follow packet p_1 until time t_2 where it reaches its destination or it is deflected by packet p_2, whichever happens first. In the latter case, follow packet p_2 until time t_3 where it reaches its destination or it is deflected by packet p_3, and so on. We continue in this manner until we follow a packet p_l which reaches its destination at time t_{l+1}. The sequence of packets p_1, $p_2, \ldots,$ p_l is defined to be the *deflection sequence* corresponding to the deflection of packet p at time t_1. The path (starting from the deflection node and ending at the destination of p_l) which is defined by the deflection sequence is said to be the *deflection path* corresponding to the deflection of packet p at time t_1.

Lemma 1. **(Borodin, Rabani, Schieber [3])** *Suppose that for any deflection of packet p from node v to node u the shortest path from node u to the destination of p_l (the last packet in the deflection sequence) is at least as long as the deflection path. Then, p_l cannot be the last packet in any other deflection sequence of packet p. Consequently we can associate (or "charge") the deflection to packet p_l.*

Lemma 1 is quite useful in the analysis of hot-potato algorithms. Consider for example the case where the routing takes place on an undirected graph and the hot-potato algorithm sends a packet away from its destination only if all edges which lead closer to its destination are used by other packets which advance closer to their destinations. Let p be an arbitrary packet which initially is *dist* steps away from its destination and assume that k packets participate in the routing (including p). According to Lemma 1, each deflection of p can be associated (or charged to) with a distinct packet which also participates in the routing. Therefore, given that the total number of packets is k, packet p can be deflected at most $k - 1$ times. So, in the worst case, packet p spends $k - 1$ steps moving away from its destination, $k - 1$ steps negating the result of the deflections (recall that the graph in this example is undirected), and *dist* steps moving towards its destination. Thus, packet p reaches its destination within at most $2(k-1) + dist$ routing steps.

3 On-line Routing

In this section we consider on-line routing on n-node trees of maximum degree d. We prove a lower bound which applies to a natural class of algorithms and we provide an algorithm which matches it (asymptotically).

3.1 A Lower Bound

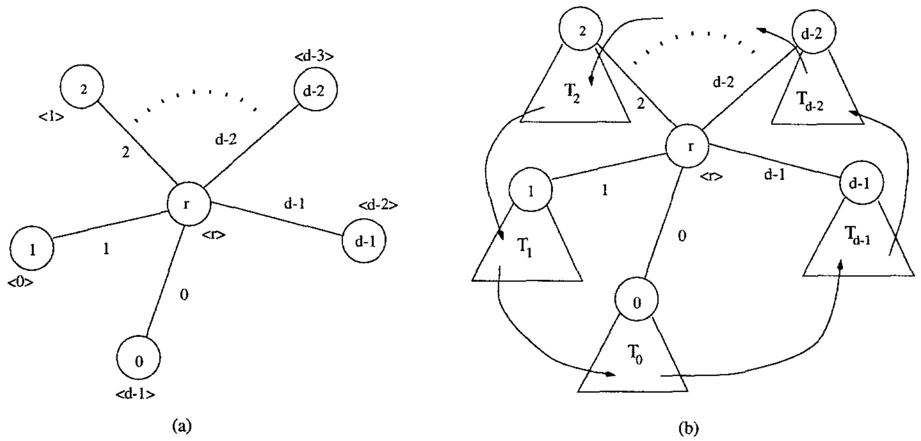

Fig. 1. Worst case permutations for (a) an n-node star of degree $d = n - 1$, and (b) a tree of maximum degree d, for some constant $d \geq 2$. The numbers in the nodes are node labels, the numbers attached to edges denote the order in which edges are activated and the numbers between angle brackets denote packets with a given destination.

In order to route a pattern under the matching model an on-line algorithm must on each step choose a matching. Once this matching has been chosen for a given step, the packets at the endpoints of each edge of the matching are compared and the decision to swap them is made depending on some rule. The on-line algorithms to which our bound applies are the ones in which the edges of each node are considered in a fixed order throughout the course of the routing. These algorithms repeatedly cycle through a fixed sequence of matchings making swapping decisions based on a deterministic rule.

Consider the permutation shown in Figure 1(a) for a *star* of degree d. We assume that the edges become active in increasing order of the labels attached to the edges of the star. Consider an arbitrary packet which originates at a node other than the centre of the star. Observe that any such packet has to spend at least $d-1$ steps at the centre of the star waiting for the edge that leads to its destination to become active. This is because the edge which leads to its destination is activated $d-1$ steps after the time the edge through which the packet arrived at the centre of the star was active. So, each of the $d = n-1$ packets occupies the centre of the star for at least $d-1$ steps and thus, $\Omega(dn)$ steps are required for the routing of this permutation on the star of degree d.

In the above routing problem the maximum degree of the tree is a function of the number of nodes in the tree. It is not difficult to construct a tree of constant degree d and a permutation for which the same bound applies. This is shown in Figure 1(b). Each subtree T_i, $0 \leq i \leq d-1$, has $(n-1)/d$ nodes and the packets in subtree T_i have destinations in subtree $T_{(i-1) \bmod d}$, $0 \leq i \leq d-1$.

3.2 The On-line Algorithm

In the description of the algorithm we assume that, at the end of each routing step, each node examines the packet it holds and if the packet was destined for that node it is consumed. Following, the consumption of the packet, if any, each node might inject a new packet into the routing.

Let T be an n-node tree of maximum degree d. The many-to-many on-line algorithm is as follows:

```
Algorithm On-Line-Tree-Routing($T$, $M$)
/* $M$ is the set of packets to be routed on tree $T = (V, E)$ */
  1. [Preprocessing] For each node $v \in V$ label the edges incident on $v$ with labels
     in $\{0, \cdots, d-1\}$, so that no two edges incident on $v$ have the same label.
  2. $t = 0$
  3. For each node $v \in V$ select a packet $p \in M$ (if any) with $orig(p) = v$ and inject
     it into the routing.
  4. While there are packets that haven't reached their destination do
     (a) For each edge $\{u, v\}$ with a label $l = t \bmod d$
             do Update($u, v$).
     (b) Consume packets that reached their destination.
     (c) Inject new packets (if there are any to be injected).
     (d) $t = t + 1$
```

Procedure *Update*(u, v) performs a swap of the packets at the endpoints of edge $\{u, v\}$ if and only if both packets will move closer to their destinations. In the description of the procedure, we assume that one packet is present at each endpoint. The procedure can be trivially extended to cover the case where none or only one packet is present at the endpoints of edge $\{u, v\}$. Consider any node $v \in V$ at time t. Then, by *packet*(v) we denote the packet $p \in M$ (if any) which resides in node v at time t.

```
Procedure Update($u, v$)
  1. $u' = dest_T(packet(u))$
  2. $v' = dest_T(packet(v))$
  3. if $dist_T(u, v') + dist_T(v, u') < dist_T(u, u') + dist_T(v, v')$ then
         swap the packets at the endpoints of $\{u, v\}$.
```

3.3 Analysis of Algorithm *On-Line-Tree-Routing*

The analysis of our on-line algorithm is based on reducing matching routing to hot-potato routing and then applying the general charging scheme that is used for the analysis of hot-potato routing algorithms. Consider the routing problem $\mathcal{R} = (T, M)$ which is routed by algorithm *On-Line-Tree-Routing*. Based on $\mathcal{R} = (T, M)$ and algorithm *On-Line-Tree-Routing*, we define a routing problem $\mathcal{R}' = (G_T, H)$ and the hot-potato Algorithm *On-Line-Simulation* such that, the number of steps required for the routing of problem $\mathcal{R} = (T, M)$ by algorithm *On-Line-Tree-Routing* is a function of the number of steps required for the routing of problem $\mathcal{R}' = (G_T, H)$ by Algorithm *On-Line-Simulation*.

Consider a tree T of maximum degree d and let each edge in T be labelled with an integer $i \in \{0, \cdots, d-1\}$, so that no two edges incident to the same node have the same label. We use T and the labels of its edges to construct a directed graph G_T as follows: For each node v of T, we create d nodes v_j, $j \in \{0, \cdots, d-1\}$, in G_T, and we say that these nodes of G_T *correspond* to node v of T. For each edge $\{u, v\}$ of T we create a node $\{u, v\}^i$ in G_T, where i is the label of $\{u, v\}$ in T. We say that this node of G_T *corresponds* to edge $\{u, v\}$ of T. For each edge $\{u, v\}$ in T with label i, we add the following four directed edges in G_T: $(u_i, \{u, v\}^i)$, $(\{u, v\}^i, u_{(i+1) \bmod d})$, $(v_i, \{u, v\}^i)$, $(\{u, v\}^i, v_{(i+1) \bmod d})$. Note that, if a node v in T has degree $d' < d$, not all labels in $\{0, \cdots, d-1\}$ appear at the edges incident to it. Consider such a node v and let l be a label that does not appear in an edge incident to v. Then we create a node $\{v\}^l$ in in G_T and we add the directed edges $(v_l, \{v\}^l)$, $(\{v\}^l, v_{(l+1) \bmod d})$. For an example of the construction of a graph G_T corresponding to a labelled tree T, see Figure 2.

3.3.1 Many-to-One Routing

For simplicity, we first analyse Algorithm *On-Line-Tree-Routing* for many-to-one routing problems. In the next section, we extend the analysis to many-to-many routing. So, assume that problem $\mathcal{R} = (T, M)$ is a many-to-one routing problem, that is, $|M| \leq n$ and for every pair of distinct packets p and $q \in M$ it holds that $orig(p) \neq orig(q)$.

We complete the construction of routing problem $\mathcal{R}' = (G_T, H)$ by describing how to construct the set of packets H based on the packets of set M. For each packet $p_m \in M$, we create a packet p_h in H and we set its origin and destination nodes as follows: Let $u = origin(p_m)$, $v = dest(p_m)$ and l be the label of the edge that is last in the shortest path from u to v in T (assume that $orig(p_m) \neq dest(p_m)$). Then, for packet p_h we set $origin(p_h) = u_0$ and $dest(p_h) = v_{(l+1) \bmod d}$.

Algorithm *On-Line-Simulation* is the hot-potato algorithm which we use for the routing of problem $\mathcal{R}' = (G_T, H)$. It specifies the rules that each of the nodes of graph G_T uses when it decides which packet to forward (if any) to each of its outgoing edges.

Algorithm *On-Line-Simulation*

Rules for nodes of G_T that correspond to nodes of T

[*On-line-node-1*] If the packet received in the previous step reached its destination it is consumed; otherwise, it is forwarded through the only out-going edge.

Rules for nodes of G_T that correspond to unused labels around nodes of T

[*On-line-label-1*] The packet received in the previous step is forwarded through the only out-going edge.

Rules for nodes of G_T that correspond to edges of T

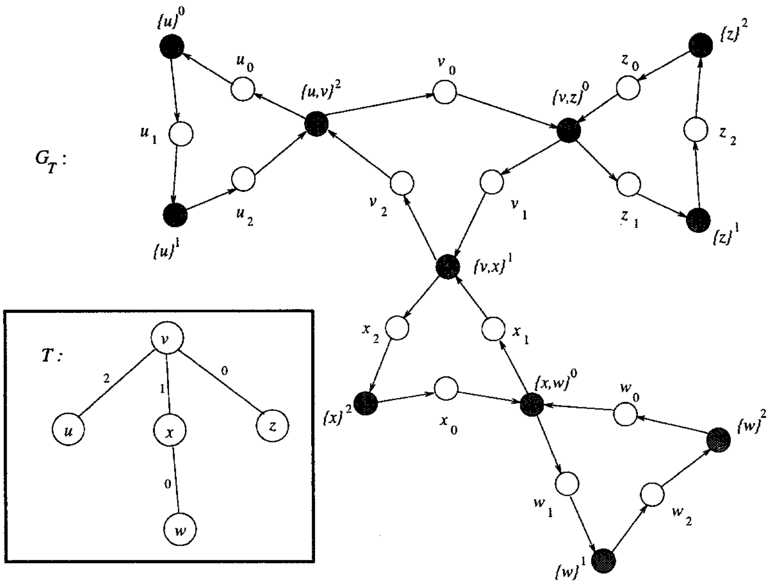

Fig. 2. Tree T and the corresponding graph G_T used in the analysis of Algorithm *On-Line-Tree-Routing.*

[*On-line-edge-1*] If there is only one packet at the node, the packet is forwarded to the edge that brings it closer to its destination.
[*On-line-edge-2*] In the case that there are two packets in the node, the decision is made as follows: Let $\{u,v\}^i$, $i \in \{0 \cdots d-1\}$, be the node under consideration. Let p_h be the packet that arrived from u_i and q_h be the packet that arrived from v_i. Moreover, let u' and v' be the nodes of T which correspond to $dest(p_h)$ and $dest(q_h)$, respectively. If $dist_T(u,v') + dist_T(v,u') < dist_T(u,u') + dist_T(v,v')$ then we forward p_h to $v_{(i+1) \bmod d}$ and q_h to $u_{(i+1) \bmod d}$; Otherwise, p_h is forwarded to $u_{(i+1) \bmod d}$ and q_h is forwarded to $v_{(i+1) \bmod d}$.

Lemma 2. *Let the many-to-one routing problem $\mathcal{R} = (T, M)$ be routed by Algorithm* On-Line-Tree-Routing *and the many-to-one routing problem $\mathcal{R}' = (G_T, H)$ by Algorithm* On-Line-Simulation. *Consider an arbitrary packet $p_m \in M$ and let packet $p_h \in H$ be the packet which corresponds to it. Then,*
(i) packet p_m is consumed at time c iff packet p_h is consumed at time $2c$, and
(ii) at time t packet p_m is at node u iff at time $2t$ packet p_h is at node $u_{t \bmod d}$, $t \leq c$.

Theorem 3. *Algorithm* On-Line-Tree-Routing *routes any many-to-one routing problem $\mathcal{R} = (T, M)$ in at most $d(k-1) + d \cdot dist$ routing steps, where d is the maximum degree of tree T, $k = |M|$ is the number of packets to be routed, and dist is the maximum distance that any packet in M has to travel in order to reach its destination.*

Proof. Based on Lemmata 1 and 2. □

3.3.2 Many-to-Many Routing

For the purposes of the analysis, we first route problem $\mathcal{R} = (T, M)$ by Algorithm *On-Line-Tree-Routing* and we observe for each individual packet the time at which it is injected into the routing. When the routing of $\mathcal{R} = (T, M)$ terminates, we are ready to fully specify problem $\mathcal{R}' = (G_T, H)$. For each packet $p_m \in M$ which was injected into the matching routing at time t, we create a packet p_h in H with $birth(p_h) = 2t$. The origin and the destination nodes of p_h are set as in the analysis of the many-to-one routing.

Lemma 4. *Consider the many-to-many routing problem $\mathcal{R} = (T, M)$ which is routed by Algorithm* On-Line-Tree-Routing *and the constructed dynamic routing problem $\mathcal{R}' = (G_T, H)$ which is routed by Algorithm* On-Line-Simulation. *Let p_m be an arbitrary packet in M and let p_h be its corresponding packet in H. If Algorithm* On-Line-Tree-Routing *injects packet p_m at time t then Algorithm* On-Line-Simulation *can inject packet p_h at time $2t$.*

Theorem 5. *Algorithm* On-Line-Tree-Routing *routes any many-to-many routing problem $\mathcal{R} = (T, M)$ in at most $d(k-1) + d \cdot dist$ routing steps, where d is the maximum degree of tree T, $k = |M|$ is the number of packets to be routed, and dist is the maximum distance that any packet in M has to travel in order to reach its destination.*

4 Off-line Routing

For our off-line routing algorithms we use some special forms of directed graphs whose underlying undirected structure is that of a tree. More specifically, by *in-tree* we refer to the directed graph that satisfies the following properties: i) its undirected version is a tree, ii) there is a single node of out-degree 0 that is designated as the *root* of the in-tree, iii) all other nodes have out-degree 1. By *1-loop in-tree* we refer to the directed graph that satisfies the following properties: i) its undirected version is a tree, ii) all nodes have out-degree 1, iii) there is a pair of adjacent nodes the outgoing edges of which form a loop, referred as the *1-loop* of the tree. Finally, a node with no incoming and no outgoing edges is referred to as an *isolated* node. Graph $G(T, t)$ in Figure 3 consists of two in-trees rooted at nodes e and f, respectively, one 1-loop in-tree with nodes a and b forming the 1-loop, and one isolated node i.e., node g.

Consider tree T at time t of the matching routing. Each node of the tree contains at most 1 packet which currently participates in the routing. We construct an auxiliary directed graph $G(T, t) = (V, E^t)$ which is used by our off-line algorithm to determine the set of edges that swap the packets at their endpoints during the next routing step. The directed edge (u, v) is in E^t if and only if at time t there is a packet p at node u and v is the first node in the shortest path from u to $dest(p)$ (of course, u and v are neighbours in T). Figure 3 shows the auxiliary graph obtained from tree T at time t, assuming that the location of each packet is as described in the figure. The out-degree of each node in graph $G(T, t)$ is at most 1 and thus $G(T, t)$ is a collection of isolated nodes, in-trees, and 1-loop in-trees.

```
Algorithm Off-Line-Tree-Routing(T, M)
/* $M$ is the set of packets to be routed on tree $T = (V, E)$ */
  1. $t = 0$
```

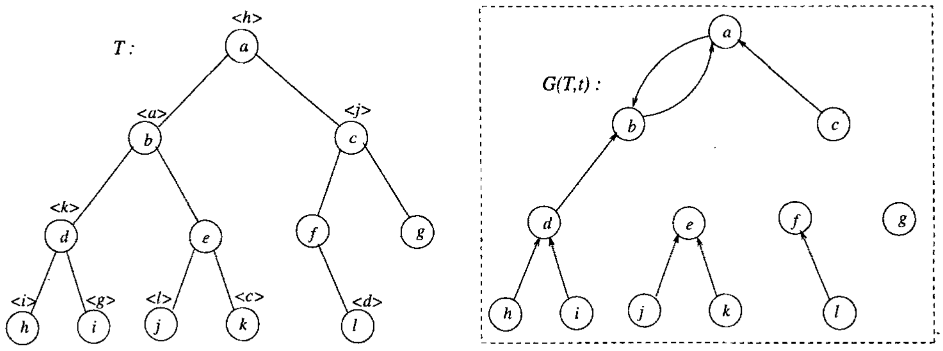

Fig. 3. Tree T at step t and the corresponding auxiliary graph $G(T, t)$.

2. For each node $v \in V$ select a packet $p \in M$ (if any) with $orig(p) = v$ and inject it into the routing.
3. While *there are packets that haven't reached their destination* do
   (a) Construct the auxiliary graph $G(T, t)$.
   (b) Denote by $\mathcal{S}$ be the set of tree edges which swap the packets at their endpoints during the next routing step. Insert into set $\mathcal{S}$:
   –One edge for each 1-loop in-tree. The edge is the one that corresponds to the 1-loop.
   –One edge for each in-tree. Out of the edges which enter the root of the in-tree, select the one which is emanating from the node of lowest order. The tree edge that is inserted in $\mathcal{S}$ is the one which corresponds to the selected edge of the in-tree.
   (c) Swap the packets at the endpoints of edges in $\mathcal{S}$.
   (d) Consume packets that have reached their destination.
   (e) Inject new packets whenever possible (if any are still to be injected into the routing).
   (f) $t = t + 1$

For example, based on the tree $G(T, t)$ of Figure 3 and assuming that the nodes of T are ordered lexicographically, the active edges which swap the packets at their endpoints are $\{a, b\}$, $\{e, j\}$, and $\{f, l\}$.

For the analysis of Algorithm *Off-Line-Tree-Routing* we again employ elements of hot-potato routing. For details see [12].

Theorem 6. *Algorithm* Off-Line-Tree-Routing *routes any many-to-one routing problem* $\mathcal{R} = (T, M)$ *in at most* $2(k-1) + dist$ *routing steps, where* $k = |M|$ *is the number of packets to be routed, and dist is the maximum distance that any packet in* M *has to travel in order to reach its destination.*

Krizanc and Zhang [10] independently showed that any many-to-one problem on an n-node tree can be solved under the matching routing model in at most $9n$ steps and they posed the question whether it is possible to complete the routing of any many-to-one pattern in less than $4n$ steps. Algorithm *Off-Line-Tree-Routing* dramatically improves upon the result of Krizanc and Zhang and answers their question to the affirmative.

Theorem 7. *Algorithm* Off-Line-Tree-Routing *routes any many-to-many routing problem* $\mathcal{R} = (T, M)$ *in at most* $2(k-1) + dist$ *routing steps, where* $k = |M|$ *is the number of packets to be routed, and dist is the maximum distance that any packet in* M *has to travel in order to reach its destination.*

References

1. Alon, Chung, and Graham. Routing permutations on graphs via matchings. *SIAM Journal on Discrete Mathematics*, 7:513–530, 1994.
2. P. Baran. On distributed communication networks. *IEEE Trans. on Commun. Systems*, CS-12:1–9, 1964.
3. A. Borodin, Y. Rabani, and B. Schieber. Deterministic many-to-many hot potato routing. Technical Report RC 20107 (6/19/95), IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY 10598, June 1995.
4. J.T. Brassil and R.L. Cruz. Bounds on maximum delay in networks with deflection routing. In *Proceedings of the 29th Allerton Conference on Communication, Control and Computing*, pages 571–580, 1991.
5. U. Feige and P. Raghavan. Exact analysis of hot-potato routing. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science (Pittsburgh, Pennsylvania, October 24–27, 1992)*, pages 553–562, Los Alamitos-Washington-Brussels-Tokyo, 1992. IEEE Computer Society Press.
6. N. Haberman. Parallel neighbor-sort (or the glory of the induction principle). Technical Report AD-759 248, National Technical Information Service, US Department of Commerce, 5285 Port Royal Road, Springfieldn VA 22151, 1972.
7. B. Hajek. Bounds on evacuation time for deflection routing. *Distributed Computing*, 5(1):1–6, 1991.
8. P. Høyer and K.S. Larsen. Permutation routing via matchings. Technical Report 16, Dept of Mathematics and Computer Science, Odense University, June 1996.
9. C. Kaklamanis, D. Krizanc, and S. Rao. Hot-potato routing on processor arrays. In *Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA'93 (Velen, Germany, June 30 – July 2, 1993)*, pages 273–282, New York, 1993. ACM SIGACT, ACM SIGARCH, ACM Press.
10. D. Krizanc and L. Zhang. Packet routing via matchings. *Unpublished manuscript*, 1996.
11. Newman and Schuster. Hot-potato algorithms for permutation routing. *IEEE Transactions on Parallel and Distributed Systems*, 6(11):1168–1176, November 1995.
12. G. Pantziou, A. Roberts, and A. Symvonis. Many-to-many routing on trees via matchings. Technical Report TR-507, Basser Dept of Computer Science, University of Sydney, July 1996. Available from ftp://ftp.cs.su.oz.au/pub/tr/TR96_507.ps.Z.
13. A. Roberts, A. Symvonis, and L. Zhang. Routing on trees via matchings. In *Proceedings of the Fourth Workshop on Algorithms and Data Structures (WADS'95), Kingston, Ontario, Canada*, pages 251–262. Springer-Verlag, LNCS 955, aug 1995. Also TR 494, January 1995, Basser Dept of Computer Science, University of Sydney. Available from ftp://ftp.cs.su.oz.au/pub/tr/TR95_494.ps.Z.
14. L. Zhang. Personal communication, 1996.